Fast Algorithms for Pareto Optimal Group-based Skyline

Wenhui Yu
Tsinghua University
Beijing, China
yuwh16@mails.tsinghua.edu.cn

Zheng Qin*
Tsinghua University
Beijing, China
qingzh@mail.tsinghua.edu.cn

Jinfei Liu
Emory University and Georgia Institute of Technology
jinfei.liu@emory.edu

Li Xiong
Emory University
Atlanta, USA
lxiong@emory.edu

Xu Chen
Tsinghua University
Beijing, China
xu-ch14@mails.tsinghua.edu.cn

Huidi Zhang
Tsinghua University
Beijing, China
zhd16@mails.tsinghua.edu.cn

ABSTRACT

Skyline, aiming at finding a Pareto optimal subset of points in a multi-dimensional dataset, has gained great interest due to its extensive use for multi-criteria analysis and decision making. Skyline consists of all points that are not dominated by, or not worse than other points. It is a candidate set of optimal solution, which depends on a specific evaluation criterion for optimum. However, conventional skyline queries, which return individual points, are inadequate in group querying case since optimal combinations are required. To address this gap, we study the skyline computation in group case and propose fast methods to find the group-based skyline (G-skyline), which contains Pareto optimal groups. For computing the front k skyline layers, we lay out an efficient approach that does the search concurrently on each dimension and investigates each point in subspace. After that, we present a novel structure to construct the G-skyline with a queue of combinations of the first-layer points. Experimental results show that our algorithms are several orders of magnitude faster than the previous work.

CCS CONCEPTS

•**Information systems →Query optimization;**

KEYWORDS

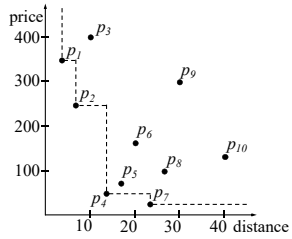
Group skyline, multiple skyline layers, concurrent search, subspace skyline, combination queue.

1 INTRODUCTION

Skyline, known as *Maxima* in computational geometry or *Pareto* in business management field, is of great significance for many applications. Skyline returns a subset of points that are Pareto optimal [5], indicating that these points cannot be dominated by any other points in the dataset. Though the exact optimal point depends on the specific criterion for optimum, skyline can provide a candidate set so that we can prune the non-candidates from the dataset when the optimal solution is queried.

hotel	distance	price
p_1	3	350
p_2	6	250
p_3	10	400
p_4	13	50
p_5	16	75
p_6	20	160
p_7	23	25
p_8	26	100
p_9	30	300
p_{10}	40	130

(a)



(b)

Figure 1: A skyline example of hotels.

Given a dataset S with n points, each point p has d numeric attributes and can be represented as a d-dimensional vector $(p[1], p[2], \cdots, p[d]) \in \mathbb{R}^d$, where $p[i]$ is the i-th attribute of the dataset. Given two points $p = (p[1], p[2], \cdots, p[d])$ and $p' = (p'[1], p'[2], \cdots, p'[d])$, point p dominates point p' if $p[i] < p'[i]$ for at least one attribute and $p[i] \leq p'[i]$ for the others ($1 \leq i \leq d$). Skyline of dataset S is defined as a subset consisting of all points that are not dominated by any other points in S. Evidently, all the points in skyline are Pareto optimal solutions.

Figure 1(a) illustrates ten hotels with two attributes (price and the distance to a given destination) in the table. Travelers desire to choose a hotel with both low price and short distance. Figure 1(b) represents ten points in two-dimensional space and each point represents a hotel in Figure 1(a) correspondingly. When choosing hotels, travelers will not benefit by choosing p_5, p_6, p_8, or p_{10} because they are dominated by p_4 and are in worse situation in both distance and price attributes. Hotels p_1, p_2, p_4, and p_7 might be chosen since they are not dominated by any other hotels. The final choice depends on the travelers' criteria, or the weights of attributes. For example, if the travelers are wealthy, they may attach more importance to the distance and choose p_1 or p_2. If they want to be cost-effective, they may prefer p_4 or p_7 because of the lower price. We can see that the skyline is a candidate set of the optimal solution, and the final decision depends on travelers' specific criteria.

Motivation. Extensively studied in the database community, skyline has been extended with many variants to provide a candidate set in different situations. However, traditional skyline, which is a candidate set of all top-1 solutions, is inadequate when optimal

*Corresponding author

School of Software, Tsinghua National Laboratory for Information Science and Technology.

CIKM'17, November 6–10, 2017, Singapore.

 ISBN 978-1-4503-4918-5/17/11…$15.00
DOI: https://doi.org/10.1145/3132847.3132950

groups or top-k solutions are searched rather than individual points. This is a very important problem with many real world applications that is surprisingly neglected. To address this gap, group skyline (with size k), which is a candidate set of all top-k solutions, is proposed. However, the few existing works either do not return complete candidate set [11, 29] or are too computationally expensive [17], so an efficient and complete method is greatly desired.

Recall our hotel example, suppose that a travel agency wants to collaborate with best three hotels. If they are mainly targeting low-end travelers, they may choose hotels $\{p_4, p_7, p_5\}$ or $\{p_4, p_5, p_6\}$. If for high-end travelers, they may choose $\{p_1, p_2, p_3\}$. And if they want to provide a more extensive service, they may choose $\{p_1, p_2, p_4\}$, $\{p_2, p_4, p_7\}$, or $\{p_1, p_4, p_7\}$. In some groups mentioned above, all points contained are skyline points, e.g., $\{p_1, p_2, p_4\}$, $\{p_2, p_4, p_7\}$, $\{p_1, p_4, p_7\}$, but in the others, non-skyline points may be included, e.g., $\{p_4, p_7, p_5\}$, $\{p_4, p_5, p_6\}$, $\{p_1, p_2, p_3\}$. It is apparent that the non-skyline points can also be chosen in this problem. However, the components of these groups are not arbitrary, and they need to satisfy certain requirements. For instance, groups $\{p_4, p_7, p_8\}$ or $\{p_4, p_7, p_6\}$ cannot be the best choice since p_5 dominates p_6 and p_8, so $\{p_4, p_7, p_5\}$ is better than them. We can see that it become more complicated when finding top-k solutions.

A group-based skyline (G-skyline) [17] is defined to address this problem. Consider a dataset S of n points in d-dimensional space. $G = \{p_1, p_2, \cdots, p_k\}$ and $G' = \{p'_1, p'_2, \cdots, p'_k\}$ are two different groups with k points. If we can find two permutations of the k points for G and G', $G = \{p_{u1}, p_{u2}, \cdots, p_{uk}\}$ and $G' = \{p'_{u1}, p'_{u2}, \cdots, p'_{uk}\}$, letting $p_{ui} \preceq p'_{ui}$ for all i ($1 \leq i \leq k$) and $p_{ui} \prec p'_{ui}$ for at least one i, we say G **g-dominates** G'. All groups containing k points that are not g-dominated by any other group with the same size compose G-skyline.

G-skyline is a complete candidate set since it contains all Pareto optimal groups, but a time-consuming algorithm returning an oversized candidate set is far from practical. Our work focuses on fast algorithms for G-skyline with more concise results.

Contributions. We briefly summarize our contributions as follows:

- We define Multiple Skyline Layers as the first k skyline layers, which is very important in both traditional skyline and group skyline problems. We propose a novel algorithm and structure to construct multiple skyline layers. Our framework searches concurrently in each dimension which reduces the redundant search and the search strategy utilizes the property of the skyline in subspace to improve efficiency. Our algorithm has a time complexity of $O\left(\mathbb{T}_k\left(n^{\frac{d-1}{d}}k + \mathbb{S}_k \log k\right)\right)$, where $\mathbb{S}_k$ is the number of the points in the first k layers and $\mathbb{T}_k$ is the subspace skyline size of the k-th layer, which is significantly more efficient than existing algorithms for skyline layers.
- Based on multiple skyline layers, we propose two fast algorithms to construct G-skyline groups. Our methods are based on the observation that skyline points contribute more to skyline groups compared to non-skyline points. We use a queue to enumerate all combinations of skyline points, and then find the groups that contain non-skyline points by the queue efficiently.
- We devise comprehensive experiments on synthetic and real datasets. The experimental results show that our algorithms are highly efficient, complete, and scalable.

Organization. The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 presents the definition of multiple skyline layers and the algorithmic technique to construct it. Two algorithms for finding G-skyline groups are presented in section 4. We evaluate the performance of each algorithm in Section 5. Section 6 concludes the paper.

2 RELATED WORK

In this section, we briefly explore previous works on skyline computation. After Kung's original work that proposed the in-memory algorithms to handle the skyline computation problem [14], skyline query has attracted extensive attention over the last decade due to its significance in computational geometry and database fields [2, 3, 12, 14, 18]. Börzsönyi [5] first studied how to process skyline queries in database systems and devised the skyline operator. Since then, the research includes improved algorithms [7, 9], progressive skyline computation [13, 23, 26], query optimization [1], and variants of skyline queries [1, 6, 8, 19, 22, 24, 25, 27, 28]. There are also works focusing on different types of data, for example, dynamic data [10] and uncertain data [20].

In recent years, there are increasing works focusing on group-based skyline [11, 15, 17, 29]. The definition of dominance between groups in these works varies greatly. [11, 15, 29] used a single aggregate point to represent a group and find the dominance relationship of groups using the aggregate points in a traditional way. An aggregate function is responsible for generating the aggregate point, whose attribute values are aggregated over the corresponding attribute value of all points in the group. Though many aggregate functions can be considered in calculating aggregate points, only several of them, such as SUM, MIN, MAX, have been discussed in previous works. [11] used SUM to aggregate the points and [29] utilized MAX and MIN. Intuitively, SUM captures the collective strength of a group, while MIN/MAX compares groups by their weakest/strongest member on each attribute. The skyline groups constructed by these methods are not complete because not all Pareto optimal groups can be captured. Instead of aggregating data, [17] proposed a different notion of group skyline called G-skyline, the dominance relationship between two groups is defined based on the pair-wise dominance between points in the two groups. Compared with previous works, G-skyline gives a more complete solution. In fact, group skyline proposed in [11] is a subset of G-skyline in [17]. However, completeness also means high complexity in computation, so this method is time-consuming and returns too many groups. To bridge this gap, we improve G-skyline method and give some fast algorithms.

3 MULTIPLE SKYLINE LAYERS

The skyline layers of a dataset consists of multiple layers of skyline, where the first layer is the skyline of the original dataset, the second layer is the skyline of the remaining points after the first layer is removed from the dataset, and so on. The first k skyline layers of a dataset is indispensable in nearly all group skyline algorithms, including G-skyline [17] and aggregation approaches [11, 29]. It

Table 1: The summary of notations.

Notation	Definition
$S \backslash layer_i$	*points in S but not in* $layer_i$
$p_i \preceq p_j$	p_i *dominates or equals to* p_j
$skyline(S)$	*the skyline of dataset S*
$layer(p_i)$	*the skyline layer of* p_i
$D(layer_i)$	*the dominance domain of* $layer_i$
p/S	*points/dataset in original space*
p'/S'	*projection of points/dataset in subspace*
$skyline_{s-s}(S')$	*the skyline of* S' *in subspace*
$D_{s-s}(layer'_i)$	*the dominance domain of* $layer'_i$ *in subspace*
$T_S(i)$	*tail set of element i in sequence S*

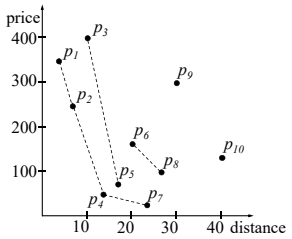


Figure 2: Multiple skyline layers ($n = 10, d = 2, k = 3$).

is proved that only the first k layers are necessary to find group skyline rather than the whole dataset [11, 17, 29]. However, very few papers study this issue, they usually focus on how to establish the group skyline. Skyline layers are also useful in traditional skyline problems [4, 16, 21]. It can be computed layer by layer using a traditional skyline method but it is ineffective. In fact, [14] shows that $O(n \log n)$ time is needed to construct one layer. So, constructing k layers costs $O(kn \log n)$ running time. [4, 17, 21] lay out several new methods that can find all layers in one enumeration with $O(n + \mathbb{S}_k \log k)$ time complexity for two- and $O(n\mathbb{S}_k)$ for higher-dimensional spaces. They are much better but there is still great room for improvement. In this section, we investigate a high-efficiency approach to construct the first k skyline layers, defined as multiple skyline layers, with $O\left(\mathbb{T}_k\left(n\frac{d-1}{d}k + \mathbb{S}_k \log k\right)\right)$ time complexity. Noticing $\mathbb{T}_k$ is 1 in the two-dimensional space, the time complexity becomes $O\left(\sqrt{n}k + \mathbb{S}_k \log k\right)$. A list of notations is summarized in Table 1.

Definition 1 (Skyline). *Given a dataset S of n points in d-dimensional space. Let p and p′ to be two different points in S, 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 consists of the points that are not dominated by any other points in S.*

Definition 2 (Multiple Skyline Layers (MSL)). *Given a dataset S with n points in d-dimensional space, MSL is a multi-layer structure. $Layer_1$ is the skyline of S, i.e., $layer_1 = skyline(S)$ and $layer_2$ is the skyline of the complementary set of $layer_1$ in S, i.e., $layer_2 = skyline(S \backslash layer_1)$. Generally, $layer_i$ is the skyline of the complementary set of the first i-1 layers in S, i.e., $layer_i = skyline\left(S \backslash \bigcup_{j=1}^{i-1} layer_j\right)$. And MSL is defined as $\bigcup_{j=1}^{k} layer_j$, where k is the size of the group.*

3.1 Simultaneous Search in Each Dimension

According to the definition of MSL, we need to establish the first k layers by computing skyline k times. To replace this ineffective way, we devise a novel method to find each layer's points. Noticing that the information of each layer's points is equivalent to the information of each point's layer, we can construct each layer by finding which layer each point belongs to. The detailed procedure is introduced below.

Definition 3 (Dominance Domain). *Given a dataset S and a skyline $layer_i$, dominance domain of $layer_i$ is defined as: $D(layer_i) = S \backslash \bigcup_{j=1}^{i} layer_j$, or $D(layer_i) = \bigcup_{j=i+1}^{l} layer_j$, where l is the maximum layer number. When in two-dimensional case, dominance domain of $layer_i$ is the set of points in the upper right area of $layer_i$ intuitively.*

Property 1. *If a point $p \in D(layer_i)$, there is at least one point $p' \in layer_i$ making $p' \prec p$ and if $p \notin D(layer_i)$, there is no point $p' \in layer_i$, $p' \prec p$.*

Proof. *When $i > 1$, $layer_i$ is the skyline of $D(layer_{i-1})$. For certain $p \in D(layer_i)$, there is at least one sequence $\{p_1, p_2, \cdots, p_m\}$ in $D(layer_{i-1})$, satisfying $p_m \preceq \cdots \preceq p_1 \preceq p$. Since no point in $D(layer_{i-1})$ can dominate p_m, $p_m \in layer_i$. For certain $p \notin D(layer_i)$, then $p \in layer_j (j = 1, \cdots, i)$ and no points in $layer_i$ can dominate it. We can get the same result when $i = 1$.*

Property 2. *The layer of point p is equal to the maximal layer of the points that dominate p plus 1, i.e., $layer(p) = \max\{layer(p') | p' \prec p\} + 1$, and $layer(p) = 1$ if $\{p' | p' \prec p\} = \varnothing$.*

Proof. *According to Definition 3 and Property 1, for any point p in $layer_i$, where $2 \leq i \leq l$. For each m_1 ($1 \leq m_1 \leq i-1$), since $p \in D(layer_{m_1})$, there must be at least one point in $layer_{m_1}$ that dominates p. And for each m_2 ($i \leq m_2 \leq l$), since $p \notin D(layer_{m_2})$, no point in $layer_{m_2}$ can dominates p. We can see that, all points that can dominate p belongs to the first i-1 layers, and at least one in each. So, $\max\{layer(p') | p' \prec p\} = i - 1$.*

Lamma 1. *To determine the layer of certain point p, we just need to compare it with all points in a hyper-cuboid space region, which is in the range of the minimum of the i-th attribute and p[i] for each $1 \leq i \leq d$, containing all points that can dominate p (the lower left area of the current point p when in two-dimensional space).*

Example 1. *Considering the point p_8 in Figure 2, we just need to search the points in its lower left area (p_4, p_5, p_7) to determine $layer(p_8)$. According to Property 2, $layer(p_8) = \max_{i=4,5,7}\{layer(p_i)\} + 1 = 3$.*

We propose a framework for MSL based on the above properties. We determine a given point's layer by Property 2 and in order to find all points that can dominate p, we only need to check the hyper-cuboid region (the lower-left area in two dimensional case) according to Lemma 1. So our main idea is to slide a hyperplane over the points along each dimension concurrently and for each point visited, determine its layer and store it into the corresponding layer. The detailed algorithm is shown in Algorithm 1. We order all points d times by all d attributes (lines 6-7) and for each attribute, slide a hyperplane along the axis point by point in an increasing

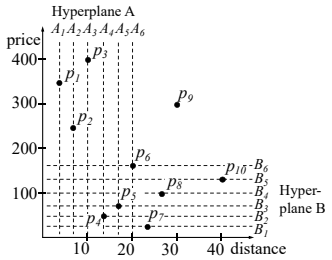
Algorithm 1: Concurrent multiple skyline layers algorithm

Input: a set of n points in d dimensional space ($S = \{p_1, p_2, \cdots, p_n\}$) and its size k.
Output: k skyline layers

```
1  for i = 1 to n do
2      S(i).times = 0;
3      S(i).layer = 1;
4  maxlayer(1..d) = 1;
5  layer(1..d)(1..k) = ∅;
6  for i = 1 to d do
7      sort the n points by the i-th dimension in ascending order and
         record the index O_i = {o_i1, o_i2, ..., o_in};
8  for i = 1 to n do
9      for j = 1 to d do
10         S(o_ji).times+ = 1;
11         flag = 0;
12         if S(o_ji).times == d && S(o_ji).layer == k then
13             flag = 1;
14             break;
15         if S(o_ji).times == 1 then
16             for layer_num = 0 to maxlayer(j)-1 do
17                 if there is a point in layer(j)(maxlayer(j)−
                     layer_num) can dominate S(o_ji) then
18                     S(o_ji).layer = maxlayer(j) − layer_num
                         + 1;
19                     break;
20         if S(o_ji).times ≤ k then
21             store S(o_ji) to layer(j)(S(o_ji).layer);
22             if S(o_ji).layer > maxlayer(j) then
23                 maxlayer(j) = S(o_ji).layer;
24     if flag == 1 then
25         break;
```

order (lines 9-25). When a point is visited by a hyperplane for the first time, the point is compared with all points that are already visited by this hyperplane to determine if it is dominated by any of them, and then labeled with the maximal layer number of the dominating points plus one (lines 15-19). The point is then stored in the corresponding dimension and layer (lines 20-23). When a point in the k-th layer has been visited by all d hyperplanes, the iteration stops (lines 12-14, lines 24-25).

Example 2. *Figure 3 illustrates the steps of constructing MSL in Figure 2. Figure 3(a) shows the location of the hyperplanes during each iteration. Figure 3(b) shows the intermediate result at each iteration ([ ] indicates one dimension and { } indicates one layer). The algorithm stops at B_6 when the current point p_6 is in the k-th layer and has been scanned d times by all hyperplanes. From this example, we can see that some points, e.g., p_4, p_5, and p_6, are visited twice by both hyperplanes and can be compared against different points for dominance. Will these comparisons return the same and correct result? Take p_6 as an example, when visited by A_6, it is compared with the points in the left area, notated with $S_1 = \{p_1, p_2, p_3, p_4, p_5\}$. Similarly, when visited by B_6, it is compared with points in the area below, notated with $S_2 = \{p_4, p_5, p_7, p_8, p_{10}\}$. While according to Lemma 1, we only need to check the points in the lower left area, notated with $S_0 = \{p_4, p_5\}$. Because $S_0 \in S_1$ and $S_0 \in S_2$, we can draw the same conclusion by checking either of the two sets. Consequently, we only check when a point is first visited. If a point has been visited already, we only save it in the corresponding dimension and layer without checking again.*



(a)

Hyperplane	Multiple skyline layers
A_1	{[p_1], []}, {[], []}, {[], []}
B_1	{[p_1], [p_7]}, {[], []}, {[], []}
A_2	{[p_1, p_2], [p_7]}, {[], []}, {[], []}
B_2	{[p_1, p_2], [p_7, p_4]}, {[], []}, {[], []}
A_3	{[p_1, p_2], [p_7, p_4]}, {[p_3], []}, {[], []}
B_3	{[p_1, p_2], [p_7, p_4]}, {[p_3], [p_5]}, {[], []}
A_4	{[p_1, p_2, p_4], [p_7, p_4]}, {[p_3], [p_5]}, {[], []}
B_4	{[p_1, p_2, p_4], [p_7, p_4]}, {[p_3], [p_5]}, {[], [p_8]}
A_5	{[p_1, p_2, p_4], [p_7, p_4]}, {[p_3, p_5], [p_5]}, {[], [p_8]}
B_5	{[p_1, p_2, p_4], [p_7, p_4]}, {[p_3, p_5], [p_5]}, {[], [p_8]}
A_6	{[p_1, p_2, p4], [p_7, p_4]}, {[p_3, p_5], [p_5]}, {[p_6], [p_8]}
B_6	{[p_1, p_2, p_4], [p_7, p_4]}, {[p_3, p_5], [p_5]}, {[p_6], [p_8, p_6]}

(b)

Figure 3: Example steps of Algorithm 1.

Advancement. Due to our novel structure, there are two main advancements in our work.

- One is that we reduce the number of points that need to be compared significantly. Our algorithm uses the same method with [4, 17, 21] to prune the comparison set of each point by ordering all points at the beginning. Besides this, we implement concurrent comparison in all dimensions, hence the number of points to investigate can be reduced significantly.
- Another advancement is that our work provides an explicit condition to stop the iterations. It is proved that only the points in the first k layers are necessary to construct group skylines, so the procedure can stop when points in the first k layers are all found and labeled. In previous works, it is very complicated to determine when to stop. In fact, in some works [4, 17, 21], they do not terminate the procedure until all points are enumerated in dataset S. But in our work, we have an explicit condition to stop the algorithm: when a point in the k-th layer is labeled by all d hyperplanes, we can interrupt the procedure.

Example 3. *Figure 4(a) gives example of MSL on an INDE synthetic dataset (points are even-distributed and independent among all attributes) with $k = 3$, $d = 2$, $n = 1000$. The investigation areas of our method are two narrow bands (the shadow area shown in Figure 4(a)) while the previous works need to investigate the whole dataset. When there is a point labeled by the two hyperplanes appears in the third layer, the procedure is interrupted.*

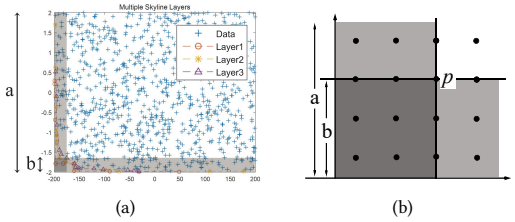


Figure 4: An example and a model for efficiency analysis.

Algorithm Efficiency. Now we give a rough analysis of the algorithm efficiency by comparing our method with that in [17], the state of the art in previous works. To simplify the problem, we assume the dataset is INDE, in which points are uniformly distributed and independent among all attributes. A simple case in two-dimensional space is investigated first and then the conclusion is extended to high-dimensional spaces. We normalize the value of each attribute into [0, a] to give a more concise presentation. For further simplification, we demonstrate a very simple model shown in Figure 4(b). The shadow area is swept by the two hyperplanes, and they intersect each other at point p at the end of the procedure. Obviously, the value of b is crucial in this problem and the case of the $b \times b$ square area depends on k only. Assuming there are l points on the edge of this area and we get the relationship $2(l-1)+1 = k$ when k is odd ($l = 3$, $k = 5$ in this example). Considering that points on the boundary just have 50% probability to fall in the area, the average number of points in the $b \times b$ area can be expressed as $n' = l^2 - (l-1) = \frac{k^2+3}{4}$. Since the density of points in the whole $a \times a$ area is constant, $b = \sqrt{\frac{n'}{n}}a$. When $k \ll n$, the investigation area in our work is $2ab - b^2 \approx \sqrt{\frac{k^2+3}{n}}a^2$. Let η be the proportionality coefficient of time complexity of our method and the previous one, whose investigation area is the whole dataset, $\eta = \frac{2ab-b^2}{a^2} \approx \sqrt{\frac{k^2+3}{n}} \ll 1$, we can see that our method is much faster. And with increasing group size k, $\eta \approx \frac{k}{\sqrt{n}}$. We can get the same conclusion when k is even.

Now consider a high-dimensional case, we need to study the b^d hyper-cuboid area. l is the number of points on the edge of this area and we have $d(l-1)+1 = k$ similarly. $\eta \approx \frac{d \cdot b a^{d-1}}{a^d}$ when neglecting overlap. Since $b = \sqrt[d]{\frac{n'}{n}}a$ and $n' = l^d - \frac{d}{2} \cdot l^{d-1}$, we can get $\eta \approx \frac{k}{\sqrt[d]{n}}$.

3.2 Subspace Skyline

We have introduced a framework to construct MSL in Subsection 3.1, but the search technique for each point is still inadequate. Comparing with all the points visited already for dominance when processing each new point is not efficient. In this section, we illustrate some detailed tricks, including utilizing skyline in subspace, and binary search, to make search more efficient.

A hyperplane mentioned in Subsection 3.1 is indeed a $d-1$ dimensional subspace, for instance, a line in planar space and a plane in three-dimensional space. When processing a point p, we can project the points that we need to compare p against to the subspace and utilize the properties of dominance to help us construct the MSL. Of special note is that though points in certain $layer_i$, which is the skyline of $D(layer_{i-1})$ in original space, cannot dominate each other, but their projection can, since one attribute is neglected in subspace.

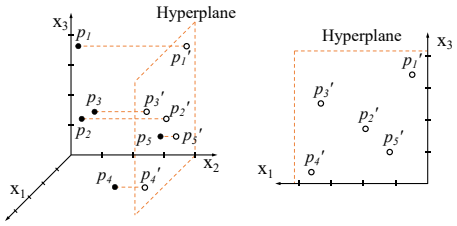


Figure 5: Points in original space and projected to subspace.

Property 3. *When points are ordered increasingly by certain attribute i and point p_2 is behind p_1. p_1' and p_2' are the projections of these two points in the i-th hyperplane. If $p_1' \prec p_2'$, then $p_1 \prec p_2$, and if $p_2' \prec p_1'$, the two points cannot dominate each other.*

Proof. *Since point p_2 is behind p_1, we know that $p_1[i] \le p_2[i]$. And since $p_1' \prec p_2'$, we know that $p_1[j] \le p_2[j]$ ($j = 1, \cdots, i-1, i+1, \cdots, m$), and $p_1[j] < p_2[j]$ for at least one j, so we know that $p_1 \prec p_2$. Similarly, when $p_2' \prec p_1'$, we can infer that no one can dominate the other.*

Theorem 1. *When investigating if current point can be dominated by certain points in the current layer in our framework, we just need to compare its projection with the subspace skyline of this layer.*

Proof. *According to Property 3, when searching if there is a point in certain layer can dominate current point, we just need to execute this procedure in the subspace. Notate current point as p_0 and the current layer as $layer_i$. According to the definition of skyline, for certain point p_1' not in $skyline_{s-s}(layer_i')$, there must be some points $p_2', \cdots, p_m'$ in $layer_i'$ making $p_m' \preceq \cdots p_2' \preceq p_1'$. Since no point can dominate p_m', $p_m' \in skyline_{s-s}(layer_i')$. So, if there is a point p_1' that can dominate current point p_0', there must be at least one corresponding skyline point p_m' dominating p_0', and vice versa. So, when investigating p_0', only points in $skyline_{s-s}(layer_i')$ need to be compared.*

Consolidating theories above, we give the detailed algorithm in Algorithm 2. To make the introduction more concise, we use the previous framework where the points are ordered by only one dimension. In this algorithm, we use a binary search to label the current point (lines 5-19). When searching if the current point is in certain layer, compare its projection with the skyline of this layer's projection in the subspace (line 6, line 9, and line 15, where $D_{s-s}(\,)$ is the dominance domain in subspace) and renew the subspace

Algorithm 2: Multiple skyline layers with subspace skyline.

Input: a set of n points in d-dimensional space.
Output: k skyline layers.

1 sort the n points by the first dimension in ascending order $S = \{p_{u1}, p_{u2}, \cdots, p_{un}\}$;
2 $p_{u1}.layer = 1$;
3 $max\_layer = 1$;
4 $sub\_skyline = \{\{p_{u1}\}\}$;
5 **for** $i = 2$ *to* n **do**
6 **if** p_{ui} *is in* $D_{s-s}(sub\_skyline(max\_layer))$ && $max\_layer \leq k$ **then**
7 $p_{ui}.layer = ++max\_layer$;
8 store p_{ui} to $sub\_skyline(max\_layer)$;
9 **else if** p_{ui} *is not in* $D_{s-s}(sub\_skyline(1))$ **then**
10 $p_{ui}.layer = 1$;
11 **if** *there are points in* $sub\_skyline(1)$ *dominated by* p_{ui} **then**
12 delete these points from $sub\_skyline(1)$;
13 store p_{ui} to $sub\_skyline(1)$;
14 **else**
15 use binary search to find $layer_j (1 < j < max\_layer)$ such that p_{ui} is in $D_{s-s}(sub\_skyline(j-1))$ but not in $D_{s-s}(sub\_skyline(j))$;
16 $p_{ui}.layer = j$;
17 **if** *there are points in* $sub\_skyline(j)$ *dominated by* p_{ui} **then**
18 delete these points from $sub\_skyline(j)$;
19 store p_{ui} to $sub\_skyline(j)$;

skyline after each search (line 8, lines 11-13, lines 17-19). Note that all dominance relationships mentioned in Algorithm 2 are in subspace. In fact, the tail point in [17] is a special case on a two-dimensional dataset of our subspace skyline.

Table 2: Example steps of Algorithm 2.

Current point	Skyline of $layer'_1$	Skyline of $layer'_2$	Multiple skyline layers
p_1	$\{p'_1\}$	$\{\}$	$\{p_1\}, \{\}$
p_2	$\{p'_1, p'_2\}$	$\{\}$	$\{p_1, p_2\}, \{\}$
p_3	$\{p'_1, p'_2\}$	$\{p'_3\}$	$\{p_1, p_2\}, \{p_3\}$
p_4	$\{p'_1, p'_2, p'_4\}$	$\{p'_3\}$	$\{p_1, p_2, p_4\}, \{p_3\}$
p_5	$\{p'_1, p'_4, p'_5\}$	$\{p'_3\}$	$\{p_1, p_2, p_4, p_5\}, \{p_3\}$

Example 4. *Table 2 shows the steps of Algorithm 2 with the example in Figure 5 ($n = 5$, $d = 3$, $k = 2$). Noticing p_5, we compare p'_5 with the subspace skyline of each layer ($\{p'_1, p'_2, p'_4\}$ and $\{p'_3\}$). Once it is labeled and added to the first layer, we renew the subspace skyline of the first layer (adding p'_5 and removing p'_2, since it is dominated by p'_5).*

3.3 MSL Algorithm

We introduce a framework based on simultaneous search in each dimension in Subsection 3.1 and detailed searching strategies in Subsection 3.2. The complete algorithm for MSL is a combination of the two algorithms. We search the points concurrently in each dimension, and in each dimension, we use binary search and subspace skyline to investigate each point.

Running Time. According to previous analysis in Subsection 3.1, the number of points we need to investigate is $\eta n \approx n^{\frac{d-1}{d}} k$. For those points that are not in the first k layers, we only need to compare it with the subspace skyline of the k-th layer. There are $\mathbb{T}_k$ points in it, so this part costs $O\left(\mathbb{T}_k n^{\frac{d-1}{d}} k\right)$. For each point p_j in the first k layers, we need to search and label it, the time complexity is $O\left(\sum_{i \in BS_j} \mathbb{T}_i\right)$, where BS_j is the set of layer index in the binary search for p_j and $|BS_j| = \log k$. When $n \gg \mathbb{S}_k$, we can ignore the influence of the first i layers, where $1 \leq i \leq k$, and regard the complementary set $D(layer_i)$ as an even-distributed and independent dataset. In this situation, all $\mathbb{T}_i$, $1 \leq i \leq k$, are approximately the same. Accordingly, the time complexity for investigating one points is $O(\mathbb{T}_k \log k)$ and the total cost of the second part is $O(\mathbb{T}_k \mathbb{S}_k \log k)$. Ignoring the preprocessing time, our algorithm requires $O\left(\mathbb{T}_k \left(n^{\frac{d-1}{d}} k + \mathbb{S}_k \log k\right)\right)$ time in total for constructing MSL. Noticing that $\mathbb{T}_k = 1$ in the two-dimensional space, the time complexity becomes $O\left(\sqrt{n}k + \mathbb{S}_k \log k\right)$.

4 FINDING G-SKYLINE GROUPS

In this section, we devise a novel structure for finding G-skyline groups.

Definition 4 (G-Skyline). *Given a dataset S of n points in d-dimensional space. $G = \{p_1, p_2, \cdots, p_k\}$ and $G' = \{p'_1, p'_2, \cdots, p'_k\}$ are two different groups with k points. If we can find two permutations of the k points for G and G', $G = \{p_{u1}, p_{u2}, \cdots, p_{uk}\}$ and $G' = \{p'_{u1}, p'_{u2}, \cdots, p'_{uk}\}$, letting $p_{ui} \preceq p'_{ui}$ for all i ($1 \leq i \leq k$) and $p_{ui} \prec p'_{ui}$ for at least one i, we say G **g-dominates** G'. All groups containing k points that are not g-dominated by any other group with the same size compose G-skyline [17].*

G-skyline is established by MSL, however we observed that not all layers in MSL contribute equally to the group skylines. The first layer is much more important. Groups that consist of only the first layer points take a great proportion, especially when k is small.

Definition 5 (Primary Group). *k-point groups consisting of only $layer_1$ points in MSL are primary groups. According to the definition of G-skyline, all primary groups are G-skyline groups because every individual point cannot be dominated.*

Definition 6 (Secondary Group). *G-skyline groups that are not primary groups are defined as secondary groups.*

Example 5. *On our synthetic INDE dataset ($n = 1000$, $d = 4$, $k = 3$), experiment shows that there are 76 points in $layer_1$, 152 points in $layer_2$, 189 points in $layer_3$, and 75034 G-skyline groups. The number of primary groups and secondary groups is $\binom{76}{3} = 68450$ and 6584 respectively. We can see that the number of primary groups dominates with an overwhelming advantage.*

Table 3 shows the percentage of primary groups on different dataset, with various parameters. We can see that the primary groups occupy a large proportion in most cases, especially with a small k and a large d. This phenomenon motivates the following.

Table 3: Percentage of primary groups.

dataset		corr				inde				anti			
d		2	4	6	8	2	4	6	8	2	4	6	8
k	2	62.5	97.6	98.8	99.5	80.3	97.9	99.6	99.9	98.1	99.9	99.8	99.9
	3	18.6	87.8	97.8	98.9	38.2	91.2	98.0	99.3	91.0	98.9	–	–
	4	2.1	84.3	–	–	16.9	87.4	–	–	88.2	–	–	–

First, primary groups, which are combinations of k points in $layer_1$, dominate G-skyline groups in number. It is important to give priority to enumerate all these combinations in the most direct and efficient way. Second, though there are significantly more points in later layers, they contribute very little to the G-skyline groups. Most of the groups containing points in later layers are not desired. So, an efficient pruning strategy is desired.

Considering these motivations, we introduce two new approaches. Our work is based on the PWise and UWise+ algorithms in [17], but we propose a novel structure utilizing a combination queue to improve the efficiency.

To do this, we also need to construct the directed skyline graph (DSG), a graph that saves all points in MSL and the dominance relationship among them, to help us find G-skyline groups. In this graph, each node is for a sample point and each edge is for the dominance relationship between a pair of points.

4.1 Fast PWise Algorithm

In this subsection, we propose the fast PWise algorithm (F_PWise) to construct G-skyline based on the combination queue. Besides the previous pruning strategies in PWise algorithm, we give the edge pruning for DSG to make the algorithm more effective.

Edge Pruning. For any two points p_i and p_j in DSG (p_j is in the tail set of p_i), if $p_j.layer - p_i.layer > 1$, delete the edge between them. Also, take Figure 2 as an example, we can delete edge (p_4, p_8) because we can connect p_4, p_8 by (p_4, p_5) and (p_5, p_8). We can also delete (p_7, p_8) even though this is the only connection between these points, because when adding p_8 to a group containing p_7, it must contain p_5 and p_4 too or the new group will be a non-skyline group and removed by subtree pruning [17]. We can find p_8 in the child set of p_5 hence the connection between p_7 and p_8 is not necessary.

Definition 7 (Combination Queue). *A queue to enumerate combinations by adding elements in the tail set of current set.*

We will enumerate all the combinations, which are not larger than k in size, of $layer_1$ points with a combination queue. It can output all the primary groups and can also be used to find secondary groups in the next procedure. The detailed algorithm is shown in Algorithm 3. The algorithm prunes the DSG by preprocessing and edge pruning (line 1). In preprocessing, we remove the nodes with more than $k-1$ parents from the DSG, because if a point is in certain G-skyline group, all its parents need to be contained in it. A combination queue is used to find all primary groups (line 2). Lines 3-14 show the procedure to find the secondary group with the intermediate results of the combination queue. Lines 9-13 show the subtree pruning since a group which is not an $i-1$-size G-skyline group cannot be an i-size G-skyline group by adding a point from its tail set.

Algorithm 3: Fast PWise algorithm

Input: a DSG and group size k.
Output: G-skyline.

```
1  prune DSG with preprocessing and edge pruning;
2  construct queue Q to output all primary groups;
3  list_point = 0;
4  while list_point < lengthofQ do
5      for each p_i in Q(list_point) do
6          for each p_j in p_i's children set in DSG do
7              if p_j is in T_DSG(Q(list_point)) then
8                  combine candidate_list(list_point) and p_i as g;
9                  if g is a skyline group then
10                     if there are k points in G then
11                         output g;
12                     else
13                         add g to Q;
14             list_point++;
```

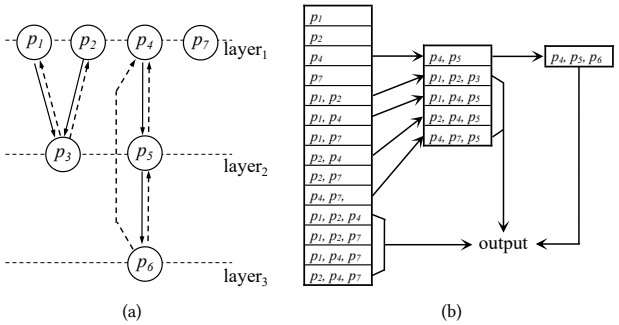


Figure 6: Example steps of Algorithm 3.

Example 6. *Figure 6 uses MSL in Figure 2 as an example to show the steps of Algorithm 3. Figure 6(a) shows a DSG, where solid line arrows point to child nodes and dotted arrows point to parent node. Node p_8 and edge (p_4, p_6) are all removed (only the solid line arrow is removed, the dotted arrows is reserved for subtree pruning). In Figure 6(b), the first column is a queue to enumerate all primary groups and the later two columns find secondary groups by adding points in $layer_2$ and $layer_3$ to groups in the first column.*

4.2 Fast UWise Algorithm

In this subsection, we propose the fast UWise algorithm (F_UWise) by adding combination queue to UWise+ algorithm. UWise+ method construct G-skyline by adding unit groups, which are defined as groups composed of certain point and all its parents in DSG. We find all primary groups by the combination queue and secondary groups with UWise+ method.

Unlike F_PWise, F_UWise gives primary groups and secondary groups independently by combination queue and UWise+ method respectively. It is a simple combination of these two methods but also has good performance.

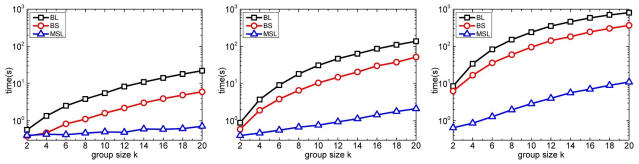
5 EXPERIMENTS

In this section, we demonstrate the effectiveness and the efficiency of our proposed methods on the synthetic and real world datasets. All algorithms were implemented in python. We conducted the experiments on a PC with Intel Core i7 2.8GHz processors, 512K L2 and 3M L3 cash, and 8G RAM.

Datasets. We performed each algorithm on synthetic datasets and a real NBA dataset. To examine the scalability of our methods, we generated independent (INDE), correlated (CORR), and anti-correlated (ANTI) datasets. These three types of data distribution are first proposed by [5] to simulate different actual application scenarios. The data of NBA players were crawled from http://stats.nba.com. There are five attributes to investigate players' performance, points (PTS), rebounds (REB), assists (AST), steals (STL), and blocks (BLK).

5.1 MSL on the Synthetic Data

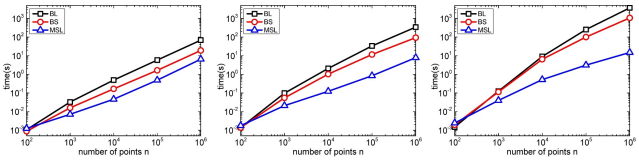
In this section, we run the approaches for MSL on synthetic datasets. Figures 8-10 show the running time of MSL algorithm (presented in Subsection 3.3), the binary search algorithm (BS) [17], and the baseline approach (BL, iteratively compute and remove each skyline layer).



(a) (b) (c)

Figure 7: Computing MSL on synthetic datasets of varying k.

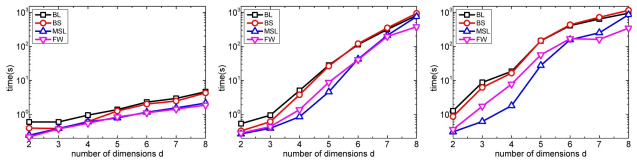
Figure 7 shows the time cost of each approach with varying group size k on three different datasets respectively ($n = 100000, d = 3$). It indicates that our approach outperforms others especially when the output size is large. When k is very small, there are very few points in MSL (Figure 7(a), $k = 2$, as an example), ordering before building MSL spends much time so that our approach is not the fastest. But with the increasing of k, the advantage of our approach becomes more and more significant.



(a) (b) (c)

Figure 8: Computing MSL on synthetic datasets of varying n.

Figure 8 presents the time cost of each approach with varying number of points n ($d = 3, k = 10$). For each approach, the running time grows almost linearly with the increasing of the point number n since the amount of computation is almost proportional to the scale of the dataset. Also, when output size is very small, our approach does not perform the best due to the overhead cost of ordering.



(a) (b) (c)

Figure 9: Computing MSL on synthetic datasets of varying d.

Figure 9 illustrates the running time of each approach with different number of dimensions d ($n = 100000, k = 2$). The result shows that the efficiency advantage of our approach decreases with the increasing of d. It is because in our search approach (shown in Algorithm 2), we save the subspace skyline of each MSL layer instead of the layer itself, and compare with them when investigating a new point. We reduce the number of points to store and compare to save time but we waste some to update the subspace skyline each time. However, both theoretical analysis and experiment show that the scale of skyline will grow intensively with the increasing of d since a large number of attributes means difficulty for points to dominate each other (and the same situation in the subspace). In this case, the time wasted for skyline updating is more than the time gained in search strategy. When d is large, the framework (FW) only, shown in Algorithm 1, performs better than the whole MSL algorithm.

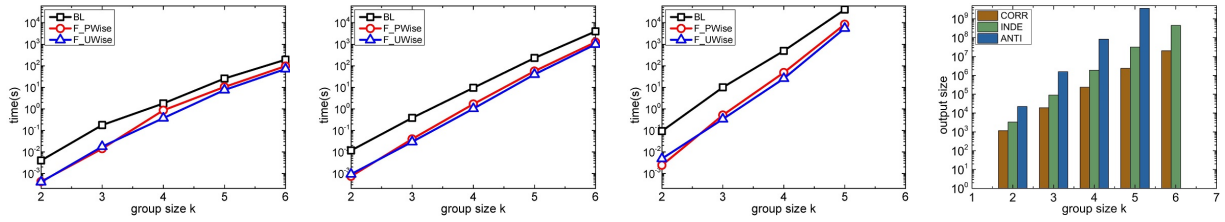
5.2 G-skyline on the Synthetic Data

In this section, we show the experimental result of the proposed approaches for computing G-skyline. F_PWise and F_UWise are executed and the baseline approach (BL) is the UWise+ algorithm, which is the best performing algorithm in [17]. To show the effectiveness of our G-skyline algorithms, we only compare the time after building MSL in the experiments, to eliminate the effect of our proposed MSL algorithm.

All existing approaches for G-skyline return a candidate set that is too large to be useful. In fact, if G-skyline is used for data pruning, primary groups are not necessary to return as result. This is because when investigating if a group is a primary group, we can search if all its points are in the skyline ($k\mathbb{S}_1$ times for the worst case) rather than searching if it is in G-skyline (more than $\binom{\mathbb{S}_1}{k}$ times for the worst case). Only when investigating if it is a secondary group, it needs to be searched in G-skyline. So, we just output secondary groups in our methods and the result shows the time cost of each approach.

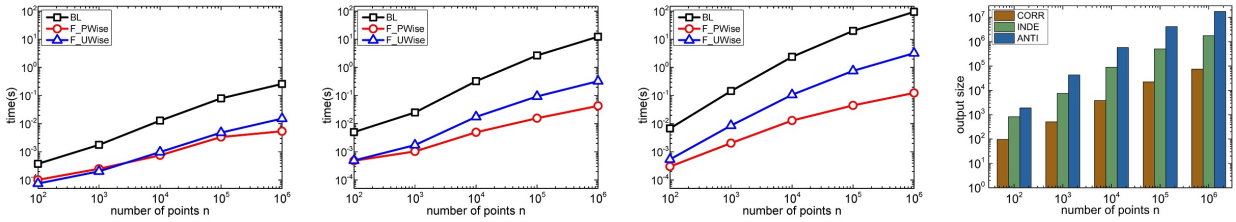
Figures 10(a)(b)(c) show the time cost of BL, F_PWise and F_UWise and Figure 10(d) shows the output size with varying group size k ($n = 100000, d = 3$). We can see that the increasing of output size is almost exponential with the increasing of the group size k, accordingly the running time of each approach also increases exponentially with it.

Figures 11(a)(b)(c) show the time cost of each approach for G-skyline with varying number of points n ($d = 5, k = 2$). Figure 11(d) shows the output size on three different datasets. Distinctly, the time cost and the output size increase approximately linearly with the increasing of n.



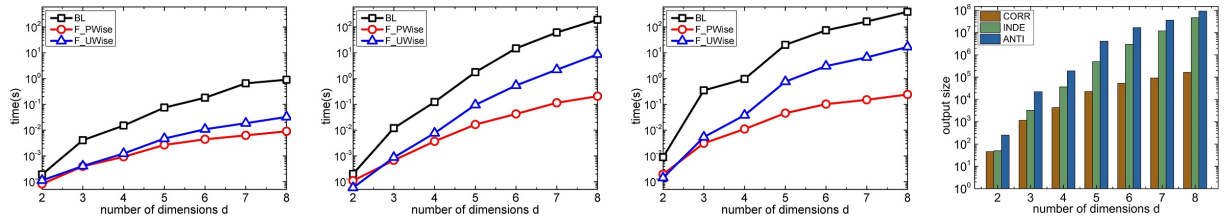
(a) (b) (c) (d)

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



(a) (b) (c) (d)

Figure 11: Computing G-skyline on synthetic datasets of varying n.



(a) (b) (c) (d)

Figure 12: Computing G-skyline on synthetic datasets of varying d.

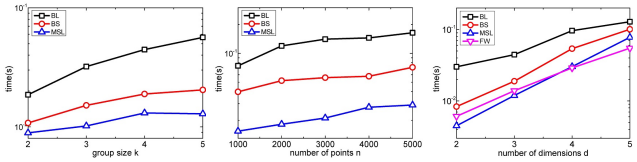
Figures 12(a)(b)(c) show the time cost with varying number of dimensions d ($n = 100000, k = 2$) and Figure 12(d) shows the output size. The time cost and the output size increase approximately exponentially with the increasing of d.

Unit group based approach is a pretty artful way for the G-skyline problem. It fits the property of G-skyline group well and UWise+ usually performs better than PWise. But in our new approaches, F_PWise performs much better than F_UWise in most situations, especially in a large output size. It may be because there are many overlaps between each unit groups. When the output size is large, there are a large number of unit groups to search and the overlap issue becomes serious whereas our point-based method is enhanced dramatically by the edge pruning.

5.3 MSL on the NBA Data

In this section we implemented all algorithms on a real NBA dataset. To get more data, we regarded one player in different stages as different samples. For example, a player's per game points in noontide and in low tide period are quite different. We gathered 5000 records of players after filtering out some inferior ones.

Figure 13 shows the influence of different parameters on the time cost of MSL on the NBA dataset. Figure 13(a) represents the variation of the running time with the impact of group size k



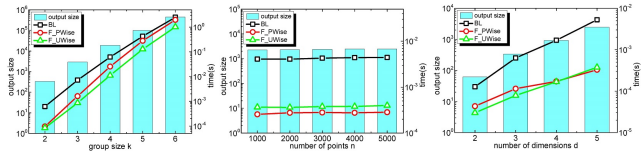
(a) (b) (c)

Figure 13: Computing MSL on synthetic datasets of varying parameters.

($n = 5000, d = 3$), we can see that our approach performs better than previous approaches. The variation of the time cost with the impact of dataset size n is presented in Figure 13(b) with $d = 3$ and $k = 10$. The time cost does not vary intensively with the varying n due to the "saturation" of each MSL layer. And the variation of the running time with the impact of dimension size d is represented in Figure 13(c) when $n = 5000$ and $k = 5$. As we have analyzed in Subsection 5.2, the efficiency advantage of our approach decreases distinctly with the increase of d because of the dramatical increase of $\mathbb{T}_k$. Our approach costs $O\left(\mathbb{T}_k\left(n^{\frac{d-1}{d}}k + \mathbb{S}_k \log k\right)\right)$ time and becomes inefficient in high-dimensional space. So, in this case, we can implement framework (FW, shown in Algorithm 1) only instead of the whole MSL algorithm.

5.4 G-skyline on the NBA Data

In this subsection we implemented BL (UWise+), F_UWise, and F_PWise on NBA dataset and report the result. Figure 14 shows the output size and the time cost of G-skyline on NBA dataset with different parameters.



(a) (b) (c)

Figure 14: Computing G-skyline on NBA dataset with varying parameters.

The line chart in Figure 14(a) shows the variation of the running time for G-skyline with the impact of group size k ($n = 5000, d = 3$). The improvement of our approaches is not so significant compared with that in other cases shown in Figure 14(b) or Figure 14(c), it may be because that our approaches omit primary groups, which account for a lower proportion when there are many layers in MSL (shown in Table 3). As a result, the time complexity is the same order of magnitude with the baseline. Another interesting phenomenon is that in most situation, F_PWise performs better, especially with a large n or d, however F_UWise generally performs better in this situation. The reason may be that with a large k, there are too many children of each node in DSG and enumerating them takes significant time in F_PWise. And in F_UWise, the overlap issue, mentioned in Subsection 5.3, is not so serious. The histogram in Figure 14(a) is the illustration of output size with different group size k.

The line chart in Figure 14(b) illustrates the variation of the time consumption with the impact of dataset size n with $d = 5$ and $k = 2$ and the histogram in Figure 14(b) illustrates the variation of the output size. We can observe that the output size does not vary prominently. The reason may be that the NBA dataset is correlated, so the skyline becomes "saturate" even with the increase of the dataset size, hence the running time and the output size of G-skyline keep constant.

The line chart in Figure 14(c) presents the variation of the running time with the impact of dimension size d when $n = 5000$ and $k = 5$. Figures 12(a)(b)(c) and the line chart in Figure 14(c) show that with the increase of d, F_Pwise becomes more and more efficient and surpasses F_UWise eventually. The reason may be that when there are more attributes, it is harder for a point to dominate another and there are fewer child nodes of each node in DSG, so the enumeration complexity reduces and F_PWise becomes more efficient.

6 CONCLUSIONS

In this paper, we proposed several novel structures to address the group skyline problem. First, we developed a novel algorithmic technique to build multiple skyline layers using concurrent search and subspace skyline properties. Then, we developed two new methods to find G-skyline by dividing the G-skyline groups into two categories, primary groups and secondary groups. Experimental results show that the proposed algorithms perform several orders of magnitude better than the baseline method in most situations.

ACKNOWLEDGMENTS

This research is supported in part by the AFOSR DDDAS Program under AFOSR grant FA9550-12-1-0240.

REFERENCES

[1] Wolf Tilo Balke, Ulrich Gntzer, and Jason Xin Zheng. 2004. Efficient Distributed Skylining for Web Information Systems.. In *Advances in Database Technology - EDBT 2004, International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, Proceedings*. 256–273.
[2] Jon Louis Bentley. 1980. Multidimensional divide-and-conquer. *Communications of The ACM* 23, 4 (1980), 214–229.
[3] Jon Louis Bentley, H T Kung, Mario Schkolnick, and Clark D Thompson. 1978. On the Average Number of Maxima in a Set of Vectors and Applications. *J. ACM* 25, 4 (1978), 536–543.
[4] Henrik Blunck and Jan Vahrenhold. 2010. In-Place Algorithms for Computing (Layers of) Maxima. *Algorithmica* 57, 1 (2010), 1–21.
[5] S Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The Skyline operator. (2001), 421–430.
[6] Beechung Chen, Kristen Lefevre, and Raghu Ramakrishnan. 2007. Privacy skyline: privacy with multidimensional adversarial knowledge. *very large data bases* (2007).
[7] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with presorting. (2003), 717–719.
[8] Evangelos Dellis and Bernhard Seeger. 2007. Efficient computation of reverse skyline queries. *very large data bases* (2007).
[9] Parke Godfrey, Ryan Shipley, and Jarek Gryz. 2005. Maximal vector computation in large data sets. *very large data bases* (2005), 229–240.
[10] Ibrahim Gomaa and Hoda MO Mokhtar. Computing Continuous Skyline Queries without Discriminating between Static and Dynamic Attributes. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering* 10, 12 (????), 1996–2001.
[11] Hyeonseung Im and Sungwoo Park. 2012. Group skyline computation. *Information Sciences* 188 (2012), 151–169.
[12] David Kirkpatrick and Raimund Seidel. 1985. Output-size sensitive algorithms for finding maximal vectors. (1985), 89–96.
[13] Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting stars in the sky: an online algorithm for skyline queries. *very large data bases* (2002), 275–286.
[14] H T Kung, Fabrizio Luccio, and Franco P Preparata. 1975. On Finding the Maxima of a Set of Vectors. *J. ACM* 22, 4 (1975), 469–476.
[15] Chengkai Li, Nan Zhang, Naeemul Hassan, Sundaresan Rajasekaran, and Gautam Das. 2012. On skyline groups. (2012), 2119–2123.
[16] He Li, Sumin Jang, and Jaesoo Yoo. 2011. An efficient multi-layer grid method for skyline queries in distributed environments. (2011), 112–119.
[17] Jinfei Liu, Li Xiong, Jian Pei, Jun Luo, and Haoyu Zhang. 2015. Finding Pareto optimal groups: group-based skyline. *Proceedings of The Vldb Endowment* 8, 13 (2015), 2086–2097.
[18] Jinfei Liu, Li Xiong, and Xiaofeng Xu. 2014. Faster output-sensitive skyline computation algorithm. *Inform. Process. Lett.* 114, 12 (2014), 710–713.
[19] Jinfei Liu, Juncheng Yang, Li Xiong, and Jian Pei. 2017. Secure Skyline Queries on Cloud Platform. In *IEEE International Conference on Data Engineering*. 633–644.
[20] Jinfei Liu, Haoyu Zhang, Li Xiong, Haoran Li, and Jun Luo. 2015. Finding Probabilistic k-Skyline Sets on Uncertain Data. (2015), 1511–1520.
[21] Hua Lu, Christian S Jensen, and Zhenjie Zhang. 2011. Flexible and Efficient Resolution of Skyline Query Size Constraints. *IEEE Transactions on Knowledge and Data Engineering* 23, 7 (2011), 991–1005.
[22] Michael Morse, Jignesh M Patel, and H V Jagadish. 2007. Efficient skyline computation over low-cardinality domains. *very large data bases* (2007), 267–278.
[23] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *international conference on management of data* 30, 1 (2005), 41–82.
[24] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. 2007. Probabilistic skylines on uncertain data. *very large data bases* (2007), 15–26.
[25] Mehdi Sharifzadeh and Cyrus Shahabi. 2006. The spatial skyline queries. *very large data bases* (2006), 751–762.
[26] Kianlee Tan, Pinkwang Eng, and Beng Chin Ooi. 2001. Efficient Progressive Skyline Computation. *very large data bases* (2001), 301–310.
[27] Ping Wu, Caijie Zhang, Ying Feng, Ben Y Zhao, Divyakant Agrawal, and Amr El Abbadi. 2006. Parallelizing skyline queries for scalable distribution. *extending database technology* 3896 (2006), 112–130.
[28] Tian Xia and Donghui Zhang. 2006. Refreshing the sky: the compressed skycube with efficient support for frequent updates. (2006), 491–502.
[29] Nan Zhang, Chengkai Li, Naeemul Hassan, Sundaresan Rajasekaran, and Gautam Das. 2014. On Skyline Groups. *IEEE Transactions on Knowledge and Data Engineering* 26, 4 (2014), 942–956.