Finding Pareto Optimal Groups: Group-based Skyline

Jinfei Liu
Emory University
jinfei.liu@emory.edu

Li Xiong
Emory University
lxiong@emory.edu

Jian Pei
Simon Fraser University
jpei@cs.sfu.ca

Jun Luo
Lenovo; CAS
jun.luo@siat.ac.cn

Haoyu Zhang
Emory University
haoyu.zhang@emory.edu

ABSTRACT

Skyline computation, aiming at identifying a set of skyline points that are not dominated by any other point, is particularly useful for multi-criteria data analysis and decision making. Traditional skyline computation, however, is inadequate to answer queries that need to analyze not only *individual* points but also *groups* of points. To address this gap, we generalize the original skyline definition to the novel group-based skyline (G-Skyline), which represents Pareto optimal groups that are not dominated by other groups. In order to compute G-Skyline groups consisting of k points efficiently, we present a novel structure that represents the points in a directed skyline graph and captures the dominance relationships among the points based on the first k skyline layers. We propose efficient algorithms to compute the first k skyline layers. We then present two heuristic algorithms to efficiently compute the G-Skyline groups: the point-wise algorithm and the unit group-wise algorithm, using various pruning strategies. The experimental results on the real NBA dataset and the synthetic datasets show that G-Skyline is interesting and useful, and our algorithms are efficient and scalable.

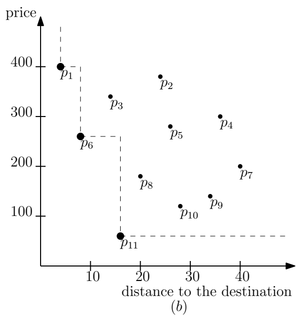
1. INTRODUCTION

Skyline, also known as *Maxima* in computational geometry or *Pareto* in business management field, is important for many applications involving multi-criteria decision making. The skyline of a set of multi-dimensional data points consists of the points for which no other point exists that is better in at least one dimension and at least as good in every other dimension.

Assume that we have a dataset of n points, referred to as P. Each point p of d real-valued attributes can be represented as a d-dimensional point $(p[1], p[2], ..., p[d]) \in \mathbb{R}^d$ where $p[i]$ is the i-th attribute of p. Given two points $p = (p[1], p[2], ..., p[d])$ and $p' = (p'[1], p'[2], ..., p'[d])$ in $\mathbb{R}^d$, p dominates p' if for every i, $p[i] \le p'[i]$ and for at least one i, $p[i] < p'[i]$ $(1 \le i \le d)$. Given the set of points P, the skyline is defined as the set of points that are not dominated by any other point in P. In other words, the skyline represents the *best points* or Pareto optimal solutions from the dataset since the points within the skyline cannot dominate each other.

hotel	distance	price
p_1	4	400
p_2	24	380
p_3	14	340
p_4	36	300
p_5	26	280
p_6	8	260
p_7	40	200
p_8	20	180
p_9	34	140
p_{10}	28	120
p_{11}	16	60

(a)



(b)

Figure 1: A skyline example of hotels.

Figure 1(a) illustrates a dataset $P = \{p_1, p_2, ..., p_{11}\}$, each representing a hotel with two attributes: the distance to the destination and the price. Figure 1(b) shows the corresponding points in the two dimensional space where the x and y coordinates correspond to the attributes of distance to the destination and price, respectively. We can see that $p_3(14, 340)$ dominates $p_2(24, 380)$ as an example of dominance. The skyline of the dataset contains p_1, p_6, and p_{11}. Suppose the organizers of a conference need to reserve *one* hotel considering both distance to the conference destination and the price for participants, the skyline offers a set of best options or Pareto optimal solutions with various tradeoffs between distance and price: p_1 is the nearest to the destination, p_{11} is the cheapest, and p_6 provides a good compromise of the two factors. p_8 will not be considered as p_{11} is better than p_8 in both factors.

Motivation. While the skyline definition has been extended with different variants and the skyline computation problem for finding the skyline of a given dataset has been studied extensively in recent years, most existing works focus on skyline consisting of *individual* points. One important problem that has been surprisingly neglected to the large extent is the need to find groups of points that are not dominated by others as many real-world applications may require the selection of *a group of* points.

Hotels Example. Consider our hotel example again, suppose the organizers need to reserve *a group of* hotels (instead

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 42nd International Conference on Very Large Data Bases, September 5th - September 9th 2016, New Delhi, India.
Proceedings of the VLDB Endowment, Vol. 8, No. 13
Copyright 2015 VLDB Endowment 2150-8097/15/09.

of one) considering both distance to the conference destination and the price for participants. In contrast to the traditional skyline problem which finds Pareto optimal solutions where each solution is a single point, we are interested in finding Pareto optimal solutions where each solution is a group of points. One may use the traditional skyline definition, and return all subsets from the skyline points p_1, p_6, and p_{11}. If the desired group size is 2, group $\{p_1, p_6\}$, $\{p_1, p_{11}\}$, and $\{p_6, p_{11}\}$ can be returned. However, we show that this definition does not capture all the best groups. For example, $\{p_{11}, p_{10}\}$ should clearly be considered a Pareto optimal group to users who use price as the main criterion, e.g. PhD students with low travel budget, since p_{11} provides the best price and p_{10} the second best price. Note that p_{10} is only second best to p_{11} which is also part of the group, hence no other groups are better than this group in terms of price. As another example, $\{p_6, p_3\}$ also presents a Pareto optimal group, as both p_6 and p_3 provide a good tradeoff and no other groups are better than this group considering both price and distance. On the other hand, group $\{p_3, p_8\}$ is not a best group because $p_{11}(p_6)$ is better than $p_8(p_3)$, i.e., group $\{p_3, p_8\}$ is dominated by group $\{p_6, p_{11}\}$.

Table 1: Top five players on Attribute PTS.

Player	PTS	REB	AST	STL	BLK
Michael Jordan	33.4	6.4	5.7	2.1	0.9
Anthony Davis	30.5	8.5	2	1.5	3
Kyrie Irving	30	3	2	1	1
Allen Iverson	29.7	3.8	6	2.1	0.2
Jerry West	29.1	5.6	6.3	0	0
...	...	...	...	...	...

NBA Example. Consider another real example with NBA players. Table 1 shows the top five players on attribute PTS (Points). For other attributes, please see the experimental section for detailed explanations. Suppose the coaches of NBA teams need to choose five players to compose a team. While the traditional skyline will compute *best players* that are not dominated by other players, we need to compute *best teams* that are not dominated by other teams. For example, a coach may prefer PTS as the main selection criteria when building a team in order to maximize the overall points that can be earned by the team. In this case, the top five players on PTS, {Michael Jordan, Anthony Davis, Kyrie Irving, Allen Iverson, Jerry West}, should be considered a best team. However, if we only build groups from skyline players, this team will not be captured because Kyrie Irving is not a skyline player being dominated by Anthony Davis. In essence, taking only skyline players will not capture those teams which may include non-skyline players who are only dominated by another player in the team but are not dominated by any other players outside the team. In summary, we argue that there is a need to define a group skyline notion for group-based decision making such that we can find Pareto optimal groups.

Contributions. In this paper, we formally define a novel group-based skyline, *G-Skyline*, for finding Pareto optimal groups. In order to find the best groups, i.e., groups not dominated by other groups, we will first define the dominance relationship between groups, group dominance. Given two different groups G and G' with k points, we say G g-dominates G', if for any point p'_i in G', we can find a distinct point p_i in G, such that p_i dominates p'_i or $p_i = p'_i$, and for at least one i, p_i dominates p'_i. The G-Skyline are those groups that are not g-dominated by any other group with same size. Intuitively, if we consider the points in each group as a set of dimensions orthogonal to the attributes of each point, the definition of G-Skyline groups with the group dominance is in spirit similar to skyline definition, in that a group is a skyline group if no permutation of any other group exists that is better for at least one point and at least as good for every other point.

G-Skyline not only captures groups of points from traditional skyline points but also groups that may contain non-skyline points. Back to our hotel example, $\{p_{11}, p_{10}\}$ is a G-Skyline group as we discussed earlier even though p_{10} is not a skyline point. Group $\{p_6, p_3\}$ is also a G-Skyline. On the other hand, group $\{p_1, p_3\}$ is not as it is dominated by $\{p_1, p_6\}$. Group $\{p_3, p_8\}$ is also not as it is dominated by $\{p_6, p_{11}\}$. In summary, the G-Skyline in this example consist of all groups composed of skyline points, $\{p_1, p_6\}$, $\{p_1, p_{11}\}$, $\{p_6, p_{11}\}$, as well as groups that contain non-skyline points, $\{p_6, p_3\}$, $\{p_{11}, p_8\}$, and $\{p_{11}, p_{10}\}$.

It's non-trivial to solve G-Skyline problem efficiently. To find k-point G-Skyline groups from n points, there can be $\binom{n}{k}$ different possible groups. Unfortunately, the G-Skyline problem is significantly different from the traditional skyline problem, to the extent that algorithms for the latter are inapplicable. A brute force solution is to enumerate all $\binom{n}{k}$ possible groups, then for each group, to compare it with all other groups to determine whether it cannot be dominated. So there are $\binom{n}{k}^2$ comparisons. For each comparison, there are $k!$ possible permutations of the points, and for each permutation, it requires k comparisons. Therefore, the time complexity is in the order of $O(\binom{n}{k}^2 \times k! \times k)$.

In this paper, we present a novel structure that represents the points in a directed skyline graph and captures all the dominance relationship among the points based on the notion of *skyline layers*. Using the directed skyline graph, the G-Skyline problem can be formulated as the classic search problem in a set enumeration tree. We exploit the properties of G-Skyline groups and propose two algorithms with efficient pruning strategies to compute G-Skyline groups.

We briefly summarize our contributions as follows.

- For the first time, we generalize the original skyline definition (for individual points) to permutation group-based skyline (for groups) which is useful for finding Pareto optimal groups in practical applications.

- We present a novel structure for finding k-point G-Skyline groups by representing the points as a directed skyline graph based on the first k skyline layers. This directed skyline graph is significant as we show that we only need the points in the first k skyline layers (k is far less than n in the usual case) rather than the entire n points to compute k-point G-Skyline. We design efficient algorithms for computing the first k skyline layers with time complexity $O(n \log n)$ for two- and $O(n \log n + n\mathbb{S}_k)$ for higher-dimensional spaces, where $\mathbb{S}_k$ is the number of points in the first k skyline layers. This can be also of independent value and used as a preprocessing step for other skyline algorithms.

- Given the directed skyline graph, we present two efficient algorithms: the point-wise and the unit group-wise algorithms, to efficiently compute G-Skyline groups. We introduce a novel notion of unit-group for each point which represents the minimum number of points that have to be included with the point in a G-Skyline. Both algorithms employ efficient pruning strategies exploiting G-Skyline properties.

- We conduct comprehensive experiments on real and synthetic datasets. The experimental results show that G-Skyline is interesting and useful, and our proposed algorithms are efficient and scalable.
- We also briefly discuss two variants of G-Skyline definition: One is AG-Skyline based on a more restrictive all-permutation group dominance. The other one is PG-Skyline based on a less restrictive partial group dominance, which can address the potential problem of large number of output groups of G-Skyline.

Organization. The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 introduces our G-skyline definitions as well as their properties. The algorithms of constructing directed skyline graph are shown in Section 4. Two algorithms for finding G-Skyline groups based on the directed skyline graph are discussed in Section 5. We report the experimental results and findings for performance evaluation in Section 6. Section 7 discusses two extensions to our work. Section 8 concludes the paper.

2. RELATED WORK

The problem of computing skyline (Maxima) is a fundamental problem in computational geometry field because the skyline is an interesting characterization of the boundary of a set of points. The skyline computation problem was firstly studied in computational geometry [17] which focused on worst-case time complexity. [15, 21] proposed output-sensitive algorithms achieving $O(n \log v)$ in the worst-case where v is the number of skyline points which is far less than n in general. Several works [2, 3, 5, 8] in both computational geometry and database fields focused on how to achieve the best average-case time complexity. For a detailed survey both for worst-case and average-case, please see [12].

Since the introduction of the skyline operator by Börzsönyi et al. [5], skyline has been extensively studied in the database field. Many algorithms are proposed in the context of relational query engine and external memory model, for example, [12, 30]. Based on the traditional skyline definition, [1, 16] studied the parallel algorithms for skyline.

Many works also studied extensions or variants of the classical skyline definitions. Papadias et al. [25] studied *group-by skyline* which groups the objects based on their values in one dimension and then computes the skyline for each group, and *k-skyband* which computes objects dominated by at most k objects (the case $k = 0$ corresponds to the conventional skyline) based on individual dominance relationship. Skyline in subspace, i.e., a subset of the dimensions or points, was studied in [6, 27, 28, 32]. [10, 33] discussed the reverse skyline problem which is similar to the reverse k-nearest neighbor problem. [9] presented skyline-based statistical descriptors for capturing the distributions over pairs of dimensions. Some works defined and studied the skyline on different data types/domains. For example, [29] and [7] studied the spatial skyline and a more general metric skyline, respectively. [13] proposed the skyline for moving objects. [26, 35, 19, 11, 22] studied the skyline problem for uncertain data.

The most related works to our group-based skyline are [18, 14, 24, 34]. [18, 14, 34] formulated and investigated the problem of computing skyline groups. However, the notion of dominance between groups in these works is defined by the dominance relationship between an "aggregate" or "representative" point of each group. More specifically, they calculate for each group a single aggregate point, whose attribute values are aggregated over the corresponding attribute values of all points in the group. The groups are then compared by their aggregate points using traditional point dominance. While many aggregate functions can be considered in calculating aggregate points, they focus on several functions commonly used in database applications, such as, SUM, MIN, and MAX. In addition to the fact that it is difficult to choose a good or meaningful function, more importantly, it will not capture all the Pareto optimal groups. This is essentially similar to the multi-objective optimization or multi-attribute skyline problem where an aggregate function, such as weighted average, can be used to combine the multiple criteria to find a single optimal solution, but it also fails to capture all the Pareto optimal solutions.

In fact, the result of skyline groups under SUM dominance [18, 14, 34] is a subset of our G-Skyline groups. If group G is dominated by G' in G-Skyline definition, then G must be dominated by G' under SUM function, but not vice versa. Consider our hotel example, group $\{p_1, p_{11}\}$ dominates $\{p_3, p_6\}$ based on SUM function. However, we cannot conclude group $\{p_1, p_{11}\}$ is better than $\{p_3, p_6\}$ because the assumption of skyline is that we do not know users' attribute weights in advance. For users who consider both distance and price in their selection criteria, they may prefer group $\{p_3, p_6\}$, because $\{p_3, p_6\}$ provides a good compromise of distance and price, and neither p_1 or p_{11} can dominate p_3 or p_6. Hence, some Pareto optimal solutions are not captured by SUM dominance.

The work in [24] also defines a group dominance notion. However, their definition is based on the uncertain skyline definition by Pei et al. [26]. In our work, we define a deterministic dominance relationship between two groups in order to find "optimal" groups of objects.

3. G-SKYLINE DEFINITIONS

In this section, we introduce our G-Skyline definition and related concepts as well as their properties which will be used in our algorithm design. For reference, a summary of notations is given in Table 2.

Table 2: The summary of notations.

Notation	Definition
$P \backslash layer_i$	points in P but not in $layer_i$
$p \preceq p'$	p dominates or equals to p'
G-Skyline(i)	G-Skyline group with i points
$p_i.layer$	the skyline layer of p_i
$\lvert S \rvert_{p(u)}$	the point(unit group) size of set S

Definition 1. (**Skyline**). Given a dataset P of n points in d-dimensional space. Let p and p' be two different points in P, p dominates p', denoted by $p \prec p'$, if for all i, $p[i] \leq p'[i]$, and for at least one i, $p[i] < p'[i]$, where $p[i]$ is the i^{th} dimension of p and $1 \leq i \leq d$. The skyline points are those points that are not dominated by any other point in P.

G-Skyline. The key of skyline is that it consists of all the "best" points that are not dominated by other points. While a linear weighted sum function can be used to combine all the attribute values of each point as a scoring function to find the best points, the relative preferences (weights) for different attributes are not known in advance. The skyline essentially covers all the best points on all linear functions. Following this notion, in order to find all the "best" groups of points that are not dominated by other groups, we introduce group dominance definition as follows.

Definition 2. (**Group Dominance**). Given a dataset P of n points in a d-dimensional space. Let $G = \{p_1, p_2, ..., p_k\}$ and $G' = \{p'_1, p'_2, ..., p'_k\}$ be two different groups with k points of P, we say group G **g-dominates** group G', denoted by $G \prec_g G'$, if we can find two permutations of the k points for G and G', $G = \{p_{u_1}, p_{u_2}, ..., p_{u_k}\}$ and $G' = \{p'_{v_1}, p'_{v_2}, ..., p'_{v_k}\}$, such that $p_{u_i} \preceq p'_{v_i}$ for all i $(1 \leq i \leq k)$ and $p_{u_i} \prec p'_{v_i}$ for at least one i.

Given the group dominance definition, we define group-based skyline, G-Skyline, as follows.

Definition 3. (**G-Skyline**). The k-point G-Skyline consists of those groups with k points that are not g-dominated by any other group with same size.

EXAMPLE 1. *Consider the dataset in Figure 1 and $k = 3$. For group $G = \{p_8, p_{10}, p_{11}\}$ and group $G' = \{p_4, p_5, p_7\}$, G g-dominates G' because we can find two permutations, $G = \{p_8, p_{10}, p_{11}\}$ and $G' = \{p_5, p_4, p_7\}$ such that $p_8 \prec p_5$, $p_{10} \prec p_4$, and $p_{11} \prec p_7$. Therefore, $G' = \{p_4, p_5, p_7\}$ is not a G-Skyline group. G is one of the G-Skyline groups as no other group with 3 points can g-dominate G.*

Next, we present a few properties of G-Skyline groups and related concepts that will be used in our algorithm design for computing G-Skyline groups.

PROPERTY 1. *(**Asymmetry**). Give two groups G and G' with same size. If $G \prec_g G'$, then $G' \not\prec_g G$.*

PROPERTY 2. *(**Transitivity**). Given three groups G_1, G_2, and G_3 with same size. If $G_1 \prec_g G_2$ and $G_2 \prec_g G_3$, then $G_1 \prec_g G_3$.*

LEMMA 1. *A point in a G-Skyline group cannot be dominated by a point outside the group.*

PROOF. By contradiction, assume a point p_i in a G-Skyline group G is dominated by a point p_j outside the group. If we use p_j to replace p_i in G, the new group will g-dominate G since p_j dominates p_i and all the other points are the same, which contradicts the G-Skyline definition. □

Skyline Layers. Motivated by Lemma 1, we present a structure representing the points and their dominance relationships based on the notion of skyline layers. A formal definition is presented as follows.

Definition 4. (**Skyline Layers**). Given a dataset P of n points in a d-dimensional space. The set of skyline layer $layer_1$ contains the skyline points of P, i.e., $layer_1 = skyline(P)$. The set of $layer_2$ contains the skyline points of $P \backslash layer_1$, i.e., $layer_2 = skyline(P \backslash layer_1)$. Generally, the set of $layer_j$ contains the skyline points of $P \backslash \bigcup_{i=1}^{j-1} layer_i$, i.e., $layer_j = skyline(P \backslash \bigcup_{i=1}^{j-1} layer_i)$. The above process is repeated iteratively until $P \backslash \bigcup_{i=1}^{j-1} layer_i = \emptyset$.

An example of skyline layers of Figure 1 is shown in Figure 2. It is easy to see from Definition 4 that for a point p, if there is no point in $layer_{i-1}$ that can dominate p, p should be in $layer_{i-1}$ or a lower layer.

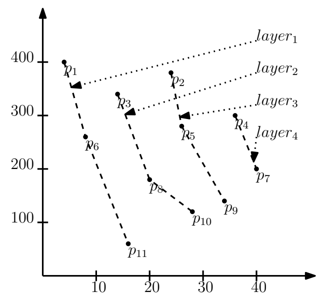


Figure 2: Skyline layers.

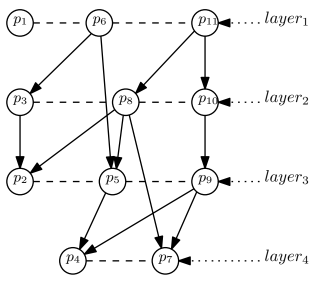


Figure 3: Directed skyline graph.

PROPERTY 3. *For a point p in $layer_i$, where $2 \leq i \leq l$ and l is the maximum layer number, there must be at least one point in $layer_j$ $(1 \leq j \leq i - 1)$ that dominates p.*

We then make an observation that in order to compute k-point G-Skyline groups, we only need to examine the points from the first k skyline layers. We formally present the theorem below.

THEOREM 1. *If a group $G = \{p_1, p_2, ..., p_k\}$ is a k-point G-Skyline group, then all points in G belong to the first k skyline layers.*

PROOF. We prove by contradiction. Assume a point of G is from the j^{th} skyline layer where $j \geq (k + 1)$. Without loss of generality, we assume this point is p_k. Since there are k points in G, there is at least one layer $layer_i, 1 \leq i \leq k$, that has no point in G. According to Property 3, p_k should be dominated by at least one point in $layer_i$, denoted by t. Then it is easy to see that group $G' = \{p_1, p_2, .., p_{k-1}, t\}$ by replacing p_k in G with t can dominate group $G = \{p_1, p_2, ..., p_{k-1}, p_k\}$. Then group G is not a G-Skyline group which is a contradiction. □

We note that the skyline layers are closely related to the k-skyband [25] we discussed in the related work. In fact, $(k - 1)$-skyband is the subset of the points in the first k skyline layers. We can alternatively compute k-point G-Skyline groups from the $(k-1)$-skyband, as versus the entire set of n points. The reason we use skyline layers instead of skyband in our preprocessing is that we can design efficient algorithms to compute the skyline layers as we will show in Section 4 and we can leverage the layers to compute skyline groups efficiently as we show in Section 5.

Directed Skyline Graph. We now present a definition of directed skyline graph, a data structure we use to represent the points from the first k skyline layers as well as their dominance relationships, in order to compute k-point G-Skyline groups.

Definition 5. (**Directed Skyline Graph (DSG)**). A directed skyline graph is a graph where a node represents a point and an edge represents a dominance relationship. Each node has a structure as follows.

$$[layer\ index, point\ index, parents, children]$$

where layer index ranging from 1 to k indicates the skyline layer that the point lies on, point index ranging from 0 to $\mathbb{S}_k - 1$ uniquely identifies the point and $\mathbb{S}_k$ is the number of points in the first k skyline layers, parents include all the points that dominate this point, and children include all the points that are dominated by the point.

EXAMPLE 2. *Figure 3 shows the DSG corresponding to the skyline layers in Figure 2. Note that $p_6 \prec p_5$ and $p_5 \prec p_4$ imply $p_6 \prec p_4$. For visualization clarity, we omit all indirect dominance edges such as $p_6 \prec p_4$.*

LEMMA 2. *Given a point p, if p is in a G-Skyline group, p's parents must be included in this G-Skyline group.*

PROOF. By Lemma 1, a point p in a G-Skyline group cannot be dominated by a point outside the group, i.e., all the points that dominate p must be included in this G-Skyline group. □

Verification of G-Skyline. Motivated by Lemma 2, we define a concept of Unit Group and then formally state a theorem for verifying whether a group is a G-Skyline group based on unit group.

Definition 6. (**Unit Group**). Given a point p in DSG, p and its parents form the unit group for p.

EXAMPLE 3. *The unit group of p_5 in Figure 3, denoted by u_5, contains p_5 and its parents p_6, p_{11}, p_8. Thus, $u_5 = \{p_6, p_{11}, p_8, p_5\}$.*

Based on Lemma 2, we have the verification of G-Skyline theorem as follows.

THEOREM 2. *(**Verification of G-Skyline**). Given a group $G = \{p_1, p_2, ..., p_k\}$, it is a G-Skyline group, if its corresponding unit group set $S = u_1 \cup u_2 \cup ... \cup u_k$ contains k points, i.e., $|S|_p = k$.*

This theorem is significant because given a group G, in order to check whether it is a G-Skyline group, we do not need to compare G with all other candidate groups any more. Instead, we only need to check whether its corresponding unit group set S has k points.

4. CONSTRUCTING DIRECTED SKYLINE GRAPH

In this section, we first present our algorithms for computing the first k skyline layers in two- and higher-dimensional space, and then briefly discuss how to construct the DSG based on skyline layers. In next section, we will present our algorithms for finding G-Skyline groups using DSG.

A straightforward way of computing skyline layers is to iteratively compute (and remove) the skyline points of each layer using any $O(n \log n)$ time complexity skyline algorithms [17]. Since a dataset may exhibit a linear number of layers, this leads to an $O(n^2 \log n)$ worst-case running time. Existing work proposed space-efficient algorithms for computing all skyline layers simultaneously with $O(n \log n)$ time complexity for two dimensions [4] and $O(n^2)$ for higher dimensions [23]. In this paper, we present an efficient $O(n \log n)$ algorithm for two dimensional space based on the ideas briefly mentioned in [4] (they did not provide any algorithm details as their focus is on designing in-place algorithms). In addition, since we only need the first k skyline layers for computing k-point G-Skyline groups, as we have shown in Theorem 1, we present more efficient output-sensitive algorithms with $O(n + \mathbb{S}_k \log k)$ time complexity for two- and $O(n\mathbb{S}_k)$ for higher-dimensional space after the points are sorted.

Computing Skyline Layers for Two Dimensions. For two dimensional space, the main intuition of the algorithm is motivated by the monotonic property of skyline points in two dimensional space, that is, if we sort the skyline points with increasing x-coordinate, their y-coordinates decrease monotonically, since they cannot dominate each other. This applies to each skyline layer as shown in Figure 2. We refer to the point with minimum y-coordinate in $layer_i$ as the *tail point* of $layer_i$. We derive the following two properties for skyline layers in a two dimensional space which will motivate our algorithm design.

PROPERTY 4. *Given a skyline layer $layer_i$ with its tail point denoted as p_{layer_i}, and a point p, if $p[x] \geq p_{layer_i}[x]$ and p is not dominated by p_{layer_i}, then p cannot be dominated by any other point in $layer_i$.*

PROOF. For a point p with $p[x] \geq p_{layer_i}[x]$, if it is not dominated by p_{layer_i}, then we have $p[y] < p_{layer_i}[y]$. Because p_{layer_i} has the smallest value on y-coordinate in $layer_i$, then all other points in $layer_i$ cannot dominate p. □

PROPERTY 5. *Given l layers for the n points in P with their tail points denoted as p_{layer_1}, p_{layer_2}, ..., p_{layer_l}, the y-coordinates of those points are in ascending order, i.e., $p_{layer_1}[y] \leq p_{layer_2}[y] \leq ... \leq p_{layer_l}[y]$.*

PROOF. We prove by contradiction. Suppose $p_{layer_i}[y] < p_{layer_{i-1}}[y]$. Since $p_{layer_{i-1}}$ is the tail point in $layer_{i-1}$, we know that the y-coordinates of all the points in $layer_{i-1}$ are larger than $p_{layer_{i-1}}[y]$, hence no points of $layer_{i-1}$ can dominate the tail point of $layer_i$. This is contradictory to the skyline layer definition. □

As an example for Property 4 in Figure 2, we can see that $layer_1$ has tail point p_{11}, given a point p with x-coordinate greater than p_{11}, if p is not dominated by p_{11}, then p cannot be dominated by p_1 and p_6. For Property 5, we can see that the tail points p_{11}, p_{10}, p_9, and p_7 are in ascending order on their y-coordinates.

Algorithm 1: Skyline layers algorithm in two-Ds.

```
input : a set of $n$ points in two dimensional space.
output: $l$ skyline layers.
1  If the points are not sorted already, sort the $n$ points on the first
   dimension in ascending order $P = \{p_{u_1}, p_{u_2}, ..., p_{u_n}\}$;
2  $p_{u_1}.layer = 1$;
3  $maxlayer = 1$;
4  tail point of $layer_1 = p_{u_1}$;
5  for $i = 2$ to $n$ do
6      if the tail point of $layer_1$ cannot dominate $p_{u_i}$ then
7          $p_{u_i}.layer = 1$;
8          tail point of $layer_1 = p_{u_i}$;
9      else if the tail point of $layer_{maxlayer}$ dominate $p_{u_i}$ then
10         $p_{u_i}.layer = ++maxlayer$;
11         tail point of $layer_{maxlayer} = p_{u_i}$;
12     else
13         use binary search to find $layer_j$ $(1 < j \leq maxlayer)$
           such that the tail point of $layer_j$ cannot dominate $p_{u_i}$
           and the tail point of $layer_{j-1}$ dominates $p_{u_i}$;
14         $p_{u_i}.layer = j$;
15         tail point of $layer_j = p_{u_i}$;
```

Based on the above properties, our key idea of the algorithm, shown in Algorithm 1, is to sort all the points in ascending x-coordinates if they are not sorted already and process them in that sequence by either adding them into an existing layer it belongs to or starting a new layer. Line 2-4 adds the first point (with minimum x value) into the first skyline layer. For each new point p_{u_i}, its x-coordinate is larger than all the points in existing layers. Based on Property 4, we only need to compare point p_{u_i} with the (current) tail point of each layer to determine if it cannot be dominated by any of the points in that layer. Based on Property 5, the tail points are sorted by y-coordinates in ascending order so we can perform a binary search to quickly find the layer that the point belongs to. If the tail point of the first layer cannot dominate p_{u_i} (line 6), we insert the point to the first layer. If the tail point of the last layer dominates p_{u_i} (line 9), we add a new layer for the point. Line 13 performs such a binary search and finds $layer_j$ to insert the point, i.e., the tail point of $layer_j$ cannot dominate p_{u_i} but the tail point of $layer_{j-1}$ dominates p_{u_i}. Once the point is inserted, it becomes the new tail point of that layer.

Since we just need the points in the first k skyline layers (Theorem 1), we can slightly modify the algorithm as follows. Once the k^{th} layer is established, for each of the remaining points p, we can compare p with the tail point of the k^{th} layer. If p is dominated by it, it means p lies outside the first k layers, and we can drop p directly. If not, we can then use binary search on the first k layers.

Running time. For each point in the first k skyline layers, we need at most $O(\log k)$ time to determine its layer because we only need to maintain k layers. This part costs $O(\mathbb{S}_k \log k)$. For those points not in the first k layers, we only need to compare it with the tail point of the k^{th} layer. Therefore, the algorithm requires $O(n + \mathbb{S}_k \log k)$ time in total for computing the first k skyline layers in two dimensional space after the points are sorted in one dimension.

Higher dimensional space. For a higher dimensional space, we can use an algorithm similar to Algorithm 1. It processes each point in order and finds an existing skyline layer to insert it or starts a new layer. However, the difference is that Properties 4 and 5 do not hold for the higher dimensional case anymore. So in order to find an existing skyline layer the point belongs to, we need to compare the point with all existing points, as versus only the tail points in each layer in two-dimensional case. For each point, we need at most $O(\mathbb{S}_k)$ time to determine its position because there are at most $\mathbb{S}_k$ points in the first k layers. Therefore, the algorithm for computing the first k skyline layers in higher dimensional space requires $O(n\mathbb{S}_k)$ time after the data are sorted in one dimension.

Constructing Directed Skyline Graph. Once we build the skyline layers, we can build a DSG to capture all the dominance relationships between the layers which will then be used to compute G-Skyline groups. Building DSG using the skyline layers is straightforward: the points are processed in the order of their skyline layers. For each point p_i, we scan all points in the previous layers and find those points that dominate p_i, add p_i to their children list, and add those points that dominate p_i as p_i's parents. It is easy to see such an algorithm takes $O({\mathbb{S}_k}^2)$ time.

5. FINDING G-SKYLINE GROUPS

In this section, we present our algorithms for efficiently finding G-Skyline groups given the DSG built from the first k skyline layers. We first present a point-wise algorithm which builds G-Skyline groups from points (adding one point at a time), then present a unit group-wise algorithm which builds G-Skyline groups from unit groups (adding one unit group at a time). Before beginning the discussion of the two algorithms, we first show a preprocessing step similar to that in [18, 14, 34] to further prune points from the first k skyline layers.

Preprocessing. Theorem 2 shows that a k-point group is a G-Skyline group, if for each point p_i in the group, its parents are also in the group, i.e., the unit group u_i is a subset of the group. Therefore, for a point p_i, if the point size of its unit group is greater than k, i.e., $|u_i|_p > k$, it will not be in any k-point G-Skyline group, and we can remove p_i directly from the DSG without having to consider it. If $|u_i|_p = k$, we can output u_i as one of the G-Skyline groups, and p_i will not be considered either as it will not contribute to any other G-Skyline groups.

EXAMPLE 4. *If we set $k = 4$ (we will use $k = 4$ in all the remaining examples of the paper), the node p_2, p_4, p_7 in Figure 3 can be removed directly because $|u_2|_p = 5, |u_4|_p = 7, |u_7|_p = 5$. Unit group $u_5 = \{p_6, p_{11}, p_8, p_5\}$ can be output as a G-Skyline group. As a result, p_2, p_5, p_4, p_7 will not be considered in our algorithms.*

5.1 The Point-Wise Algorithm

The problem of finding G-Skyline groups can be tackled by the classic set enumeration tree search framework. The idea is to expand possible groups over an ordered list of points as illustrated in Figure 4. Each node in the set enumeration tree is a *candidate* group. The first level contains the root node which is the empty set, while the ith level contains all i-point groups. Naively, we can enumerate all $\binom{\mathbb{S}_k}{k}$ candidates and employ Theorem 2 to check each candidate. However, this baseline method is too time-consuming which will be verified in our experiments.

The main idea of our algorithm is to dynamically generate the set enumeration tree of candidate groups one level at a time while pruning the non-G-Skyline candidates as much as possible without having to check them. For each node, we store a tail set that consists of the points with point index larger than the points in current node. Each node can be expanded to create a set of new nodes at the next level, each by adding a new point from its tail set. The root node contains an empty set with a tail set composed of all remaining points from the first k skyline layers after the preprocessing. We present our tree expansion and pruning strategies in detail below.

Subtree Pruning. We observe a property of superset monotonicity which allows us to do subtree pruning when a node is not a G-Skyline group.

THEOREM 3. ***(Superset Monotonicity).*** *If a group G_i with i points is not a G-Skyline group, by adding a new point from its tail set, the new group G_{i+1} with $i + 1$ points is not a G-Skyline group either.*

PROOF. If G_i is not a G-Skyline group, there is a group G'_i with i points that dominates G_i. For any superset of G_i with a new point p_j added from G_i's tail set, we denote it by $G_i \cup p_j$. Since the points are ordered by skyline layers in our DSG, and any point in G_i's tail set has a larger point index than the points in G_i, hence p_j will not dominate any points in G_i, and will not be in G'_i. It is then easy to see that the group $G'_i \cup p_j$ dominates $G_i \cup p_j$, i.e., $G'_i \cup p_j \prec G_i \cup p_j$. □

This theorem implies that if a candidate group G is not a G-Skyline group, we do not need to expand it further or check its subtree.

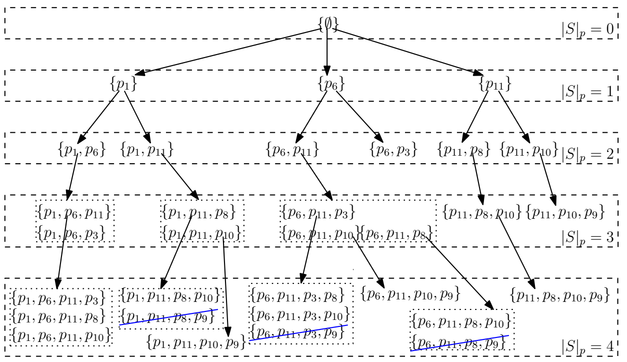
Tail Set Pruning. Each node in our tree can be expanded to a set of new nodes by adding a point from its tail set. However, not all tail points need to be considered. Recall Lemma 1, a point in a G-Skyline group cannot be dominated by a point outside the group. In other words, if a point p is to be added to a group G to form a new group that is a G-Skyline group, its parents must be in the group already. This means that p must be either a skyline point (with no parent), or a child of some points in G. Note this is a necessary condition but not sufficient. Once the candidate group is built, we still need to check whether p's parents are all in G (Theorem 2). However, this necessary condition allows us to quickly prune those points from the tail set of G that are not skyline points or not a child of points in G. In addition, given a candidate group G, if all its points are in the first i skyline layers, p must come from the first $i+1$ skyline layers. Otherwise, we can find a point outside G that lies on the $(i+1)^{th}$ layer to dominate p. Hence we can also prune the points beyond the $(i+1)^{th}$ layer. Once a point is pruned from the tail set, it will not be used to expand the current node. This is because based on the superset monotonicity, any node in the subtree of the new node will not be a G-Skyline group either.

Algorithm 2: The point-wise algorithm for computing G-Skyline groups.

```
   input  : a DSG and group size $k$.
   output: G-Skyline(k) groups.
1  initialize the G-Skyline(0) group at root node as an empty set
   and its tail set as all points from DSG after preprocessing;
2  for $i$=1 to $k$ do
3      for each G-Skyline(i-1) group $G$ do
4          for each point $p_l$ in $G$ do
5              add $p_l$'s children to Children Set $CS$;
6          for each point $p_j$ in Tail Set(TS) of $G$ do
7              if $p_j$ is not in $CS$ && $p_j$ is not a skyline point
               then
8                  delete $p_j$;
9              if $p_j.layer - max_l\{p_l.layer\} \geq 2$ then
10                 delete $p_j$;
11         for each remaining point $p$ in tail set of $G$ do
12             add $p$ to $G$ to form a new candidate G-Skyline(i)
               group;
13             if the new candidate group is not a G-Skyline
               group then
14                 delete;
```

Algorithm. Given the above two pruning strategies, we show our complete point-wise algorithm in Algorithm 2. Line 1 initializes the root node and its tail set. Tail set pruning is implemented in Line 4 to Line 10. For each node G_j at level i, it is expanded with a new point p from its tail set to form a new candidate group only if p is a child of some points in the current node or a skyline point and p is in the first $i+1$ layers. The for loop in Line 4 and Line 6 can be finished in linear time $O(|CS|)$ and $O(|CS|+|TS|)$, respectively, by employing the idea of merge sort, where $|CS|$ and $|TS|$ are the size of children set and tail set. The candidate group is verified in Line 13. If it is not a G-Skyline group, it will be pruned from the tree (subtree pruning).

Example 5. *We show a running example of Algorithm 2 in Figure 4 based on Figure 3. The root node at level* $|S|_p = 0$ *has an initial tail set* $\{p_1, p_6, p_{11}, p_3, p_8, p_{10}, p_9\}$*. Points* p_3, p_8, p_{10}, p_9 *can be pruned immediately because they are not skyline points, i.e., their parents are not in the root node. The remaining points* p_1, p_6, p_{11} *are used to create the new nodes at level* $|S|_p = 1$*. Similarly, for node* $\{p_1, p_{11}\}$ *at level* $|S|_p = 2$*, its tail set is* $\{p_3, p_8, p_{10}, p_9\}$*. Point* p_3 *can be pruned because* p_3 *is not a child of either* p_1 *or* p_{11}*. Point* p_9 *also can be pruned as* $p_9.layer - max\{p_1.layer, p_{11}.layer\} = 2$*. Hence, the remaining points* p_8, p_{10} *are used to create the new candidate groups at level* $|S|_p = 3$*, namely* $\{p_1, p_{11}, p_8\}$ *and* $\{p_1, p_{11}, p_{10}\}$*. As a result, level* $|S|_p = 4$ *shows all candidate 4-point groups that need to be checked. After checking, the ones with blue slash are not G-Skyline groups while the remaining ones are G-Skyline groups. Based on our pruning strategies, only* 31 *candidate groups are generated and checked while the baseline approach needs to enumerate and check* $\binom{8}{4} = 70$ *candidate groups.*


$\{\emptyset\}$
$|S|_p = 0$
$\{p_1\}$
$\{p_6\}$
$\{p_{11}\}$
$|S|_p = 1$
$\{p_1, p_6\}$
$\{p_1, p_{11}\}$
$\{p_6, p_{11}\}$
$\{p_6, p_3\}$
$\{p_{11}, p_8\}$
$\{p_{11}, p_{10}\}$
$|S|_p = 2$
$\{p_1, p_6, p_{11}\}$
$\{p_1, p_6, p_3\}$
$\{p_1, p_{11}, p_8\}$
$\{p_1, p_{11}, p_{10}\}$
$\{p_6, p_{11}, p_3\}$
$\{p_6, p_{11}, p_{10}\}$
$\{p_6, p_{11}, p_8\}$
$\{p_{11}, p_8, p_{10}\}$
$\{p_{11}, p_{10}, p_9\}$
$|S|_p = 3$
$\{p_1, p_6, p_{11}, p_3\}$
$\{p_1, p_6, p_{11}, p_8\}$
$\{p_1, p_6, p_{11}, p_{10}\}$
$\{p_1, p_{11}, p_8, p_{10}\}$
~~$\{p_1, p_{11}, p_8, p_9\}$~~
$\{p_1, p_{11}, p_{10}, p_9\}$
$\{p_6, p_{11}, p_3, p_8\}$
$\{p_6, p_{11}, p_3, p_{10}\}$
~~$\{p_6, p_{11}, p_3, p_9\}$~~
$\{p_6, p_{11}, p_{10}, p_9\}$
$\{p_6, p_{11}, p_8, p_{10}\}$
~~$\{p_6, p_{11}, p_8, p_9\}$~~
$\{p_{11}, p_8, p_{10}, p_9\}$
$|S|_p = 4$
Figure 4: The point-wise algorithm for finding G-Skyline groups when $k = 4$.

5.2 The Unit Group-Wise Algorithm

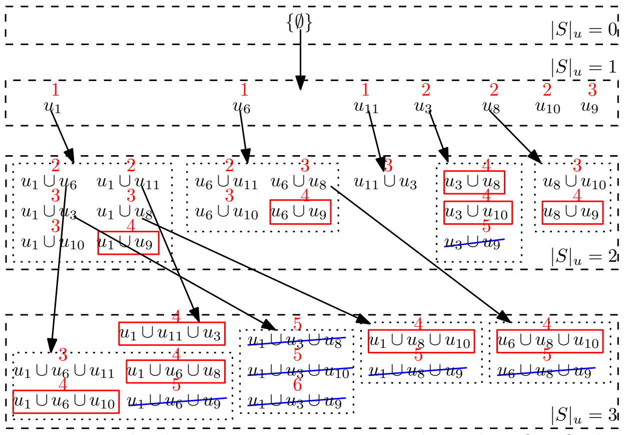
The point-wise algorithm expands candidate groups one point at a time. We already showed (in Lemma 1) that a point in a G-Skyline group cannot be dominated by a point outside the group. In other words, for a point in a G-Skyline group, its unit group must be in the group. This motivates our unit group-wise algorithm which expands candidate groups by unit groups, adding one unit group at a time. Similar to the point-wise algorithm, we can represent the entire search space of candidate groups as a set enumeration tree, where each node is a set of unit groups. We can dynamically generate the tree while pruning as much non-G-Skyline groups as possible. We can use similar pruning strategies as in the point-wise algorithm.

Superset Pruning. Given a candidate group G with at least k points, the candidate groups in G's subtree will have more than k points. Hence, we can prune G's subtree directly.

Example 6. *Figure 5 shows the dynamically generated set enumeration tree for the unit group-wise algorithm. At level* $|S|_u = 2$, $|u_3 \cup u_8|_p = 4$*, hence we do not need to check the candidate groups in its subtree, e.g.,* $u_3 \cup u_8 \cup u_{10}$*. This is because* $|u_3 \cup u_8 \cup u_{10}|_p = |\{p_6, p_{11}, p_3, p_8, p_{10}\}|_p = 5$.

Tail Set Pruning. For a candidate group G at Level $|S|_u = i$, we do not need to add the children of the unit groups in G to form a new candidate group at Level $|S|_u = i + 1$. Therefore, the children of the unit groups in G can be pruned from the tail set.

Example 7. *We show a running example for the unit group-wise algorithm with the two pruning strategies in Figure 5. The number on each candidate group represents the number of points in this candidate group. All candidate groups with one unit group are checked at Level $|S|_u = 1$. For candidate group u_6 at Level $|S|_u = 1$, its tail set is $\{u_{11}, u_3, u_8, u_{10}, u_9\}$. However, u_3 can be pruned by Tail Set Pruning because u_3 is the child of u_6. From Level $|S|_u = 2$ to Level $|S|_u = 3$, candidate group $u_3 \cup u_8$'s subtree does not need to be checked since $|u_3 \cup u_8|_p = 4$, thanks to Superset Pruning. Based on the two pruning strategies, only 35 candidate groups need to be checked as shown. The resulting G-Skyline(4) groups are shown in red (solid) boxes and we can see that they come from different levels while the point-wise algorithm outputs all G-Skyline(4) groups at Level $|S|_p = 4$.*



{∅}
|S|u = 0
|S|u = 1
u1 u6 u11 u3 u8 u10 u9
u1∪u6 u1∪u11 u1∪u3 u1∪u8 u1∪u10 u1∪u9
u6∪u11 u6∪u8 u6∪u10 u6∪u9
u11∪u3
u3∪u8 u3∪u10 u3∪u9
u8∪u10 u8∪u9
|S|u = 2
u1∪u11∪u3 u1∪u6∪u11 u1∪u6∪u8 u1∪u6∪u10 u1∪u6∪u9
u1∪u3∪u8 u1∪u3∪u10 u1∪u3∪u9
u1∪u8∪u10 u1∪u8∪u9
u6∪u8∪u10 u6∪u8∪u9
|S|u = 3

Figure 5: The basic unit group-wise algorithm for finding G-Skyline groups when $k = 4$.

In addition to the above strategies, we show a few additional refinements that further improve the algorithm.

Unit Group Reordering. We observe that Superset Pruning is important to reduce the candidate groups. This motivates us to reorder the unit groups in order to increase the effectiveness of Superset Pruning. Recall that Superset Pruning can be applied when a candidate group G has $|G|_p \geq k$. Therefore it would be beneficial if we build large candidate groups first which allows us to prune more non-G-Skyline candidate groups at an early stage. A good heuristic for accomplishing this is to reorder the unit groups for each point in the reverse order of their point index. This is because a point with a larger index, i.e., at a higher skyline layer, tends to have more parents, and hence a larger unit group size. By ordering the unit groups this way, they are likely in the (although not monotonically) decreasing order in their unit group size. We simply need to modify the Tail Set Pruning strategy such that for each node G, the parents (instead of the children) of the unit groups of G are pruned from its tail set.

Subset Pruning. Given a candidate group G_i, if $|G_i|_p \leq k$, then any candidate groups that are G_i's subset have at most k points. This motivates us to employ Subset Pruning. For

Algorithm 3: The unit group-wise algorithm for computing G-Skyline groups.

```
input  : a DSG and group size $k$.
output: G-Skyline(k) groups.
1  build 1-unit group as candidate groups following reverse order of point index;
2  for each candidate group $G$ in 1-unit groups do
3      if $|G^{last}|_p = k$ then
4          output $G^{last}$;
5          break;
6      else if $|G^{last}|_p < k$ then
7          break;
8      i=2;
9      while the set of candidate group $G'$, such that $|G'|_u = i - 1$, is not empty do
10         for each $G'$ do
11             for each unit group $u_i$ in $G'$ do
12                 add $u_i$'s parents to Parents Set $PS$;
13             for each unit group $u_j$ in tail set of $G'$ do
14                 delete $u_j$ if $u_j$ is in $PS$;
15             for each remaining unit group $u$ in tail set of $G'$ do
16                 add $u$ to $G'$ to form a new candidate group $G''$ that $|G''|_u = i$;
17         delete $G'$;
18         output $G''$ if $|G''|_p = k$;
19         delete $G''$ if $|G''|_p \geq$k;
20         i++;
```

each candidate group G_i with $|G_i|_p < k$ at Level $|S|_u = 1$, we can check a new candidate group by adding the entire tail set of G_i to G_i, i.e., the last or deepest leaf candidate group G_i^{last} in G_i's subtree. If $|G_i^{last}|_p \leq k$, the entire subtree of G_i can be pruned directly and G_i^{last} can be output if $|G_i^{last}|_p = k$. Furthermore, we do not need to check those candidate groups in the subtree of G_i's right siblings G_{sib} because the last candidate group in G_{sib}'s subtree, G_{sib}^{last}, is a subset of G_i^{last}, hence, $|G_{sib}^{last}|_p < k$ too. While both depth-first and breadth-first expansion of the tree will work equally well for Superset Pruning, Subset Pruning benefits from a depth-first expansion such that more siblings can be pruned.

Algorithm. Given the Superset Pruning, Unit Group Reordering, Subset Pruning, and Tail Set Pruning strategies, the complete unit group-wise algorithm is shown in Algorithm 3. Line 1 applies Unit Group Reordering. Subset Pruning is applied to the 1-unit groups in Lines 3 and 6. We also note that while Subset Pruning can be employed at every node, it also adds additional cost for checking the leaf node. So we only apply them at Level $|S|_u = 1$ to gain the most pruning benefit, as k is far less than n in the usual case. Tail Set Pruning is applied in Line 11 to Line 14 to prune those candidate groups that do not need to be checked. Line 19 applies Superset Pruning to prune those candidate groups with more than k points.

Example 8. *We show a running example of Algorithm 3 in Figure 6. The 1-unit groups are built in reverse order of point index at Level $|S|_u = 1$. For candidate group u_9, we first check the last candidate group (linked by red (thin) arrow) which has $|u_9 \cup u_{10} \cup u_8 \cup u_3 \cup u_{11} \cup u_6 \cup u_1|_p = |u_9 \cup u_8 \cup u_3 \cup u_1|_p = 7 > 4$. So Subset Pruning cannot be applied, and we still need to check u_9's subtree. We build each branch with a depth-first strategy. At candidate group u_3, the* *last candidate group of its subtree has* $|u_3 \cup u_{11} \cup u_6 \cup u_1|_p = |u_3 \cup u_{11} \cup u_1|_p = 4$*, so it is a G-Skyline(4) group. Based on the Subset Pruning strategy, the algorithm is terminated because there are no more candidate groups that need to be checked. As a result, only* 27 *candidate groups need to be checked as shown in Figure 6.*

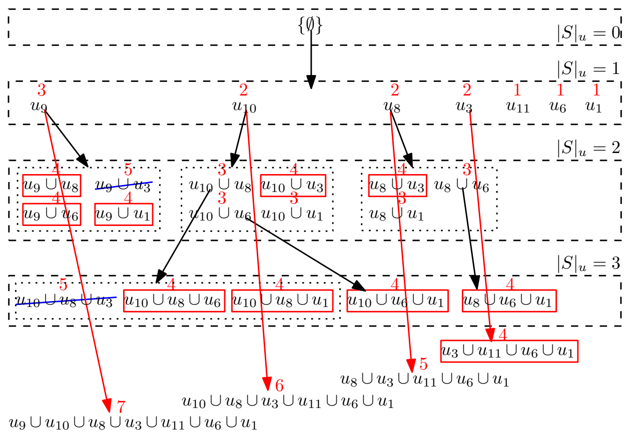

{∅}
$|S|_u = 0$
$|S|_u = 1$
$|S|_u = 2$
$|S|_u = 3$
u_9
u_{10}
u_8
u_3
u_{11}
u_6
u_1
$u_9 \cup u_8$
~~$u_9 \cup u_3$~~
$u_9 \cup u_6$
$u_9 \cup u_1$
$u_{10} \cup u_8$
$u_{10} \cup u_3$
$u_{10} \cup u_6$
$u_{10} \cup u_1$
$u_8 \cup u_3$
$u_8 \cup u_6$
$u_8 \cup u_1$
~~$u_{10} \cup u_8 \cup u_3$~~
$u_{10} \cup u_8 \cup u_6$
$u_{10} \cup u_8 \cup u_1$
$u_{10} \cup u_6 \cup u_1$
$u_8 \cup u_6 \cup u_1$
$u_3 \cup u_{11} \cup u_6 \cup u_1$
$u_8 \cup u_3 \cup u_{11} \cup u_6 \cup u_1$
$u_{10} \cup u_8 \cup u_3 \cup u_{11} \cup u_6 \cup u_1$
$u_9 \cup u_{10} \cup u_8 \cup u_3 \cup u_{11} \cup u_6 \cup u_1$

Figure 6: The enhanced unit group-wise algorithm for finding G-Skyline groups when $k = 4$.

6. EXPERIMENTS

In this section, we present experimental studies evaluating our approach.

6.1 Experiment Setup

We first present a small user study using the example hotel dataset (as shown in Figure 1) to verify the motivation of G-Skyline. We then evaluate the algorithm for computing the skyline layers, and then perform an extensive empirical study to examine the point-wise and unit group-wise algorithms using both synthetic and real datasets.

Since this is the first work for group-based skyline with the new definition of G-Skyline, our performance evaluation was conducted against the enumeration method as a baseline. We implemented the following algorithms in Java and ran experiments on a machine with Intel Core i7 running Ubuntu with 8GB memory.

- **PWise**: Point-wise algorithm presented in Subsection 5.1.
- **UWise**: Basic unit group-wise algorithm with Superset Pruning and Tail Set Pruning.
- **UWise+**: Refined unit group-wise algorithm with Unit Group Reordering and Subset Pruning.
- **BL**: We enumerate all $\binom{\mathbb{S}_k}{k}$ candidates, and use Theorem 2 to verify each candidate.

We used both synthetic datasets and a real NBA dataset in our experiments. To study the scalability of our methods, we generated independent (INDE), correlated (CORR), and anti-correlated (ANTI) datasets following the seminal work [5]. We also built a dataset that contains 2384 NBA players who are league leaders of playoffs. The data was extracted from http://stats.nba.com/leaders/alltime/?ls=iref:nba:gnav on 04/15/2015. Each player has five attributes that measure the player's performance. Those attributes are Points (PTS), Rebounds (REB), Assists (AST), Steals (STL), and Blocks (BLK).

6.2 Case Study

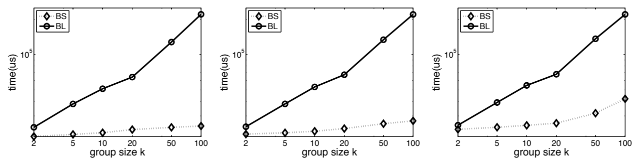
We performed a small user study using the hotel example dataset (Figure 1). We posted a questionnaire using the conference scenario to ask 38 students and staff members in our department and 30 workers from Amazon Mechanical Turk. We asked them to answer with groups of 2 hotels that they think are the best and provide the reasons of their selections when possible. We received 61 responses in total. Table 3 shows the number of answers for each hotel combinations. The results indeed showed that our group skyline definition covers all the returned groups, while taking any k-skyline points and other existing SUM based group skyline definitions [18, 14, 34] will miss a number of groups that are perceived relevant by the users, such as $\{p_6, p_3\}$ and $\{p_{11}, p_{10}\}$.

Table 3: Results of case study.

$\{p_{11}, p_8\}$	$\{p_6, p_{11}\}$	$\{p_{11}, p_{10}\}$	$\{p_6, p_3\}$	$\{p_1, p_{11}\}$	$\{p_1, p_6\}$
10	14	12	10	8	7

6.3 Computing Skyline Layers

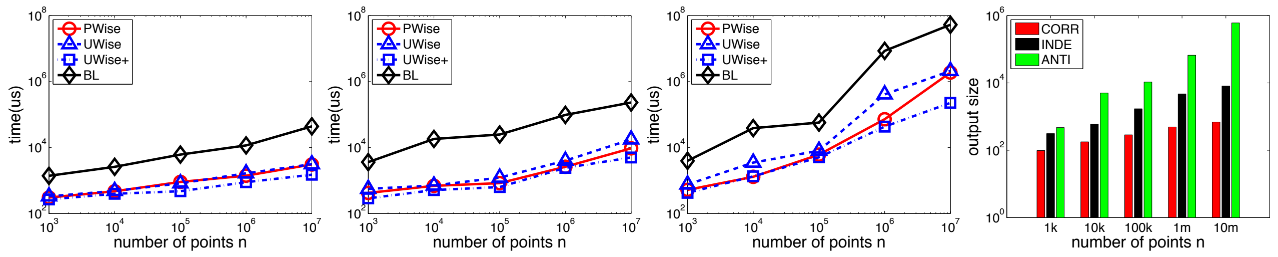
We first evaluate our algorithms for computing skyline layers. A baseline approach (BL) is to iteratively compute and then remove the skyline points for each layer. We compare our binary search algorithm (BS) that builds all skyline layers simultaneously for two dimensional space to the baseline approach.



(a) CORR (b) INDE (c) ANTI

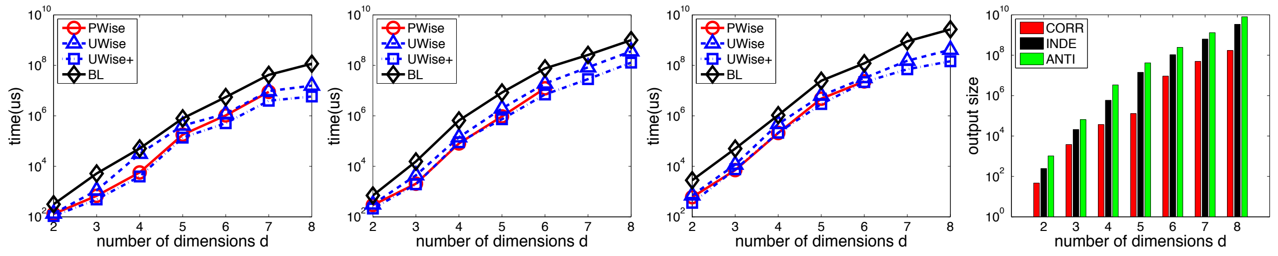
Figure 10: Computing skyline layers on synthetic datasets of varying k.

Figure 10 shows the runtime of our binary search algorithm and the Baseline algorithm for varying group size k on the three different datasets (n=10k) respectively. The runtime for the baseline algorithm is not significantly different for the three datasets because the number of skyline points (which differ in the three datasets) has minimal impact on the skyline algorithms in two-dimensional space. For each dataset, the time of the baseline algorithm almost linearly increases with the increase of group size k because the algorithm iteratively computes the skyline points for each layer. Different from the baseline algorithm, the running time of our binary search algorithm is affected by the different datasets and shows little growth from CORR to INDE, and from INDE to ANTI dataset. The reason is, in our algorithm, only the points in the first k skyline layers will trigger the binary search on the existing layers while the points not in the first k skyline layers are dropped directly. Because of the distribution or correlation patterns of the datasets, the average number of points in each skyline layer (a) for the datasets follows $CORR.a < INDE.a < ANTI.a$, which explains the runtime difference among the datasets. Finally, our binary search algorithm significantly outperforms the baseline algorithm on all datasets. We also implemented and evaluated the higher dimensional case. Even though both algorithms are sensitive to the data distribution in higher-dimensional



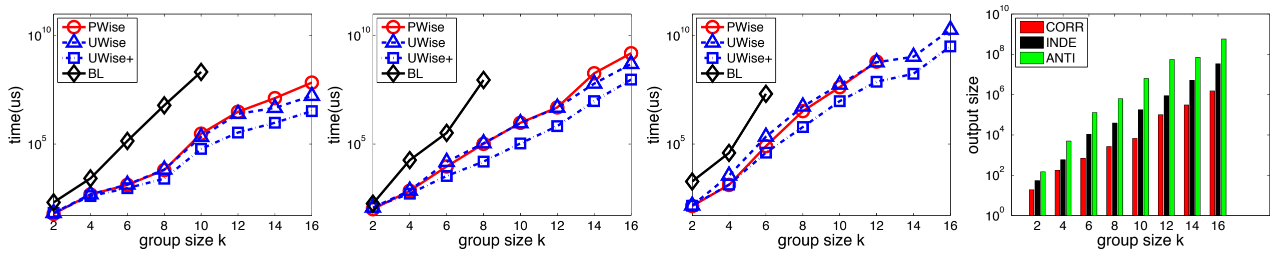
(a) time cost of CORR (b) time cost of INDE (c) time cost of ANTI (d) output size

Figure 7: Computing G-Skyline groups on synthetic datasets of varying n.



(a) time cost of CORR (b) time cost of INDE (c) time cost of ANTI (d) output size

Figure 8: Computing G-Skyline groups on synthetic datasets of varying d.



(a) time cost of CORR (b) time cost of INDE (c) time cost of ANTI (d) output size

Figure 9: Computing G-Skyline groups on synthetic datasets of varying k.

space, BS still significantly outperforms BL. We did not report them here due to limited space.

6.4 G-Skyline Groups in the Synthetic Data

In this subsection, we report the experimental results for computing G-Skyline groups based on synthetic data.

Figures 7(a)(b)(c) present the time cost of UWise, UWise+, PWise, and BL with varying number of points n for the three datasets ($d = 2, k = 4$). Figure 7(d) shows the output size with varying n on the three datasets. Because only the points in the first k skyline layers (total number is $\mathbb{S}_k$) are used to compute G-Skyline groups and $\mathbb{S}_k \ll n$ in general, we can see that the time cost and output size are not significantly impacted by n. Figures 7(a)(b)(c)(d) show that the time cost and output size grow approximately linearly with n.

Figures 8(a)(b)(c) show the time cost of UWise, UWise+, PWise, and BL with varying number of dimensions d on the three datasets ($n = 10000, k = 3$). Figure 8(d) shows the output size with varying d on the three datasets. The time cost and output size increase exponentially with respect to the increasing d. This is largely due to the increasing number of points in the first k skyline layers. We did not report the result of the PWise algorithm in some figures due to the high space cost of PWise since it needs to generate much more candidates than UWise.

Figures 9(a)(b)(c) show the time cost of UWise, UWise+, PWise, and BL with varying group size k on the three datasets ($n = 10000, d = 2$). Figure 9(d) shows the output size with varying k on the three datasets. We did not report the result of the BL algorithm in some figures due to the high cost when k is big. The time cost increases exponentially with respect to the increasing k. Furthermore, the group size k has a significant impact on the output size because there are $\binom{\mathbb{S}_k}{k} \approx \mathbb{S}_k{}^k$ candidate groups. Empirically, the result shows that the output size also grows exponentially with k as shown in Figure 9(d).

From the viewpoint of different datasets, the time cost and output size are in increasing order for CORR, INDE, and ANTI, due to the increasing number of points in the first k skyline layers. Comparing different algorithms, UWise, UWise+, and PWise significantly outperform BL, which validates the benefit of our pruning strategies. PWise is better than UWise when k is small but worse when k is big. Furthermore, PWise is highly space-consuming. Both UWise and UWise+ outperform PWise when k is big, which shows the benefit of the unit group notion. UWise+ outperforms

UWise, thanks to the Unit Group Reordering and Subset Pruning strategies.

Discussion. We note that the large output size is indeed a challenging problem for our G-Skyline definition as well as the other group skyline definitions and even the original skyline definition, especially for the ANTI datasets. We provide some discussions as follows. First, the essence of skyline is arguably not to fully help users to choose points given the assumption that the users' attribute weights or preferences are unknown in advance. Rather skyline can be particularly useful to prune those points that are certain to be inferior or dominated by others given any attribute weights. Hence, in a way, we can consider skyline as a preprocessing step for multi-criteria decision making. In this regard, the (relative) output ratio in our results is significantly small compared to the number of all possible groups. Second, if the output size is too large to be consumed by users, additional steps can be performed to choose meaningful representative points. Several existing works [6, 20, 31] investigated this challenging problem. Finally, we also show a weaker group dominance relationship definition, PG-Skyline, which alleviates this issue in Section 7.

6.5 G-Skyline Groups in the NBA Data

In this subsection, we report the experimental results on the NBA real data.

Figure 11(a) shows the time cost with varying n when $d = 5, k = 5$. For each value of $n = 500, 1000, 1500, 2000$, we took the average result based on 100 experiment runs and for each experiment, we randomly chose n out of 2384 players. UWise, UWise+, and PWise again significantly outperform BL due to the efficient pruning strategies and UWise+ performs the best. However, varying n does not have a significant impact on the runtime and output size, as shown in Figure 11(b). The reason is that only the number of points in the first k layers is used to compute the G-Skyline groups and $\mathbb{S}_k \ll n$ in general.

Figures 12(a) and (b) show the time cost and output size for different d when $n = 2384, k = 5$. We took the average result of all possible dimension combinations. We see the number of dimensions d has a large impact on both the runtime and output size which increase with increasing d.

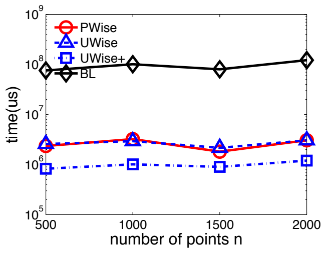
Figure 13 shows the time cost and output size for different k when $n = 2384, d = 5$. k also has a large impact since the number of points in the first k skyline layers increases significantly as k increases while our approaches are less impacted than the Baseline.

We also report a sample of the final G-Skyline groups in Table 3. There are $\binom{2384}{5} \approx 6.4 \times 10^{14}$ candidate groups, but our algorithm only returns 4865073 G-Skyline groups, i.e., 1 out of 1.3×10^8. We can see the sample groups are formed by elite players with different strengths. For example, G3 is excellent in PTS, REB, AST, STL, and BLK while G1 excels in PTS, and G4 is a good balanced group.

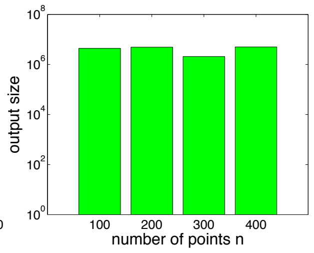
7. EXTENSIONS

In this section, we discuss two interesting extensions of our proposed work: 1) an AG-Skyline definition based on a more restrictive all-permutation group dominance than G-Skyline, and 2) a PG-Skyline definition based on a less restrictive partial group dominance.

7.1 AG-Skyline

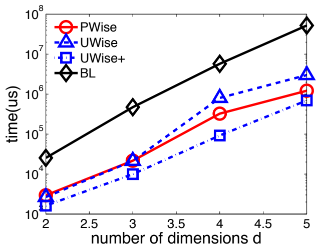


(a) impact of n.

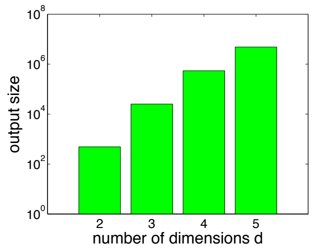


(b) impact of n.

Figure 11: G-Skyline on NBA dataset of varying n.

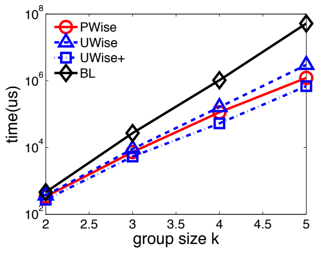


(a) impact of d.

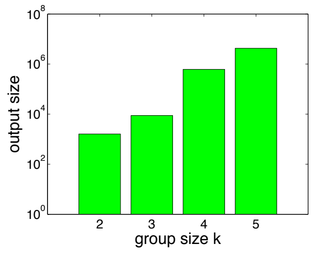


(b) impact of d.

Figure 12: G-Skyline on NBA dataset of varying d.



(a) impact of k.



(b) impact of k.

Figure 13: G-Skyline on NBA dataset of varying k.

Our G-Skyline definition is based on the group dominance defined between a pair of permutations of the points in each group. We formulate an alternative definition AG-Skyline as follows.

Definition 7. (**AG-Skyline**). Given a dataset P of n points in a d-dimensional space. Let $G = \{p_1, p_2, ..., p_k\}$ and $G' = \{p'_1, p'_2, ..., p'_k\}$ be two different groups with k points, we say group G ag-dominates group G' if for all (i, j) pairs, $p_i \preceq p'_j$, and for at least one pair (i, j), $p_i \prec p'_j$. The AG-Skyline are those groups that are not ag-dominated by any other groups with same size.

While g-dominance only requires point-wise domination between two groups for *one* permutation of the points in each group (*one-to-one* point domination), ag-dominance requires each point in one group dominates all points in the other group (*one-to-all* point domination). In other words, ag-dominance requires point-wise domination between two groups for *all* permutations of the points. Because the ag-dominance relationship of AG-Skyline is more strict than G-Skyline, less candidate groups can be dominated by other groups, that is, the output of AG-Skyline is a superset of the output of G-Skyline with same group size on the same input dataset.

Table 4: Sample of G-Skyline groups on the NBA dataset.

G1	Michael Jordan	Anthony Davis	Kyrie Irving	Allen Iverson	Jerry West	high PTS
G2	Magic Johnson	John Stockton	Isaiah Thomas	Chris Paul	Rajon Rondo	high AST
G3	Michael Jordan	Bill Russell	Magic Johnson	Lance Blanks	Hakeem Olajuwon	high PTS,REB,AST,STL,BLK
G4	Maurice Cheeks	Rich Barry	Slick Watts	Baron Davis	Brad Daugherty	very balanced
G5	Julius Erving	Elvin Hayes	Michael Jordan	Khris Middleton	Alvin Robertson	high STL,BLK

7.2 PG-Skyline

A potential limitation of our proposed definition is the large number of output groups. As the number of dimensions and group size increase, the chance for one group to g-dominate another group is low. As such, the number of G-Skyline groups becomes significantly large, especially for anti-correlated datasets. A potential solution to circumvent this issue is to define an alternative, more relaxed group-dominance relationship. We define PG-Skyline which is similar to the notion in [6] for individual skyline points. The key idea is to relax the dominance requirement from point-wise dominance for *all* points in each group to (*partial*) p points where $p \leq k$.

Definition 8. (**PG-Skyline**). Given a dataset P of n points in a d-dimensional space. Let $G = \{p_1, p_2, ..., p_k\}$ and $G' = \{p'_1, p'_2, ..., p'_k\}$ be two different groups with k points of P, we say group G pg-dominates group G' if for p points ($p \leq k$) in G and G', we can find two permutations of the p points, $G = \{p_{u_1}, p_{u_2}, ..., p_{u_p}\}$ and $G' = \{p'_{v_1}, p'_{v_2}, ..., p'_{v_p}\}$, such that $p_{u_i} \preceq p'_{v_i}$, for all i ($1 \leq i \leq p$) and $p_{u_i} \prec p'_{v_i}$ for at least one i. The PG-Skyline are those groups that are not pg-dominated by any other group with same size.

Because the dominance relationship of PG-Skyline is less strict than G-Skyline, more candidate groups can be dominated by other groups, that is, the output of PG-Skyline is a subset of the output of G-Skyline with the same group size on the same input dataset.

8. CONCLUSIONS

In this paper, we proposed the problem of G-Skyline groups for finding Pareto optimal groups, instead of finding Pareto optimal points in the classic definition of skylines. Our definition is based on a dominance relationship between groups with same number of points. This is the first work to extend the original skyline definition to group level which captures the quintessence of original skyline definition. To compute the G-Skyline groups efficiently, we presented a novel structure based on skyline layers that not only partitions the points efficiently but also captures the dominance relationship between the points. We then presented point-wise and unit group-wise algorithms to compute the G-Skyline groups efficiently. A comprehensive experimental study is reported demonstrating the benefit of our algorithms. We also discussed two alternative dominance definitions, AG-Skyline and PG-Skyline, which are the superset and subset of G-Skyline, respectively.

Acknowledgement

The authors would like to thank the anonymous reviewers for their helpful comments. This research is supported in part by the Air Force Office of Scientific Research (AFOSR) DDDAS Program under award number FA9550-12-1-0240, the National Natural Science Foundation of China (Grant No. 11271351), Guangdong Science and Technology Program Fund 2013B091300019, and an NSERC Discovery grant. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

9. REFERENCES

[1] F. N. Afrati, P. Koutris, D. Suciu, and J. D. Ullman. Parallel skyline queries. In *ICDT*, pages 274–284, 2012.
[2] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In *SODA*, pages 179–187, 1990.
[3] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, 1978.
[4] H. Blunck and J. Vahrenhold. In-place algorithms for computing (layers of) maxima. In *Algorithm Theory - SWAT 2006*, pages 363–374, 2006.
[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
[6] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, pages 503–514, 2006.
[7] L. Chen and X. Lian. Dynamic skyline queries in metric spaces. In *EDBT*, pages 333–343, 2008.
[8] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.
[9] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Summarizing two-dimensional data with skyline-based statistical descriptors. In *SSDBM*, pages 42–60, 2008.
[10] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, pages 291–302, 2007.
[11] X. Ding, X. Lian, L. Chen, and H. Jin. Continuous monitoring of skylines over uncertain data streams. *Inf. Sci.*, 184(1):196–214, 2012.
[12] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007.
[13] Z. Huang, H. Lu, B. C. Ooi, and A. K. H. Tung. Continuous skyline queries for moving objects. *IEEE Trans. Knowl. Data Eng.*, 18(12):1645–1658, 2006.
[14] H. Im and S. Park. Group skyline computation. *Inf. Sci.*, 188:151–169, 2012.
[15] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Symposium on Computational Geometry*, pages 89–96, 1985.
[16] H. Köhler, J. Yang, and X. Zhou. Efficient parallel skyline processing using hyperplane projections. In *SIGMOD*, pages 85–96, 2011.
[17] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.
[18] C. Li, N. Zhang, N. Hassan, S. Rajasekaran, and G. Das. On skyline groups. In *CIKM*, pages 2119–2123, 2012.
[19] X. Lian and L. Chen. Reverse skyline search in uncertain databases. *ACM Trans. Database Syst.*, 35(1), 2010.
[20] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.
[21] J. Liu, L. Xiong, and X. Xu. Faster output-sensitive skyline computation algorithm. *Inf. Process. Lett.*, 114(12):710–713, 2014.
[22] J. Liu, H. Zhang, L. Xiong, H. Li, and J. Luo. Finding probabilistic k-skyline sets on uncertain data. In *CIKM 2015*.
[23] H. Lu, C. S. Jensen, and Z. Zhang. Flexible and efficient resolution of skyline query size constraints. *IEEE Trans. Knowl. Data Eng.*, 23(7):991–1005, 2011.
[24] M. Magnani and I. Assent. From stars to galaxies: skyline queries on aggregate data. In *EDBT*, pages 477–488, 2013.
[25] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.
[26] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.
[27] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB*, pages 253–264, 2005.
[28] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu, and Q. Zhang. Towards multidimensional subspace skyline analysis. *ACM Trans. Database Syst.*, 31(4):1335–1381, 2006.
[29] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, pages 751–762, 2006.
[30] C. Sheng and Y. Tao. On finding skylines in external memory. In *PODS*, pages 107–116, 2011.
[31] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009.
[32] Y. Tao, X. Xiao, and J. Pei. Efficient skyline and top-k retrieval in subspaces. *IEEE Trans. Knowl. Data Eng.*, 19(8):1072–1088, 2007.
[33] G. Wang, J. Xin, L. Chen, and Y. Liu. Energy-efficient reverse skyline query processing over wireless sensor networks. *IEEE Trans. Knowl. Data Eng.*, 24(7):1259–1275, 2012.
[34] N. Zhang, C. Li, N. Hassan, S. Rajasekaran, and G. Das. On skyline groups. *IEEE Trans. Knowl. Data Eng.*, 26(4):942–956, 2014.
[35] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. X. Yu. Probabilistic skyline operator over sliding windows. In *ICDE*, pages 1060–1071, 2009.