Top-*k* Dominating Queries on Skyline Groups

Haoyang Zhu, Xiaoyong Li, *Member, IEEE*, Qiang Liu, *Member, IEEE*, and Zichen Xu, *Member, IEEE*

Abstract—The top-k dominating (TKD) query on skyline groups returns k skyline groups that dominate the maximum number of points in a given data set. The TKD query combines the advantages of skyline groups and top-k dominating queries, thus has been frequently used in decision making, recommendation systems, and quantitative economics. Traditional skylines are inadequate to answer queries from both *individual* and *groups* of points. The group size could be too large to be processed in a reasonable time as a single operator (i.e., the skyline group operator). In this paper, we address the performance problem of grouping for TKD queries in skyline database. We formulate the problem of grouping, define the group operator in skyline, and propose several efficient algorithms to find top-k skyline groups. Thus, we provide a systematic study of TKD queries on skyline groups and validate our algorithms with extensive empirical results on synthetic and realworld data.

Index Terms—Top-k dominating queries, skyline queries, skyline groups, query processing

1 Introduction

The skyline query [1] is widely used in multi-criteria optimal decision making applications, which aims at retrieving points that are not dominated by other points in a data set. Given two multi-dimensional points P and Q, P dominates Q *iff* P is not worse than Q in all dimensions but strictly better than Q in at least one dimension. Assume that we have a data set D of n d-dimensional points. Q^i denotes the ith point and $Q^i = (Q^i_1, Q^i_2, \ldots, Q^i_d)$. If *large* values are preferred, then Q^i dominates Q^j, denoted as $Q^i \succ Q^j$, *iff* for each λ, $Q^i_\lambda \geq Q^j_\lambda$ and for at least one λ, $Q^i_\lambda > Q^j_\lambda$ $(1 \leq \lambda \leq d)$. Fig. 1 shows a skyline example. The data set in Fig. 1 (left) consists of 10 points and each point has three dimensions d_1, d_2 and d_3. We can see that $Q^1(10, 0, 0) \succ Q^4(7, 0, 0)$ as an example of dominance relationship between points. As shown in Fig. 1 (right), the skyline contains Q^1, Q^2, Q^3 and Q^7.

Though skyline computation is particularly useful in multi-criteria decision making applications, it is inadequate to answer queries that need to analyze not only *individual* points but also their *combinations* [2], [3], [4], [5], [6]. Specifically, in many real-world applications, we need to find groups of points that are not dominated by other groups of equal size. For instance, in the fantasy sports, a gamer forms his or her team by selecting athletes in the pool of available athletes. Thus a team may consist of athletes who are not in the same real-world team. Each athlete is represented as a point consisting of several statistical categories. Obviously, the gamer would like to form a team that cannot be dominated by other teams.

As shown in [2], [3], [4], [5], [6] that a skyline group may consist of both skyline points and non-skyline points, all points in the data set have a chance to form a skyline group. Assume that a group consists of l points in the data set. Thus there are total C^l_n combinations, which are far more than the n candidates in traditional skyline computation. Though skyline group operator is powerful in pruning the candidate groups, the output size of skyline groups may be still significantly large. The experimental results proposed in [2], [3], [4], [5], [6], [7] show that the output size is million scale even when the input is a few thousand points. The large output size is less informative and it may be hard for users to make a good selection fast enough.

Motivation. The enormous output size urges us to design efficient algorithms to select the best k skyline groups. Such k skyline groups should be the most representative. Inspired by the top-k dominating queries [8], [9] and representative skyline [10], we quantify the concept of "representative" by counting the number of points dominated by the group. We find that in many real-world applications such as recommendation systems, the need to select top-k groups of points is in great demand. Moreover, solving this problem efficiently is very challenging. To facilitate the presentation, we define that a point Q is dominated by a group G *iff* there exists at least one point Q' in G satisfying $Q' \succ Q$. We also define a function $score(G)$ that counts the number of the points dominated by group G.

An intuitive method for supporting the TKD queries on skyline groups is to conduct existing skyline groups algorithms to get all skyline groups, then compute the score for each skyline group and return the k skyline groups with the highest scores. Obviously, the approach is inefficient due to

- *H. Zhu is with the Institute of Systems Engineering, Academy of Military Sciences, People's Liberation Army, Beijing 100091, China. E-mail: zhuhaoyang@nudt.edu.cn.*
- *X. Li is with the College of Meteorology and Oceanography, National University of Defense Technology, Changsha 410073, China. E-mail: sayingxmu@nudt.edu.cn.*
- *Q. Liu is with the College of Computer, National University of Defense Technology, Changsha 410073, China. E-mail: qiangliu06@nudt.edu.cn.*
- *Z. Xu is with the College of Computer Science and Technology, Nanchang University, Nanchang 330031, China. E-mail: xuz@ncu.edu.cn.*

Manuscript received 13 June 2017; revised 18 Nov. 2018; accepted 4 Mar. 2019. Date of publication 8 Mar. 2019; date of current version 3 June 2020. (Corresponding author: Xiaoyong Li.)
Recommended for acceptance by K. S. Candan.
Digital Object Identifier no. 10.1109/TKDE.2019.2904065

Points	d_1	d_2	d_3
Q^1	10	0	0
Q^2	1	7	1
Q^3	2	3	3
Q^4	7	0	0
Q^5	1	5	0
Q^6	0	4	1
Q^7	0	0	4
Q^8	1	1	2
Q^9	0	1	3
Q^{10}	1	0	3

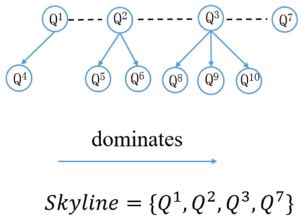


Fig. 1. A skyline example.

two aspects. First, it is quite expensive to compute skyline groups using existing algorithms especially when the group size l and data set size $|D|$ are large. Second, unlike TKD queries on individual points in which we only need to compute the score of a point once, we need to check points dominated by a point in skyline groups multiple times. Because $score(G)$ does not equal to $\sum_{Q \in G} score(Q)$ in most cases. This will lead to heavy double counting problems. Therefore, computing scores for groups is much more complicated and expensive.

To address the first aspect, we develop various pruning techniques to boost query efficiency. Unlike the intuitive method which needs to materialise all skyline groups, our algorithms can return top-k dominating skyline groups without materialising all skyline groups. This greatly improves the efficiency of TKD queries on skyline groups. To address the second aspect, we design a bitmap index method to compute the scores of groups, which significantly cuts down the cost of score computation. Moreover, we integrate bitmap compression techniques proposed in [11] with our bitmap index method, which significantly reduces the cost of space. We briefly summarize our main contributions as follows:

- We formalize the novel problem of TKD queries on skyline groups. To the best of our knowledge, we are the first to address this problem.
- We develop effective pruning techniques for processing TKD queries on skyline groups under all existing skyline groups definitions. With these techniques we can return top-k dominating skyline groups without materialising all skyline groups.
- We design a bitmap index method to compute the scores of groups efficiently, using bitmap compression techniques to minimize the size of bitmap index.
- We conduct extensive experiments on both real and synthetic data sets to validate the effectiveness and efficiency of our proposals.

This paper extends a conference paper [12] in several substantial ways. First, compared to [12] which needs to generate all skyline groups, we develop novel effective pruning techniques with which we can return top-k dominating skyline groups without generating all skyline groups. Second, bitmap index compression techniques are integrated into the implementation to cut the space cost. Finally, we conduct much more experiments to evaluate the performance of our proposed algorithms.

The rest of the paper is organized as follows. We formalize the problem in Section 2 and review related works in Section 3. Section 4 elaborates our algorithms for TKD queries on skyline groups. We propose our bitmap index method to compute the scores of groups and compression techniques in Section 5. We present experimental results in Section 6 and we conclude our paper in Section 7.

TABLE 1
The Summary of Notations

Notation	Description
D	A d-demonical data set
d	Number of dimensions
n	Number of points in D
k	k in top-k
l	Size of a group
Q^i	The ith point in D
Q^i_j	The value on the jth dimension of Q^i
$\succ$	Preference/dominance relation
$Skyline$	The skyline of data set D
$GSkyline$	The skyline groups of data set D
S_k	Set of top-k dominating skyline groups
$score(G)$	Number of points dominated by G

2 Problem Definition

In this section, we introduce the problem definition. For reference, a summary of frequently used notions is given in Table 1.

First, we introduce the definitions of dominance relationship between groups defined in [2], [3], [4], [5], [6]. We use $\succ$ to denote the dominance relationship between groups. Let $G \succ G'$ denote G dominates G'. The dominance relationship between groups defined in existing works can be divided into two kinds.

Definition 1. ($\succ_p$) *[5] Assume that $G = \{Q^1, Q^2, \ldots, Q^l\}$ and $G' = \{Q'^1, Q'^2, \ldots, Q'^l\}$ are two different groups with l points. We say that $G \succ_p G'$,* iff *there exist two permutations of the l points for G and G', $G = \{Q^{u1}, Q^{u2}, \ldots, Q^{ul}\}$ and $G' = \{Q'^{u1}, Q'^{u2}, \ldots, Q'^{ul}\}$ satisfying that for each i, $Q^{ui} \succeq Q'^{ui}$ and for at least one i, $Q^{ui} \succ Q'^{ui}$ $(1 \leq i \leq l)$.*

For instance in the Fig. 1, since $Q^1 \succ Q^4$ and $Q^2 \succ Q^5$, thus $\{Q^1, Q^2\} \succ_p \{Q^4, Q^5\}$.

Definition 2. ($\succ_f$) *[2], [3], [4], [6] For an aggregate function f and a group $G = \{Q^1, Q^2, \ldots, Q^l\}$, then G is represented by a point Q, where $Q_j = f(Q^1_j, Q^2_j, \ldots, Q^l_j)$. For two distinct groups G and G', Q and Q' represent G and G' respectively. We define $G \succ_f G'$* iff *$Q \succ Q'$.*

In this paper, we study two kinds of aggregate function. The first one is strictly monotone, which means $f(Q^1_j, Q^2_j, \ldots, Q^l_j) > f(Q^{1'}_j, Q^{2'}_j, \ldots, Q^{l'}_j)$ if $Q^i_j \geq Q^{i'}_j$ for each $i \in [1, l]$ and $\exists\lambda$ such that $Q^\lambda_j > Q^{\lambda'}_j$, where $1 \leq \lambda \leq l$. For the strictly monotone function, we study SUM in this paper. We also investigate aggregate functions that are not strictly monotone such as MAX and MIN. Table 2 shows the dominance relations under different aggregate functions.

Based on the Definition 1 or Definition 2, skyline group is defined as follows:

Definition 3. (skyline groups) *For a given group size l, a l-point group is a l-point skyline group, or skyline group in short, if and only if it is not dominated by any other l-point*

TABLE 2
Dominance Relations under Different Aggregate Functions

	Points			*SUM*	*MAX*	*MIN*
G	Q^2(1,7,1)	Q^3(2,3,3)	Q^4(7,0,0)	(10,10,4)	(7,7,3)	(1,0,0)
G'	Q^3(2,3,3)	Q^4(7,0,0)	Q^5(1,5,1)	(10,8,4)	(7,5,3)	(1,0,0)
Dominance Relation				$G \succ_f G'$	$G \succ_f G'$	$G = G'$

TABLE 3
Skyline Groups under Different Definitions

$\succ$	Skyline Groups
Permutation	$\{Q^2, Q^3\}$, $\{Q^1, Q^3\}$, $\{Q^1, Q^2\}$, $\{Q^3, Q^9\}$, $\{Q^3, Q^8\}$, $\{Q^3, Q^{10}\}$, $\{Q^3, Q^7\}$, $\{Q^2, Q^7\}$, $\{Q^2, Q^5\}$, $\{Q^2, Q^6\}$, $\{Q^1, Q^7\}$, $\{Q^1, Q^4\}$
SUM	$\{Q^2, Q^3\}$, $\{Q^1, Q^3\}$, $\{Q^3, Q^7\}$, $\{Q^1, Q^2\}$, $\{Q^3, Q^9\}$, $\{Q^3, Q^{10}\}$, $\{Q^3, Q^8\}$, $\{Q^2, Q^7\}$, $\{Q^2, Q^6\}$, $\{Q^2, Q^5\}$, $\{Q^1, Q^7\}$, $\{Q^1, Q^4\}$
MAX	$\{Q^2, Q^3\}$, $\{Q^1, Q^3\}$, $\{Q^1, Q^2\}$, $\{Q^3, Q^7\}$, $\{Q^2, Q^7\}$, $\{Q^1, Q^7\}$
MIN	$\{Q^2, Q^3\}$, $\{Q^3, Q^{10}\}$, $\{Q^3, Q^8\}$, $\{Q^3, Q^9\}$, $\{Q^2, Q^6\}$, $\{Q^2, Q^5\}$, $\{Q^1, Q^4\}$

group in D. Skyline groups is a set of all l-point skyline groups. For simplicity, we use $GSkyline$ to denote skyline groups.

Consider the data set shown in Fig. 1 (left) and assume that $l = 2$, then the skyline groups based on above definitions are shown in Table 3.

We define the problem of TKD query on skyline groups in the following. We first give the definition of $score(G)$.

Definition 4. *(Score of a group G)*

$$score(G) = |\{Q \in D - G | \exists Q' \in G \wedge Q' \succ Q\}|.$$

For instance in Fig. 1, if $G = \{Q^2, Q^3\}$, then

$$score(G) = |\{Q^5, Q^6, Q^8, Q^9, Q^{10}\}| = 5.$$

Definition 5. (TKD query on skyline groups) *TKD query on skyline groups retrieves the set $S_k \subseteq GSkyline$ of k skyline groups with highest score values. Then we have*

$$\forall G \in S_k, \forall G' \in (GSkyline - S_k) \rightarrow score(G) \geq score(G').$$

Obviously, Definition 5 can be applied to find top-k dominating skyline groups based on both Definitions 1 and 2. Assume that $k = 2$, then the top-2 dominating skyline groups under different definitions are shown in Table 4. The scores of groups contained in S_k are larger than or equal to the $Threshold$ while the scores of groups outside the S_k are smaller than or equal to the $Threshold$.

Our objective is to design efficient algorithms to process TKD queries on skyline groups under different definitions.

3 Related Work

In this section, we first review related works on skyline groups queries, TKD queries, and other rank-aware queries. Then we make a comprehensive comparison between existing works and our work.

TABLE 4
Results of TKD Queries on Skyline Groups

$\succ$	$k = 2$	*Threshold*
Permutation	$\{Q^2, Q^3\}$, $\{Q^1, Q^3\}$	4
SUM	$\{Q^2, Q^3\}$, $\{Q^1, Q^3\}$	4
MAX	$\{Q^2, Q^3\}$, $\{Q^1, Q^3\}$	4
MIN	$\{Q^2, Q^3\}$, $\{Q^3, Q^8\}$	3

3.1 Skyline Groups Queries

Skyline groups queries is proposed to address the problem that traditional skyline is inadequate to answer queries that need to analyze groups of points. With regard to the concept of skyline groups queries, the most related works are [2], [3], [4], [5], [6], [13]. Im and Park [2], [3], [4], [6], [13] investigate the skyline groups query based on Definition 2. While many aggregate functions can be considered in calculating representative points, they focus on SUM, MAX, and MIN that are commonly used in database applications. However, in some cases, it is difficult to choose a good aggregate function thus the representative point may not be able to fully represent the corresponding group. To address this problem, Liu et al. [5] generalize the original skyline definition (for individual points) to permutation group-based skyline (for groups), which is Definition 1 in our paper. However, the output sizes of both Definition 1 and Definition 2 are significantly large, which is a potential limitation of the skyline group operator.

Among the aforementioned works, the most related work is [13]. In their work, skyline group is based on Definition 2. They rank skyline groups based on a predefined preferred attribute order. For instance, let $A = \{a_1, a_2, \ldots, a_d\}$ be the set of dimensions such that a_i is more preferred by the user than a_j if $i < j$. Assume that two points Q and Q' represent two groups G and G' respectively. G has a higher rank than G', if (1) $Q_{a_1} > Q'_{a_1}$ or (2) $Q_{a_i} = Q'_{a_i}$, for $i = 1, 2, \ldots, j\,(j < d)$ and $Q_{a_{j+1}} > Q'_{a_{j+1}}$. Obviously, the problem proposed in [13] is inherently different from our problem. Moreover, the ranking method proposed in [13] is not applicable to skyline groups under permutation (Definition 1), since a group cannot be represented by a point based on Definition 1. Therefore, solutions in [13] cannot be directly applied to our problem.

3.2 Top-*k* Dominating Queries

Since Papadias et al. [8] first introduced top-k dominating query based on skyline query on the traditional complete data set, many proposals of improved algorithms [9], [14], variations of TKD queries [10], [15], [16], [17], [18], [19] have been investigated. With massive uncertain and incomplete data generating from many practical applications, efficient TKD queries on uncertain and incomplete data have received considerable attentions from database community. Lian and Chen [20], [21], [22] investigate the probabilistic TKD queries on uncertain data and [23] investigates TKD queries on incomplete data.

Among the aforementioned works, the most related works to ours are [8], [10], [20], [22], [23]. In [10] they propose a top-k representative skyline points query. It aims to compute a set of k skyline points such that the total number of points dominated by one of the k skyline points is maximized. Obviously, it is inherently different from our problem. Moreover, a skyline group may consist of both

skyline points and non-skyline points. Therefore, the techniques proposed in [10] are not applicable to our problem.

In [8], [20], [22], [23], they all rank points based on the scores of points and return k points with highest scores. Obviously, they are inherently different from our problem. Because they only focus on individual points but our problem focuses on groups of points. Thus they only need to record the number of dominated points for each point and do not need to consider that a point may be dominated by many other points. Therefore, the techniques proposed in [8], [20], [22], [23] are not applicable to our problem either. Since we aim at processing TKD queries on groups of points rather than individual points, existing TKD algorithms cannot be used to our problem.

3.3 Other Rank-Aware Queries

Beside top-k dominating queries, there are many other rank-aware queries. Dong et al. [24], [25] investigate the problem of reverse k-ranks queries. Tang et al. [26] study extracting top-k insights by a score function to facilitate enterprise knowledge workers. Yang and Cai [27] propose querying improvement strategies to minimize cost and maximize hit in top-k queries. Tao et al. [28], [29], [30] investigate representative skyline. They find that top-k dominating queries typically return k points that are very similar to each other (concentrated near the origin). Therefore, they select representative skyline based on distance or diversity.

In general, the aforementioned works focus on finding a group of k points. These k points either rank in the top k or can best represent the whole data set based on predefined criteria. However, our work focus on finding k groups of points. Therefore, the techniques proposed in the aforementioned works cannot be applied to our problem. Moreover, a skyline group can contain both skyline points and non-skyline points. As a result, the points of forming skyline groups are not always very similar to each other, especially under the Definition 1 or when the aggregate function are MAX and MIN. Therefore, the limitation discussed in [28], [29], [30] do not apply to skyline groups.

To the best of our knowledge, we are the first to investigate the problem of top-k dominating queries on skyline groups.

4 TKD Query on Skyline Groups

In this section we provide a systematic study of TKD queries on skyline groups under different definitions.

4.1 Skyline Groups under Permutation

4.1.1 The Unit-Wise+ Algorithm

It is shown in [5] that if a point Q^i is in a skyline group G based on Definition 1, all points that dominate Q^i must be contained in G. Therefore, if a point Q is dominated by more than $l-1$ points, then Q will not be in any skyline groups. Based on this important conclusion, *unit-wise+* [5] first prunes the input points, then computes *units* for each point left behind. Here, a *unit* of a point contains the point and all points dominate this point. Finally, *unit-wise+* unites $units$ together to get l-point skyline groups.

Example 1. Assume that $l = 3$ and the data set is shown in Fig. 1 (left). The units of residual points are shown in Fig. 2. Then units are processed in the order of $< u_{10}, u_9, u_8, u_6, u_5, u_4, u_7, u_3, u_2, u_1 >$. For u_i and u_j, if $|u_i \cup u_j| = l$, $u_i \cup u_j$ is output as a skyline group. For instance, $u_{10} \cup u_9$ is output as a skyline groups. All skyline groups are shown in red solid boxes. If $|u_i \cup u_j| > l$, $u_i \cup u_j$ is marked as pruned. For instance, $u_{10} \cup u_6$ is marked as pruned. If $u_i \subset u_j$ or $u_j \subset u_i$, $u_i \cup u_j$ is marked as pruned. For instance, $u_{10} \cup u_3$ is marked as pruned. If $|u_i \cup u_j| < l$, $u_i \cup u_j$ is used to unite other units. For instance, $u_7 \cup u_3$ is used to unite u_2 to form a skyline group $u_7 \cup u_3 \cup u_2$. As shown in Fig. 2, unit-wise+ builds 58 groups, among which 26 groups are skyline groups.

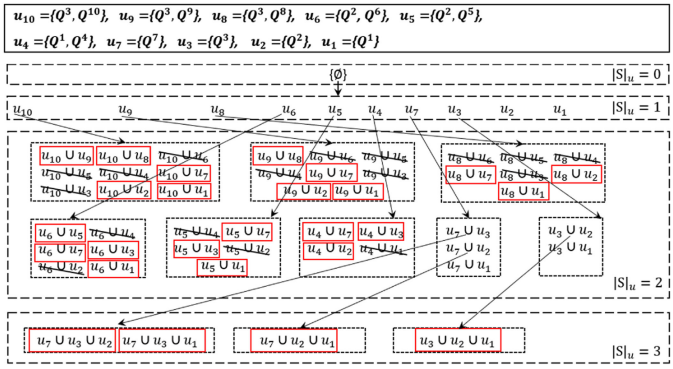


Fig. 2. An example of finding 3-point skyline groups under permutation.

4.1.2 TKD Skyline Groups under Permutation

Lemma 1. *For Definition 1, if $G \in GSkyline$ and $G = \{Q^1, Q^2, \ldots, Q^l\}$, then for each $Q^i \in G$, we have $Q^i \in Skyline$ or $\exists Q^j \in G$ such that $Q^j \in Skyline$ and $Q^j \succ Q^i$.*

Proof. We prove the lemma by contradiction. Assume that $Q^j \succ Q^i$ *and* $Q^j \in Skyline$, if $Q^j \notin G$, we can use Q^j to replace Q^i in G, the new group is denoted as G'. For Definition 1, since $Q^j \succ Q^i$ and all the other points are the same, then $G' \succ_p G$ which contradicts $G \in GSkyline$. □

Algorithm 1. *TKD-Permutation (TKDP)*

Input: D k, l;
Output: S_k
1: $Residual \leftarrow Input\ prunning(D)$
2: $Residual \leftarrow Residual - Skyline$
3: $PQ \leftarrow \emptyset$; /*PQ *is a priority queue sorting groups in the ascending order of their scores.* */
4: $candidate \leftarrow \emptyset$;/**initialize can as a group containing zero point.* */
5: $Generate\ Groups(0, l, Skyline, candidate)$
6: **for** each group G in PQ **do**
7: **if** $|G| < l$ **then**
8: $PQ \leftarrow PQ - \{G\}$
9: Add G in the group vector $candidate$ for reviving
10: **for** each group G in $candidate$ **do**
11: **if** $|PQ| < k$ or $Score(G) > Score(PQ.top())$ **then**
12: $Revive(0, l - |G|, Residual, G)$
13: $S_k \leftarrow PQ$

From the Lemma 1, we know that non-skyline points make no contribution to the scores of skyline groups. Therefore, top-k dominating skyline groups should contain as many skyline

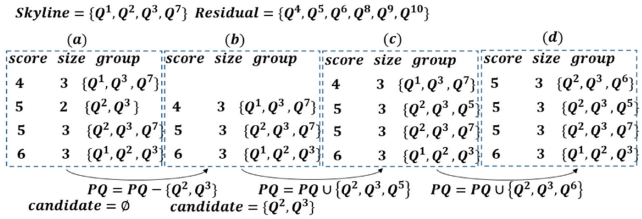


Fig. 3. Graphic presentation of Algorithm 1.

points as possible. Based on this intuition, we propose Algorithm 1 to compute top-k dominating skyline groups without materialising all skyline groups. We first employ Algorithm 2 to generate all groups consisting of skyline points. The size of groups is between $[1, l]$. PQ is used to store k groups with the highest scores. $Skyline^i$ in Algorithm 2 denotes the ith point in the $Skyline$. Since some groups in PQ consists of less than l points, we employ Algorithm 3 to compute l-point skyline groups based on these groups. $Residual^i$ in Algorithm 3 denotes the ith point in $Residual$.

Algorithm 2. *Generate Groups*

Input: Pos, $length$, $Skyline$, $candidate$;
Output: PQ
/* *initialize candidate as a group containing zero point.* */
if *length = 0* **then**
  return
else
  for $i \leftarrow pos; i < |Skyline|; i++$ **do**
    $temp \leftarrow candidate$; /* *temp is a group.* */
    $temp \leftarrow temp \cup \{Skyline^i\}$
    if $|PQ| < k$ **then**
      $PQ.push(temp)$
    else
      if $Score(temp) > Score(PQ.top())$ **then**
        $PQ.pop()$
        $PQ.push(temp)$
    $Generate\ Groups(i+1, length-1, Skyline, temp)$

Example 2. Assume that $k = 4, l = 3$, then the output of Algorithm 2 is in Fig. 3a. Since the size of group $\{Q^2, Q^3\}$ is 2 smaller than l, we need to compute l-point groups based on this group. We remove $\{Q^2, Q^3\}$ from PQ, then add it into $candidate$, which is shown in Fig. 3b. Because we have built all groups consisting with skyline points in Algorithm 2, we use non-skyline points after input pruning to build l-point groups in Algorithm 3. In Algorithm 3, Q^4 is skipped because $Q^1 \succ Q^4$ but $Q^1 \notin \{Q^2, Q^3\}$. Then we add $\{Q^2, Q^3, Q^5\}$ and $\{Q^2, Q^3, Q^6\}$ in PQ as shown in Fig. 3c and 3d respectively. Since the scores of $\{Q^2, Q^3, Q^8\}, \{Q^2, Q^3, Q^9\}$ and $\{Q^2, Q^3, Q^{10}\}$ are no larger than the score of the first group in PQ, PQ remains the same. In Algorithm 2, we build $C^1_{|Skyline|} + C^2_{|Skyline|} + C^3_{|Skyline|} = 14$ groups and in Algorithm 3, we build 6 groups. Thus we build 20 groups, and among which 8 groups are 3-point skyline groups.

4.1.3 Time Complexity Analysis

From Examples 1 and 2 we can see that Algorithm 1 builds much less groups than *unit-wise+*. Moreover, after materialising only 8 skyline points, Algorithm 1 returns top-4 dominating skyline groups. Let $|U|$ denote the number of units used in *unit-wise+*, then $|U| = |Skyline| + |Residual|$. For each new built group in *unit-wise+*, we need to check whether it is a skyline group. Therefore, the time complexity of *unit-wise+* is $C^1_{|U|} + C^2_{|U|} + \cdots + C^l_{|U|} = \sum_{i=1}^{l} C^i_{|U|}$ in the worst case. The time complexity of Algorithm 2 is $\sum_{i=1}^{l} C^i_{|Skyline|}$ and we need to build at most $\sum_{i=1}^{l} C^i_{|Residual|}$ groups in Algorithm 3, thus the time complexity of Algorithm 1 is $\sum_{i=1}^{l} C^i_{|Skyline|} + \sum_{i=1}^{l} C^i_{|Residual|}$ in the worst case. Therefore, the time complexity of Algorithm 1 is much less than that of *unit-wise+*.

Algorithm 3. *Revive*

Input: Pos, $length$, $Residual$, $candidate$;
Output: PQ
if *length = 0* **then**
  if $|PQ| < k$ **then**
    $PQ.push(candidate)$
  else if $Score(candidate) > Score(PQ.top())$ **then**
    $PQ.pop()$
    $PQ.push(candidate)$
else
  for $i \leftarrow pos; i < |Residual|; i++$ **do**
    $temp \leftarrow candidate$; /* *temp is a group.* */
    if *temp contains all points that dominate* $Residual^i$ **then**
      $temp \leftarrow temp \cup \{Residual^i\}$
      $Revive(i+1, length-1, Residual, temp)$

4.2 Skyline Groups under SUM

4.2.1 The Dynamic Programming Algorithm for SUM

In order to compute skyline groups based on SUM, existing works [2], [3], [4], [6] adopt similar dynamic programming algorithms. We briefly denote these dynamic programming algorithms as *DPSUM*. Let Sky^n_l denote the set of l-point skyline groups with regard to $\{Q^1, \ldots, Q^n\}$ and Sky^{n-1}_{l-1} denote the set of $l-1$ point skyline groups with regard to $\{Q^1, \ldots, Q^{n-1}\}$. Thus we have:

Lemma 2. *Under SUM, given* $G \in Sky^n_l$, *if* $Q^n \in G$, *then* $G \backslash \{Q^n\} \in Sky^{n-1}_{l-1}$.

Based on Lemma 2, Sky^n_l is computed as follows:

$$Sky^n_l = skyline(Sky^{n-1}_l + \{G \cup \{Q^n\} | G \in Sky^{n-1}_{l-1}\}). \quad (1)$$

As shown in [2], [3], [4], [6] that if a point Q is dominated by more than $l-1$ points, then Q will not be in any skyline groups. Therefore, the input pruning used in *DPSUM* is the same as that used in *unit-wise+*.

Example 3. Fig. 4 is used to present the process of finding skyline groups using *DPSUM*. For instance, before Q^4 is added, $Sky^3_2 = \{\{Q^1, Q^2\}, \{Q^1, Q^3\}, \{Q^2, Q^3\}\}$. Then Sky^4_2 is initialized as Sky^3_2. Next, for each group $G \in Sky^3_1$, we add $G \cup \{Q^4\}$ into Sky^4_2. Thus $Sky^4_2 = \{\{Q^1, Q^2\}, \{Q^1, Q^3\}, \{Q^2, Q^3\}, \{Q^1, Q^4\}, \{Q^2, Q^4\}, \{Q^3, Q^4\}\}$. Then we compute the skyline of Sky^4_2, thus $Sky^4_2 = \{\{Q^1, Q^2\}, \{Q^1, Q^3\},$

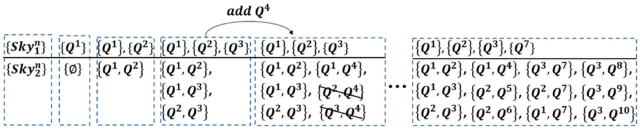


Fig. 4. The example of finding 2-point skyline groups using *DPSUM*.

$\{Q^2,Q^3\},\{Q^1,Q^4\}\}$. After all points are processed, Sky_2^{10} contains 12 2-point skyline groups as shown in Fig. 4.

4.2.2 TKD Skyline Groups under SUM

Algorithm 4 is proposed to compute top-k dominating skyline groups under SUM. Assume that point Q presents group G. Let $L_1(G)=\sum_{j=1}^{d} Q_j$. The intuition of Algorithm 4 is that a group G with large $L_1(G)$ tends to have high $Score(G)$. In Algorithm 4 we first prune the input then sort the residual points in the descending order of L_1 norm. Q^j denotes the jth point in D after sorting. Let T denote the set of points in D excluding the first j points, then $T=\{Q^{j+1},Q^{j+2},\ldots,Q^{|D|}\}$.

Definition 6. *If $|Sky_l^j|\geq k$, $PQ(Sky_l^j)$ denotes the set of k skyline groups with highest scores. $Scoremin$ denotes the minimum $Score(G)$ of all groups in PQ and L_1min denotes minimum $L_1(G)$ for all groups in PQ.*

As shown in Fig. 4, if $k=2$, then $PQ(Sky_2^4)=\{\{Q^1,Q^3\},\{Q^2,Q^3\}\}$. Thus $Scoremin=Score(\{Q^1,Q^3\})=4$, $L_1min=L_1(\{Q^2,Q^3\})=17$.

Definition 7. *(Descendants of Sky_i^j) If a group $G\in Sky_i^j(0<i<l)$ and B is a set consisting of any $l-i$ points in T, $Q\cup B$ is a descendant of G. We define descendants of Sky_i^j as $\bigcup_{G\in Sky_i^j}\{G\cup B|B\in C_T^{l-i}\}$. If $i=0$, $Sky_0^j=\{\emptyset\}$ and the descendants of Sky_0^j are C_T^l ($T=\{Q^{j+1},Q^{j+2},\ldots,Q^{|D|}\}$).*

Lemma 3. *For each group $G\in Sky_i^j(0\leq i<l)$, if $Score(G\cup T)<=Scoremin$ and $L_1(G)+\sum_{u=1}^{l-i}L_1(Q^{j+u})\leq L_1min$, then the descendants of Sky_i^j have no influence on the final result of TKD queries on skyline groups.*

Proof. If G' is a descendant of $G(G\in Sky_i^j)$, obviously $Score(G')\leq Score(G\cup T)\leq Scoremin$. Thus $G'\notin S_k$ because we have already got k skyline groups whose scores are no smaller than $Score(G')$. Moreover, since $L_1(G')\leq L_1(G)+\sum_{u=1}^{l-i}L_1(Q^{j+u})\leq L_1min$, G' cannot dominate any groups in $PQ(Sky_l^j)$. Therefore, the descendants of G' have no influence on the final result. □

Lemma 4. *For each $Sky_i^j(0\leq i<l)$, if the descendants of Sky_i^j have no influence on the final result. then $PQ(Sky_l^j)$ is the result of TKD queries on skyline groups under SUM.*

It is a direct conclusion from Lemma 3.

Example 4. In Fig. 4, before Q^4 is added, $Sky_0^3=\{\emptyset\}$, $Sky_1^3=\{\{Q^1\},\{Q^2\},\{Q^3\}\}$ and $T=\{Q^4,Q^5,\ldots,Q^{10}\}$. Since $PQ(Sky_2^3)=\{\{Q^1,Q^3\},\{Q^2,Q^3\}\}$, then $Scoremin=4$, $L_1min=17$. For Sky_0^3, since $Score(T)=0<Scoremin$ and $L_1(Q^4)+L_1(Q^5)=13<L_1min$, then the descendants of Sky_0^3 have on influence on the final result. For $\{Q^1\}$ in Sky_1^3, since $Score(\{Q^1\}\cup T)=1<Scoremin$ and $L_1(Q^1)+L_1(Q^4)=17\leq L_1min$, then the descendants of $\{Q^1\}$ have on influence on the final result. We find that the descendants of $\{Q^2\}$ and $\{Q^3\}$ have no influence on the final result in the same way. Therefore, descendants of Sky_1^3 have on influence on the final result.

Based on Lemma 4, $\{\{Q^1,Q^3\},\{Q^2,Q^3\}\}$ is the top-2 dominating skyline groups. In our method, after processing only 3 points we can return the result, which is far more efficient than the *DPSUM*.

Algorithm 4. *TKD-SUM (TKDS)*

Input: D k, l;
Output: S_k
1: $D\leftarrow Input\ prunning(D)$
2: Modify all points in D using Equation (2)
3: Sort points in D in the descending order of L_1 norm
4: $PQ\leftarrow\emptyset$; /* *PQ is a priority queue sorting groups in the ascending order of their scores.* */
5: **for** $j\leftarrow 1; j\leq|D|; j++$ **do**
6: **for** $i\leftarrow min(j,l); i\geq 1; i--$ **do**
7: **if** $i=1$ **then**
8: $Sky_1^j=skyline(Sky_1^{j-1}\cup\{D^j\})$
9: **else**
10: $Sky_i^j=skyline(Sky_i^{j-1}+\{G\cup\{D^j\}|G\in Sky_{i-1}^{j-1}\})$
11: **if** $|Sky_l^j|\geq k$ **then**
12: **if** *the descendants of* $\forall Sky_i^j$ $(0\leq i<l)$ *have no influence on the final result* **then**
13: $S_k\leftarrow PQ(Sky_l^j)$
14: *return* S_k
15: $S_k\leftarrow PQ(Sky_l^{|D|})$

We propose some techniques to improve the pruning power of Lemma 3. In some cases, the values of one dimension are significantly larger than the values of other dimensions, which slows down the performance Algorithm 4. Let d_jmax denote the largest value of the jth dimension. For each point Q, we modify all dimensions of Q in the following way.

$$Q_j=Q_j\div d_jmax\quad(1\leq j\leq d). \tag{2}$$

In Lemma 3, the conditions that the descendants of Sky_{l-1}^j have no influence on the final result are too loose. We tighten the conditions in the following.

Lemma 5. *For each $G\in Sky_{l-1}^j$ and for each point $Q\in T$, if $Score(G\cup\{Q\})\leq Scoremin$ and $L_1(G\cup\{Q\})\leq L_1min$, then the descendants of Sky_{l-1}^j have no influence on the final result.*

In order to avoid double counting problem in applying Lemma 4, we mark $G\in Sky_i^j(0\leq i<l)$ as pruned if all descendants of G have no influence on the final result. Therefore, when we check descendants of $Sky_i^j(0\leq i<l)$, we only need to check groups $G\in Sky_i^j$ that are not marked as pruned. This saves considerable computation effort.

In Algorithm 4, points are processed in the descending order of L_1 norm. j denotes the position of points in D. For instance, D^j denotes the jth point in D. i denotes the size of a group. For instance, Sky_i^j denotes the set of i-point skyline groups on the first j points. When Sky_l^j is computed ($line$ 5-$line$ 10), we utilize Lemma 4 ($Line$ 12) to check whether Algorithm 4 can terminate.

4.2.3 Time Complexity Analysis

DPSUM needs to process all residual points after input pruning to get all skyline groups while our algorithm can return S_k before materialising all skyline groups, which significantly improves the performance of the TKD queries on skyline groups under SUM. Assume that there are n points left in D after pruning, thus for each j ($j \leq n$), there are l sub problems which are $\{Sky_1^j, Sky_2^j, \ldots, Sky_l^j\}$. Totally, there are $n \times l$ sub problems. For each sub problem Sky_i^j, there are at most C_j^i groups, thus the time complexity of finding skyline on these groups is $O(C_j^i)^2$. Therefore, the time complexity of *DPSUM* in the worst case is $O(n \times l \times (C_n^l)^2)$.

Assume that Algorithm 4 terminates at the jth point, then there are $j \times l$ sub problems. For each sub problem $Sky_i^{j'}$ ($j' \leq j$), if $i = l$, we need to check whether Lemma 4 holds. The time complexity of checking Lemma 4 ($Line$ 12) is $O(|Sky_1^j| + |Sky_2^j| + \cdots + |Sky_{l-1}^j|)$, which is less than $O(l \times C_j^l)$. Since we need to check Lemma 4 at most j times, the time complexity of using Lemma 4 is $O(j \times l \times C_j^l)$. Therefore, the time complexity of Algorithm 4 is $O(j \times l \times (C_j^l)^2) + O(j \times l \times C_j^l) = O(j \times l \times (C_j^l)^2)$. Obviously, the time complexity decreases dramatically with decreasing j. Therefore, Algorithm 4 is much more efficient than the *DPSUM*.

Algorithm 5. *TKD-MAX (TKDMAX)*

Input: D k, l;
Output: S_k
1: $Skyline \leftarrow skyline(D)$
2: **if** $|Skyline| \leq l$ **then**
3: $G \leftarrow \{Skyline\}$
4: **for** *each* $(l - |G|)$*-point combination* G' *among the points in* $D - Skyline$ **do**
5: $S_k \leftarrow S_k \cup \{G \cup G'\}$
6: **if** $|S_k| = k$ return S_k
7: $PQ \leftarrow \emptyset$; /* *PQ is a priority queue sorting groups in the ascending order of their scores.* */
8: **for** $j \leftarrow 1$; $j \leq |Skyline|$; $j++$ **do**
9: **for** $i \leftarrow min(j, l)$;$i \geq 1$;$i--$ **do**
10: **if** $i = 1$ **then**
11: $Sky_1^j = skyline(Sky_1^{j-1} \cup \{Skyline^j\})$
12: **else**
13: $Sky_i^j = skyline(Sky_i^{j-1} + \{G \cup \{Skyline^j\} | G \in Sky_{i-1}^{j-1}\})$
14: Sort groups in $Sky_l^{|Skyline|}$ in the descending order of $Score(G)$
15: For groups sharing the same skyline vector, we keep the one with the highest score and add this group into $Queue$
16: **for** *each group* G *in* $Queue$ **do**
17: **if** $|PQ| < k$ or $Score(G) > Score(PQ.top())$ **then**
18: *Construct Groups* (G)
19: **else**
20: break
21: $S_k \leftarrow PQ$

4.3 Skyline Groups under MAX

4.3.1 The Dynamic Programming Algorithm for MAX

Many skyline groups under MAX share the same aggregate vector. For instance, if Q dominates all points in D, Q and any $l - 1$ points can form a l-point skyline group and all the skyline groups share the same aggregate vector. An aggregate vector is denoted as $(v[1], v[2], \ldots, v[d])$. In the above example, we have $v[j] = Q_j$ $(1 \leq j \leq d)$. We denote skyline vector as the aggregate vector of a skyline group.

In [6], they propose two dynamic programming algorithms to compute skyline vectors. The first algorithm is based on Equation (1) and the second algorithm is based on the following equation.

$$Sky_l = skyline(\{G \cup \{Q^i\} | G \in Sky_{l-1} \text{ and } Q^i \notin G\}). \tag{3}$$

The experimental results in [6] show that the first algorithm outperforms the second algorithm in most cases. Therefore, we adopt the first algorithm to compute skyline vectors. After computing the skyline vectors, we construct skyline groups sharing the identical skyline vectors. As shown in [6], it is an NP-hard problem. The NP-hardness directly follows from that of *SET-COVER*. The main idea is that for each skyline vector v, we find groups G ($|G| \leq l$) achieving v. If $|G| < l$, then G and any $l - |G|$ points in $D - G$ form the skyline groups achieving v.

Example 5. Assume that $l = 2$ and the data set is shown in Fig. 1 (left). After computing skyline vectors based on Equation (1), there are 6 skyline vectors as shown in Fig. 5. Then we need to construct skyline groups based on these skyline vectors. We define $V_j = \{Q | Q_j = v[j]\}$. For instance, if $v = (2, 7, 3)$, then $V_1 = \{Q^3\}$, $V_2 = \{Q^2\}$ and $V_3 = \{Q^3, Q^9, Q^{10}\}$. Therefore, there are 3 combinations of selecting points from V_1, V_2 and V_3. They are $\{Q^3, Q^2, Q^3\}$, $\{Q^3, Q^2, Q^9\}$ and $\{Q^3, Q^2, Q^{10}\}$. Since the sizes of $\{Q^3, Q^2, Q^9\}$ and $\{Q^3, Q^2, Q^{10}\}$ are larger than l, we prune these 2 groups. For $\{Q^3, Q^2, Q^3\}$, since its size is equal to l, we do not need to combine other points with it to form a skyline group. Actually, it is a skyline group. After constructing skyline groups for each skyline vector, we get 6 skyline groups as shown in Fig. 5.

Algorithm 6. *Construct Groups (MAX function)*

Input: G
Output: PQ
1: v is the skyline vector of G
2: Compute $\{V_1, \ldots, V_d\}$ $(V_j = \{Q' | Q'_j = v[j]\})$
3: **for** *each combination* G^c $(G^c = \{Q^1, Q^2, \ldots, Q^d\},\ Q^j \in V_j)$ **do**
4: Remove duplicate points in G^c
5: **if** $|G^c| \leq l$ **then**
6: **for** *each* $(l - |G^c|)$*-point combination* G' on $D - G^c$ **do**
7: **if** $|PQ| < k$ *and* $G^c \cup G' \notin PQ$ **then**
8: $PQ.push(G^c \cup G')$
9: **else if** $Score(G^c \cup G') > Score(PQ.top())$ *and* $G^c \cup G' \notin PQ$ **then**
10: $PQ.pop()$
11: $PQ.push(G^c \cup G')$
12: return PQ

4.3.2 TKD Skyline Groups under MAX

As shown in [6], it is safe to exclude any non-skyline points from the input without influencing the skyline vectors.

skyline vector	(2,7,3)	(10,3,3)	(10,7,3)	(2,3,4)	(1,7,4)	(10,0,4)
skyline group	$\{Q^2, Q^3\}$	$\{Q^1, Q^3\}$	$\{Q^1, Q^2\}$	$\{Q^3, Q^7\}$	$\{Q^2, Q^7\}$	$\{Q^1, Q^7\}$
score	5	4	4	3	2	1
order	1	2	3	4	5	6

Fig. 5. The example of finding 2-point skyline groups under MAX.

Queue	$\{Q^2, Q^3\}$	$\{Q^1, Q^3\}$	$\{Q^1, Q^2\}$	$\{Q^3, Q^7\}$	$\{Q^2, Q^7\}$	$\{Q^1, Q^7\}$
score	5	4	4	3	2	1

PQ: 5 $\{Q^2, Q^3\}$ (*score group*); 4 $\{Q^1, Q^3\}$ / 5 $\{Q^2, Q^3\}$ (*score group*)

Fig. 6. Graphic presentation of Algorithm 5.

Therefore, in Algorithm 5, we use the $Skyline$ to compute the skyline groups ($Line$ 8). The aggregate vectors of skyline groups in $Sky_l^{|Skyline|}$ cover all distinct skyline vectors. Then we sort groups G in $Sky_l^{|Skyline|}$ in the descending order of $Score(G)$, which is shown in Fig. 5. Next, among all skyline groups achieving v, we keep the one with the highest score and add it into $Queue$ ($Line$ 15). Thus we have no duplicate skyline vectors in $Queue$.

Lemma 6. *If $|Skyline| \geq l$ and $Scoremax$ is the highest score of all groups sharing an identical skyline vector, then there must exist a group G satisfying $Score(G) = Scoremax$ and consisting of skyline points .*

Proof. Assume that a skyline group G contains a non-skyline point Q that is dominated by a skyline point Q'. If $Q' \notin G$, then we can replace Q with Q' to form a group G'. Obviously, $G' \succeq_g G$, since G is a skyline group, then G and G' share the same skyline vector. In this case, $Score(G') \geq Score(G)$. If $Q' \in G$, then we can replace Q with any skyline point outside G to form a group G'. Thus $G' \succeq_g G$, since G is a skyline group, then G and G' share the same skyline vector. In this case, $Score(G') \geq Score(G)$. Therefore, Lemma 6 is proved. □

Based on Lemma 6, we know that for each skyline vector v, if $G \in Queue$ and G achieves v, then G has the highest score among all skyline groups sharing v. For each group G in $Queue$, we adopt Algorithm 6 to construct skyline groups sharing the same skyline vector. In Algorithm 6, the number of combination G^c is $|V_1| \times |V_2| \times \cdots \times |V_d|$. Then we build skyline groups based on each G^c.

Example 6. Assume that $k = 2$ and $l = 2$, $Queue$ is shown in Fig. 6. After processing the second group in $Queue$, $|PQ| = 2$ and $Score(PQ.top()) = 4$. When processing the third group $\{Q^1, Q^2\}$, since $Score(\{Q^1, Q^2\}) \leq 4$, we skip constructing skyline groups based on $\{Q^1, Q^2\}$. Because the skyline groups achieving the aggregate vector of $\{Q^1, Q^2\}$ cannot have higher scores than scores of groups in PQ. Moreover, since the scores of the groups behind $\{Q^1, Q^2\}$ in $Queue$ are no higher than that of $\{Q^1, Q^2\}$, our algorithm terminates after processing the third group in $Queue$.

Unlike the dynamic programming algorithm which needs to construct all groups for all distinct skyline vectors, our algorithm can return top-k dominating skyline groups without materialising all skyline groups. As shown in Fig. 6, after processing only 3 groups in $Queue$, we get the final result.

skyline vector	(1, 3, 1)	(1, 0, 3)	(1, 1, 2)	(0, 1, 3)	(1, 5, 0)	(0, 4, 1)	(7, 0, 0)
skyline group	$\{Q^2, Q^3\}$	$\{Q^3, Q^{10}\}$	$\{Q^3, Q^8\}$	$\{Q^3, Q^9\}$	$\{Q^2, Q^5\}$	$\{Q^2, Q^6\}$	$\{Q^1, Q^4\}$
score	5	3	3	3	2	2	1
order	1	2	3	4	5	6	7

Fig. 7. The example of finding 2-point skyline groups under MIN.

4.3.3 Time Complexity Analysis

The time complexity of computing skyline vectors in the worst case is $O(|Skyline| \times l \times (C^l_{|Skyline|})^2)$. There are at most $C^l_{|Skyline|}$ skyline vectors thus the time complexity of computing $Queue$ is $O((C^l_{|Skyline|})^2)$.

If all points in D are identical, then we have $|V_j| = n$ $(1 \leq j \leq d)$. In this case, there are n^d combinations of G^c. For each G^c, we need to build at most C^{l-1}_{n-1} skyline groups. Therefore, the time complexity of Algorithm 6 is $O(n^d \times C^l_n)$.

Because there are at most $C^l_{|Skyline|}$ groups in $Queue$, then the time complexity of finding all skyline groups with identical skyline vectors is $O(n^d \times C^l_n \times C^l_{|Skyline|})$. Since $n^d > |Skyline| \times l$ and $C^l_n > C^l_{|Skyline|}$, the time complexity of computing skyline groups under MAX is $O(n^d \times C^l_n \times C^l_{|Skyline|})$.

The above analysis and the NP-hardness of this problem indicate that it is very expensive to find all skyline groups achieving the same skyline vector. As shown in Example 6, our algorithm can return top-k dominating skyline groups without processing all groups in $Queue$. Therefore, our algorithm significantly improves the performance of TKD on skyline groups under MAX.

4.4 Skyline Groups under MIN

4.4.1 The Dynamic Programming Algorithm for MIN

Similar to MAX, two dynamic programming algorithms are proposed to compute skyline groups under MIN based on Equations (1) and (2) respectively. We adopt the one based on Equation (1) for its better performance.

For a skyline vector v under MIN, the first step of constructing skyline groups based on v is to find a set $\Theta(v)$ with points that dominate or equal to v. Then we have $\Theta(v) = \{Q | Q \succeq v\}$. For any l-point combination on $\Theta(v)$, its aggregate vector v' must be equal to v. Because v is skyline vector, then $v' \not\succ v$. Thus we have $v' = v$. Moreover, if a group G contains a point outside $\Theta(v)$, then the aggregate vector of G must have smaller values than v on some dimensions, thus G cannot achieve v. Therefore, for each v, the skyline groups sharing v are the set of any l-point combinations on $\Theta(v)$.

Example 7. Assume that $l = 2$ and the data set is shown in Fig. 1 (left). After computing skyline vectors based on Equation (1), there are 7 skyline vectors as shown in Fig. 7. We need to construct skyline groups based on these vectors. For instance, if $v = (1, 3, 1)$, then $\Theta(v) = \{Q^2, Q^3\}$. There is only one 2-point combination on $\Theta(v)$, thus there is only one skyline groups achieving v.

After processing all skyline vectors, we get 7 skyline groups as shown in Fig. 7.

Algorithm 7. *TKD-MIN (TKDMIN)*

Input: D k, l;
Output: S_k
1: $T \leftarrow Input\ prunning(D)$
2: $PQ \leftarrow \emptyset$; /* *PQ is a priority queue sorting groups in the ascending order of their scores.* */
3: **for** $j \leftarrow 1; j \leq |T|; j++$ **do**
4: **for** $i \leftarrow min(j, l); i \geq 1; i--$ **do**
5: **if** $i = 1$ **then**
6: $Sky_1^j = skyline(Sky_1^{j-1} \cup \{T^j\})$
7: **else**
8: $Sky_i^j = skyline(Sky_i^{j-1} + \{G \cup \{T^j\} | G \in Sky_{i-1}^{j-1}\})$
9: Remove duplicate skyline vectors in $Sky_l^{|T|}$
10: **for** *each* v *in* $Sky_l^{|T|}$ **do**
11: Compute $\Theta(v)$
12: **if** $|PQ| < k$ or $Score(\Theta(v)) > Score(PQ.top())$ **then**
13: **for** *each l-point combination* G on $\Theta(v)$ **do**
14: **if** $|PQ| < k$ *and* $G \notin PQ$ **then**
15: $PQ.push(G)$
16: **else if** $Score(G) > Score(PQ.top())$ *and* $G \notin PQ$ **then**
17: $PQ.pop()$
18: $PQ.push(G)$
19: $S_k \leftarrow PQ$

4.4.2 TKD Skyline Groups under MIN

Let T denote the set of points that dominated by at most $l-1$ points. As shown in [6], we can compute all distinct skyline vectors under MIN using points in T. In Algorithm 7, we first prune the input ($Line$ 1). Then we compute skyline groups ($Line$ 3-8) and remove duplicate skyline vectors ($Line$ 9). For each skyline vector v left in $Sky_l^{|T|}$, we compute $\Theta(v)$. If $|PQ| = k$ and $Score(\Theta(v)) \leq Score(PQ.top())$, we skip constructing skyline groups for v. Because the score of a group G consisting of any l points on $\Theta(v)$ is no higher than the scores of groups in PQ. Therefore, Algorithm 7 can return top-k dominating skyline groups without materialising all skyline groups.

Example 8. Assume that $k = 2$, Fig. 8 shows the result after processing the second skyline vector. We have $|PQ| = k$ and $Score(PQ.top()) = 3$. When processing the third skyline vector $v = (1, 2, 2)$, since $\Theta(v) = \{Q^3, Q^8\}$, we have $Score(\Theta(v)) \leq 3$. Then we skip constructing skyline groups for v. Moreover, for each skyline vector v' behind v, since $Score(\Theta(v')) \leq 3$, we skip constructing skyline groups for v'. Totally, for only 2 out of 7 skyline vectors, we need to construct skyline groups. Therefore, our algorithm can return top-k dominating skyline groups without materialising all skyline groups.

4.4.3 Time Complexity Analysis

The time complexity of computing skyline vectors in the worst case is $O(|T| \times l \times (C_{|T|}^l)^2)$. For each skyline vector v, the time complexity of constructing skyline groups for v in the worst case is $O(C_n^l)$. Since there are at most $C_{|T|}^l$ skyline vectors, the time complexity of constructing skyline groups for all skyline vectors is $O(C_n^l \times C_{|T|}^l)$. Therefore, time complexity of computing skyline groups under MIN is $O(|T| \times l \times (C_{|T|}^l)^2 + C_n^l \times C_{|T|}^l)$, in which $O(C_n^l \times C_{|T|}^l)$ takes a large part. As shown in Fig. 8, since our algorithm needs to constructing skyline groups for only a few skyline vectors, our algorithm can largely improve the performance of TKD queries on skyline groups under MIN.

skyline vector	$(1,3,1)$	$(1,0,3)$	$(1,1,2)$	$(0,1,3)$	$(1,5,0)$	$(0,4,1)$	$(7,0,0)$
skyline group	$\{Q^2,Q^3\}$	$\{Q^3,Q^{10}\}$	$\{Q^3,Q^8\}$	$\{Q^3,Q^9\}$	$\{Q^2,Q^5\}$	$\{Q^2,Q^6\}$	$\{Q^1,Q^4\}$
score	5	3	3	3	2	2	1

PQ	5	$\{Q^2,Q^3\}$	3	$\{Q^3,Q^{10}\}$
	score	*group*	*score*	*group*

Fig. 8. Graphic presentation of Algorithm 7.

5 Bitmap Index Method

Although our algorithms can significantly improve the performance of TKD queries on skyline groups under all existing skyline group definitions, we notice that the cost of score computation is too expensive. Because we need to check points dominated by a point in skyline groups multiple times. This will lead to heavy double counting problems. Therefore, in this section we propose a bitmap index method to compute the scores of skyline groups.

Let $dom(Q)$ denote the set of points dominated by point Q, then $score(G) = |\bigcup_{Q \in G} dom(Q)|$. In order to compute $score(G)$ efficiently, we maintain a bit vector for each point in the group, then we can employ fast bit-wise operations for much more efficient score computation.

Definition 8. ***([Q])*** *$[Q]$ denotes the bit vector of Q. $[Q]$ has the length of $|D|$ bits, with one bit corresponding to a point in D. If a point Q^j is dominated by Q, then the jth bit is set to 1. Otherwise, the bit is set to 0.*

Based on Definition 8, $score(G)$ equals the number of "1" in $[Q^1]|[Q^2]|...|[Q^l]$ ($G = \{Q^1, Q^2, \ldots, Q^l\}$). For instance, $[Q^2] = 0000110000$, $[Q^3] = 0000000111$. If $G = \{Q^2, Q^3\}$, $score(G)$ equals the number of "1" in $[Q^2]|[Q^3] = 0000110111$. Therefore, $score(G) = 5$.

5.1 Scores of Groups under Permutation and SUM

In Lemma 1, we prove that for each point $Q^i \in G$ (G is a skyline group under permutation), we have $Q^i \in Skyline$ or $\exists Q^j \in G$ such that $Q^j \in skyline$ and $Q^j \succ Q^i$. We prove that this lemma is also true for skyline groups under strictly monotone aggregate functions, such as SUM.

Lemma 7. *For strictly monotone aggregate functions under Definition 2, if $G \in GSkyline$ and $G = \{Q^1, Q^2, \ldots, Q^l\}$, then for each $Q^i \in G$, we have $Q^i \in Skyline$ or $\exists Q^j \in G$ such that $Q^j \in skyline$ and $Q^j \succ Q^i$.*

Proof. We prove by contradiction. Assume that $Q^j \succ Q^i$ and $Q^j \in Skyline$, if $Q^j \notin G$, we can use Q^j to replace Q^i in G, the new group is denoted as G'. Since $Q^j \succ Q^i$, we assume that $Q_t^j > Q_t^i$, then we have $f(Q_t^1, \ldots, Q_t^i, \ldots, Q_t^l) < f(Q_t^1, \ldots, Q_t^j, \ldots, Q_t^l)$. On other dimensions, we have $f(Q_{t'}^1, \ldots, Q_{t'}^i, \ldots, Q_{t'}^l) \leq f(Q_{t'}^1, \ldots, Q_{t'}^j, \ldots, Q_{t'}^l)$. Therefore, $G' \succ_f G$, which contradicts $G \in GSkyline$. Therefore, for strictly monotone aggregate

functions under Definition 2, if $G \in GSkyline$, then for each $Q^i \in G$, we can get that $Q^i \in Skyline$ or $\exists Q^j \in G$ such that $Q^j \in skyline$ and $Q^j \succ Q^i$. □

Based on Lemmas 1 and 7, we do not need to compute $[Q]$ for each point Q in D, instead we only need to compute $[Q]$ for each Q in the $Skyline$.

Lemma 8. *For Definition 1 and strictly monotone aggregate functions under Definition 2, we know that if $Q \in G$ and $Q \notin Skyline$, then Q has zero contribution to $score(G)$. Thus $[Q]$ is modified as follows:*

$$[Q] = \begin{cases} [Q], Q \in Skyline \\ 0...0, \quad Others \end{cases}.$$

5.2 Scores of Groups under MAX and MIN

We find that for MAX and MIN, we also do not need to compute $[Q]$ for each point Q in D.

Lemma 9. *For MAX and MIN under Definition 2, if $\exists Q^i \in G$ ($G \in GSkyline$) and Q^i is dominated by at least k points, then either (1) $\exists Q^j \in G$ and $Q^j \succ Q^i$ or (2) it is safe to exclude G from S_k.*

Proof. Obviously, it is possible to have a point $Q^j \in G$ *and* $Q^j \succ Q^i$. In this situation we have $score(G) = score(G \setminus \{Q^i\})$.

In the second situation, all points dominate Q^i are not in G. If $Q^j \succ Q^i$, we use Q^j to replace Q^i in G, the new group is denoted as G'. Since $Q^j \succ Q^i$ and all the other points are the same, then $G' \succeq_g G$ under MAX and MIN. As G is a skyline group, we have that G' and G share the same skyline vector, which means that G' is also a skyline group under MAX and MIN. Moreover, we have $socre(G') \geq score(G)$. Since Q^i is dominated by at least k points, we have at least k skyline groups whose scores are equal to or greater than $score(G)$. Therefore, it is safe to exclude G from S_k. □

We use *(k-1)-skyband* [8] to denote the set of points that are dominated by at most $k-1$ points in a data set. Based on Lemma 9, we only need to compute bit vectors for points in the *(k-1)-skyband*.

Lemma 10. *For MAX and MIN, because if $Q \notin$ (k-1)-skyband then either Q has zero contribution to $score(G)$ or it is safe to exclude a candidate group containing Q from S_k, thus all points outside of the (k-1)-skyband will not affect the result of TKD queries on skyline groups under MAX and MIN. We modify $[Q]$ in the following.*

$$[Q] = \begin{cases} [Q], Q \in (k-1) - skyband \\ 0...0, \quad Others \end{cases}.$$

5.3 Time Complexity Analysis

If we take the brute force method which employs existing skyline groups algorithms to compute all skyline groups, then we need to compute scores of all groups. For each skyline group G, the brute force method to compute $Score(G)$ needs to compare each point in G with all points in D, thus the time complexity of computing $Score(G)$ is $O(l \times n)$. Therefore, the time complexity of computing scores for all skyline groups is $O(|GSkyline| \times l \times n)$.

TABLE 5
Algorithms Tested in this Section

Brute force method: apply existing skyline groups algorithms and compute scores in the brute force way		TKD method: apply TKD algorithms and compute scores in the brute force way		TKD plus method: apply TKD algorithms and compute scores using bitmap index method	
BPer	BSUM	TPer	TSUM	TPer+	TSUM+
BMAX	BMIN	TMAX	TMIN	TMAX+	TMIN+

In our algorithms, if all skyline groups have the same score, we need to compute the scores of all skyline groups, which is the worst case. For each skyline group G, we need $l-1$ bit-wise operations to get $Score(G)$ using bitmap index method. For $Permutation$ and SUM, the time complexity of computing bit-vectors for points in $Skyline$ is $O(|Skyline| \times n)$. Therefore, the time complexity of computing scores for all skyline groups under $Permutation$ and SUM is $O(|GSkyline| \times l + |Skyline| \times n)$. Let $|Band|$ denote the size of *(k-1)-skyband*. The time complexity of computing bit-vectors for points in *(k-1)-skyband* is $O(|Band| \times n)$. Therefore, the time complexity of computing scores for all skyline groups under MAX and MIN is $O(|GSkyline| \times l + |Band| \times n)$.

Obviously, our bitmap index method significantly cuts down the cost of score computation.

We are aware that the large size of bitmap index may not be friendly to the memory. For instance, if $|D| = 10^6$, we need $|D|^2 = 10^{12}$ bits to store the bitmaps for all points in D. To this end, we integrate bitmap compression techniques in [11] with our bitmap index method. Please refer to [11] for implementation details. For instance, if $|D| = 10^6$ and $[Q]$ is made up of "0", then we only need three words (a word is made up of 32 bits) to represent $[Q]$. Therefore, combining with Lemmas 8 and 10, the bitmap compression techniques significantly cut down the size of bitmap index.

6 Experimental Evaluation

In this section, we conduct extensive experiments to evaluate the run-time performance and scalability of our algorithms. All our experiments are carried out on the same machine with 64 GB memory and dual eight-core Intel Xeon E7-4820 processors clocked at 2.0 Ghz. We implement all algorithms in Table 5. All algorithms are implemented in C++. To study the scalability of our algorithms, we generated correlated, independent and anti-correlated data sets using the standard skyline data generator from [1]. For real data sets, we use the NBA data set described in [31]. The data set consists of 17264 players and each player has 8 attributes.

6.1 Output Size of Skyline Groups

We do experiments on the NBA data set with the first 4 attributes. Because it is too expensive to compute skyline groups for all 8 attributes when $l \geq 4$. The output sizes of skyline groups under different definitions are shown in Table 6.

TABLE 6
Number of Skyline Groups under Different Definitions on NBA ($d = 4$) Data Set

l	*SUM*	*MAX*	*MIN*	*Permutation*
4	7685	1	2284	2.2M
5	18709	17260	4022	38M
6	40186	0.3B	6399	0.5B

NBA data set. M: million, B: billion.

TABLE 7
Storage Cost and Compression Ratios of Different Data Sets

Data set	n	d	Storage cost		Compression ratio	
			Skyline	(k-1)-Skyband	Skyline	(k-1)-Skyband
NBA	17264	8	0.209 MB	0.213 MB	5.9×10^{-3}	6.0×10^{-3}
Corr	$1M$	6	13.113 MB	23.842 MB	1.1×10^{-4}	2.0×10^{-4}
Indep	$1M$	6	50.067 MB	83.446 MB	4.2×10^{-4}	7.0×10^{-4}
Anti	$1M$	6	11.682 MB	11.801 MB	9.8×10^{-5}	9.9×10^{-5}

We can see that $|GSkyline|$ quickly becomes very large under all definitions with increasing l. The output sizes based on all definitions are too large to make quick selections. Therefore, it is not trivial to explore TKD queries on skyline groups.

6.2 Bitmap Index Compression Ratio

We use n^w words to represent the bitmap index. Then the *compression ratio* is computed as:

$$Compression\ ratio = \frac{|word| \times n^w}{n \times n}.$$

We set $k = 5$ and a $word$ is 32-bit long. Points are sorted in the descending order of L_1 norm, then the storage cost and compression ratios of different data sets are shown in Table 7. We find that the bitmap index storage cost of anti-correlated data set is less than that of correlated and independent data set. Since it is very difficult for a point in the anti-correlated data set to dominate other points, the bit vector of each skyline point in the anti-correlated data set is set to almost "0"s. Hence, the compression ratios of bitmap index of these skyline points are significant high. Though the skyline of the anti-correlated data set contains more points, the overall storage cost of the bitmap index of the anti-correlated data set is less. Table 7 indicates that the storage cost is low on all data sets of different distributions, which guarantees that our algorithms will not hurt the memory on challenging workloads.

TABLE 8
Parameter Ranges and Default Values

Parameter	Range
n	6000, 10000, 14000, **17264**
d	2, 3, **4**, 5
l	2, **3**, 4, 5
k	2, 4, **8**, 16

6.3 Experiments on Different Definitions

In this section, we report the performance of our algorithms under different skyline groups definitions. We experiment with the NBA data set. We experiment with different settings, including number of best skyline groups k, number of points n, number of dimensions d, and number of points in a group l. The settings of all these parameters are summarized in Table 8, where the default values are shown in bold. In every set of experiments, we only change one parameter, with the rest set to their defaults.

6.3.1 Experiments on Permutation

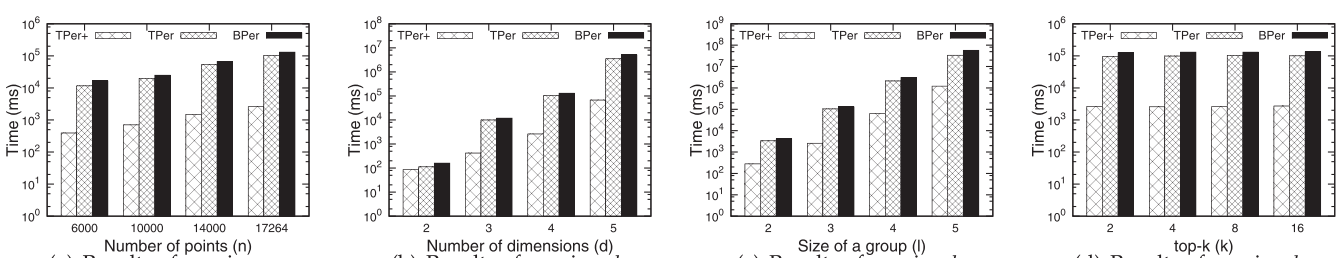
We compare the performance of *BPer*, *TPer*, and *TPer+* in different settings. From Fig. 9 we can see that the runtime of the three algorithms grows exponentially with increasing n, d, and l, while the runtime of the three algorithms keeps stable with increasing k.

The experimental results in the four sub figures show that *TPer* saves one third to half of the runtime of *BPer*, because in *TPer* we do not need to materialise all skyline groups. This significantly improves the performance of TKD queries on skyline groups under permutation. From the four sub figures we can see that *TPer+* outperforms *BPer* by 1-2 orders of magnitude. Because bitmap index method significantly cuts down the cost of score computation.

From the above analysis, we conclude that integrating with effective pruning techniques and bitmap index method, our approach is very efficient to compute top-k dominating skyline groups under permutation in different settings.

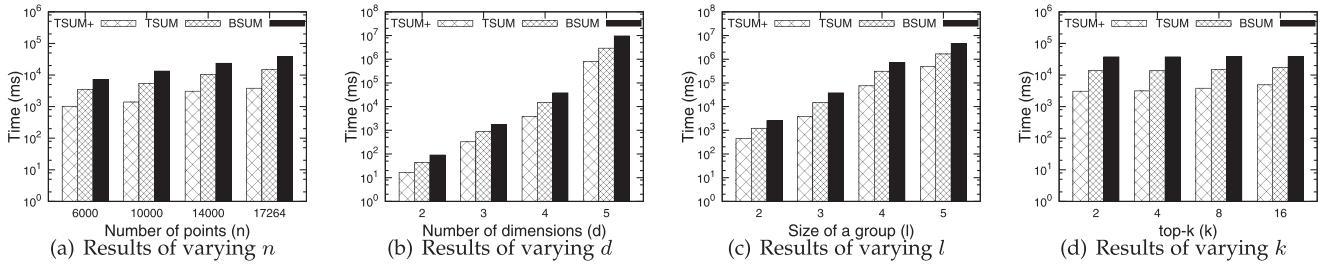
6.3.2 Experiments on SUM

We compare the performance of *BSUM*, *TSUM*, and *TSUM+* in different settings. From Section 4.2 we know that *BSUM* needs to process all points in the *(k-1)-skyband* while our algorithms can return top-k dominating skyline groups without processing all points. From experimental results we find that in most cases, after processing less than half of the points in the *(k-1)-skyband*, our algorithms can return the top-k dominating skyline groups.



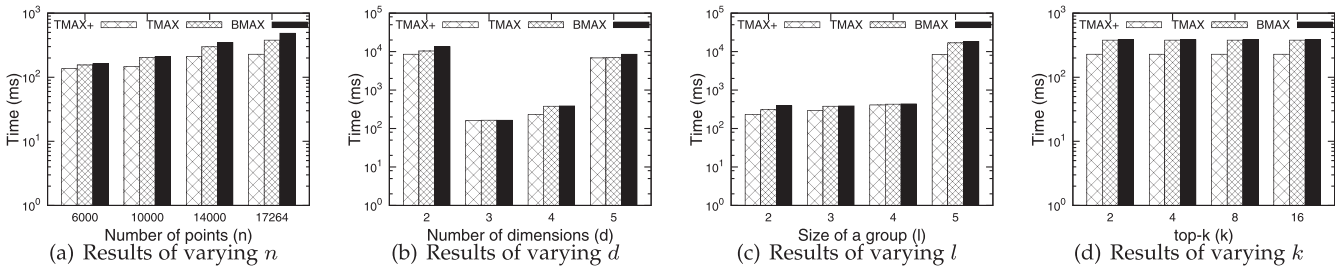
(a) Results of varying n (b) Results of varying d (c) Results of varying l (d) Results of varying k

Fig. 9. Experimental results on permutation.



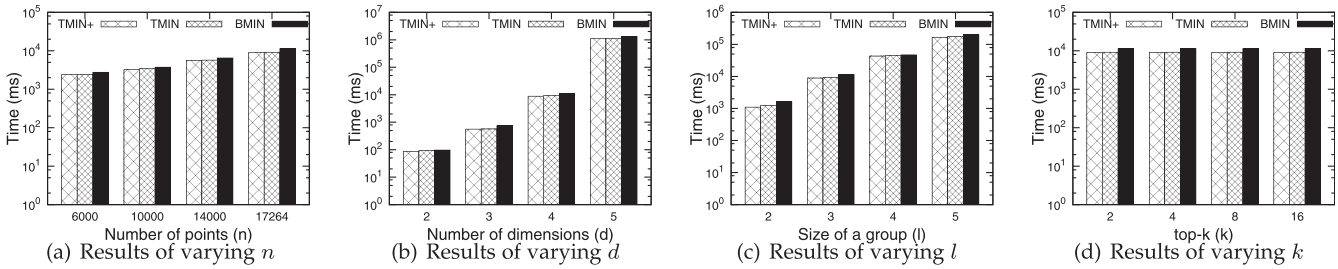
(a) Results of varying n (b) Results of varying d (c) Results of varying l (d) Results of varying k

Fig. 10. Experimental results on SUM.



(a) Results of varying n (b) Results of varying d (c) Results of varying l (d) Results of varying k

Fig. 11. Experimental results on MAX.



(a) Results of varying n (b) Results of varying d (c) Results of varying l (d) Results of varying k

Fig. 12. Experimental results on MIN.

Similar to *Permutation*, the runtime of the three algorithms grows exponentially with increasing n, d, and l, while the runtime of the three algorithms keeps stable with increasing k.

From the four sub figures in Fig. 10 we can see that *TSUM+* and *TSUM* outperform *BSUM* under all circumstances. *TSUM* is about $2.4\times$ to $3.5\times$ faster than *BSUM* while *TSUM+* is about $4.3\times$ to $11.3\times$ faster than *BSUM* on all data sets. Experimental results on all circumstances validate that our pruning techniques and bitmap index method are effective and efficient.

6.3.3 Experiments on MAX

From Section 4.3 we know that *BMAX* needs to construct skyline groups for all skyline vectors while our algorithms can return top-k dominating skyline groups after processing at most k skyline vectors.

From the four sub figures in Fig. 11 we can see that *TMAX+* and *TMAX* outperform *BMAX* on all circumstances. We make an important observation for MAX when $l \geq d$, there is only one distinct skyline vector which takes the MAX value on every attribute. Therefore, when building skyline groups, after finding points that cover all the MAX values, the rest points can be arbitrary. Thus the number of skyline groups can be extraordinarily large. As shown in Fig. 11c, the runtime of three algorithms in the case of $l = 5$ is significantly more than the runtime of other cases. Though our pruning techniques do not work in the case of $l \geq d$, by employing the bitmap index method to compute scores of groups efficiently, *TMAX+* achieves much better performance than *BMAX* as shown in Fig. 11c in the case of $l = 5$. The experimental results show that *TMAX+* is about $1.5\times$ faster than *BMAX* on challenging workloads.

6.3.4 Experiments on MIN

As shown in Fig. 12, the runtime of three algorithms grows with increasing n, d, and l, while the runtime of the three algorithms keeps stable with increasing k. From the four sub figures in Fig. 12, we can see that *TMIN+* and *TMIN* outperform *BMIN* on all circumstances. The speedup of *TMIN+* over *BMIN* increases with increasing n, d, and l. Thus *TMIN+* reaches much better performance on challenging workloads. Experimental results in Fig. 12 show that *TMIN+* is about $1.3\times$ speedup over *BMIN*.

6.4 Experiments on Different Distributions

In this section, we report the performance of our algorithms on synthetic data sets of different distributions. As shown in Figs. 13, 14, and 15 the performance of the same algorithm varies dramatically under different data distributions.

This is not surprising that on the data sets of the same size the runtime of *TPer+*,*TSUM+*,*TMAX+*, and *TMIN+* is the minimal on correlated data sets. The reason lies in that

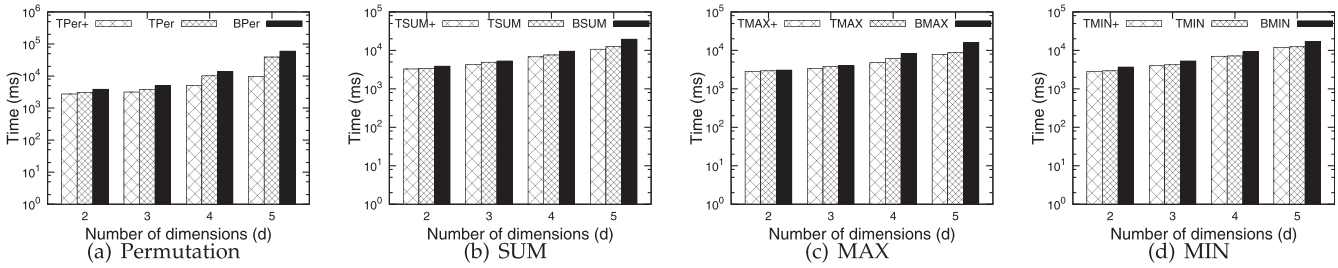


Fig. 13. Experimental results on correlated data sets ($n = 1\ million, l = 2, k = 8$).

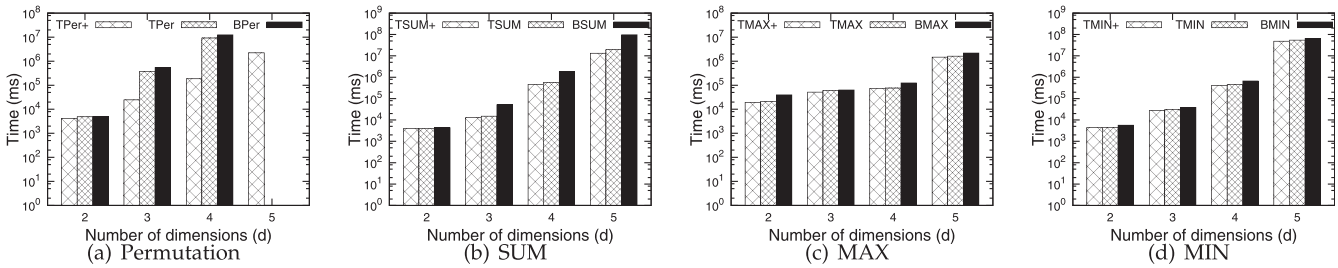


Fig. 14. Experimental results on independent data sets ($n = 1\ million, l = 2, k = 8$).

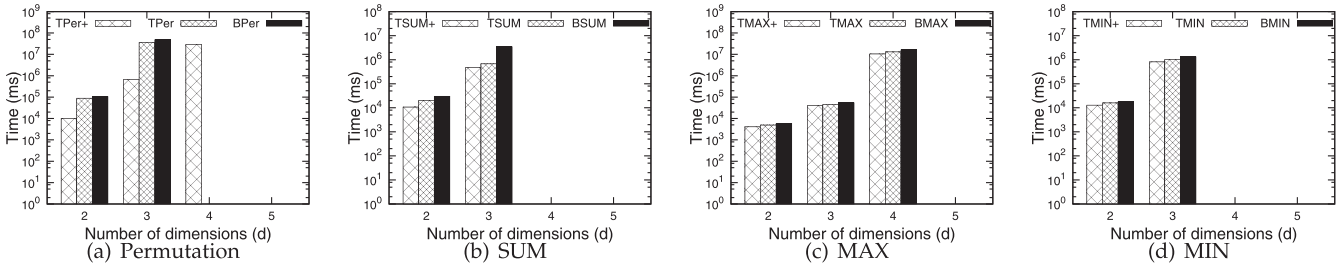


Fig. 15. Experimental results on anti-correlated data sets ($n = 1\ million, l = 2, k = 8$).

large number of points are pruned in the input pruning. Thus the workload on the correlated data sets are much smaller compared to the independent data sets and the anti-correlated data sets of the same size. On anti-correlated data sets, it is more difficult for a point to dominate other points thus there are much more candidate points to build skyline groups. Obviously, the workload of the anti-correlated data sets are the heaviest among the three distributions. We can see that the runtime on independent and anti-correlated data sets increases quickly with increasing d. Thus *TPer* and *BPer* cannot finish within reasonable amount of time on the independent data sets when $d \geq 5$. Moreover, as shown in Fig. 15 it takes too much time for all algorithms to finish on the anti-correlated data set when $d \geq 5$. Therefore, the corresponding bars are omitted.

Though the runtime varies dramatically, the speedups of our algorithms over the brute force method are similar to that on the NBA data set. We report the experimental results of the speedups of our algorithms over the brute force method in Table 9.

TABLE 9
Speedups on Different Distributions
($n = 1\ million, l = 2, d = 3, k = 8$)

Data set	*Speedup*			
	TPer+	*TSUM+*	*TMAX+*	*TMIN+*
Corr	1.7	1.2	1.2	1.2
Indep	22.5	4.9	2.0	1.3
Anti	77.0	14.7	2.3	1.3

The speedup is obtained by dividing the time of brute force algorithm by the time of corresponding TKD+ algorithm. From Figs. 13, 14 and 15, we can see that $d = 3$ is the maximum dimension value that all algorithms can finish. So we fix $d = 3$ to compute the speedups. Experimental results in Table 9 show that our algorithms reach better performance on challenging workloads. For correlated data sets, since most points are pruned in the input pruning, the candidate points to form skyline groups are quite few. Therefore, the speedups are similar on correlated data sets. From the experimental results in Table 9, we find that our algorithms preserve the advantages on different data distributions, which validates the scalability of our algorithms.

7 Conclusions

In this paper we carry out a systematic study of TKD queries on skyline groups under all existing skyline group definitions. To the best of our knowledge, this is the first study on processing TKD queries on skyline groups. We develop various effective pruning techniques, with which we can return top-k dominating skyline groups without materialising all skyline groups. We significantly improve the efficiency of score computation by designing a bitmap index method. Using the bitmap index compression techniques, we significantly cut down the size of bitmap index, which makes our

algorithms friendly to the memory. Moreover, extensive experiments on real and synthetic data sets have verified the effectiveness and the efficiency of our methods.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant No. 2018YFB0203801, the NSFC under Grant No. 61702250, No.61702539, No.61572510, and No. 61502511, and the Tecent Research Award 11002675.

References

[1] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. 17th Int. Conf. Data Eng.*, 2001, pp. 421–430.
[2] H. Im and S. Park, "Group skyline computation," *Inf. Sci.*, vol. 188, pp. 151–169, 2012.
[3] C. Li, N. Zhang, N. Hassan, S. Rajasekaran, and G. Das, "On skyline groups," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 2119–2123.
[4] Y.-C. Chung, I.-F. Su, and C. Lee, "Efficient computation of combinatorial skyline queries," *Inf. Syst.*, vol. 38, no. 3, pp. 369–387, 2013.
[5] J. Liu, L. Xiong, J. Pei, J. Luo, and H. Zhang, "Finding pareto optimal groups: Group-based skyline," *Proc. VLDB Endowment*, vol. 8, no. 13, pp. 2086–2097, 2015.
[6] N. Zhang, C. Li, N. Hassan, S. Rajasekaran, and G. Das, "On skyline groups," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 942–956, Apr. 2014.
[7] H. Zhu, P. Zhu, X. Li, and Q. Liu, "Computing skyline groups: an experimental evaluation," in *Proc. ACM Turing 50th Celebration Conf.*, 2017, pp. 48:1–48:6.
[8] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[9] M. L. Yiu and N. Mamoulis, "Efficient processing of top-k dominating queries on multi-dimensional data," *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 483–494.
[10] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator," in *IEEE 23rd Int. Conf. Data Eng.*, 2007, pp. 86–95.
[11] K. Wu, E. J. Otoo, and A. Shoshani, "Compressing bitmap indexes for faster search operations," in *Proc. Int. Conf. Sci. Statist. Database Manage.*, 2002, pp. 99–108.
[12] H. Zhu, P. Zhu, X. Li, and Q. Liu, "Top-k skyline groups queries," in *Proc. Int. Conf. Extending Database Technol.*, 2017, pp. 442–445.
[13] I. F. Su, Y. C. Chung, and C. Lee, "Top-k combinatorial skyline queries," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2010, pp. 79–93.
[14] M. L. Yiu and N. Mamoulis, "Multi-dimensional top-k dominating queries," *VLDB J.*, vol. 18, no. 3, pp. 695–718, 2009.
[15] E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos, "Progressive processing of subspace dominating queries," *VLDB J.*, vol. 20, no. 6, pp. 921–948, 2011.
[16] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous top-k dominating queries," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 5, pp. 840–853, May 2012.
[17] B. J. Santoso and G. M. Chiu, "Close dominance graph: An efficient framework for answering continuous top-k dominating queries," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1853–1865, Aug. 2014.
[18] X. Han, J. Li, and H. Gao, "Tdep: Efficiently processing top-k dominating query on massive data," *Knowl. Inf. Syst.*, vol. 43, no. 3, pp. 689–718, 2015.
[19] E. Tiakas, G. Valkanas, A. N. Papadopoulos, Y. Manolopoulos, and D. Gunopulos, "Metric-based top-k dominating queries," in *Proc. EDBT*, 2014, pp. 415–426.
[20] X. Lian and L. Chen, "Top-k dominating queries in uncertain databases," in *Proc. Int. Conf. Extending Database Technol.*, 2009, pp. 660–671.
[21] W. Zhang, X. Lin, Y. Zhang, J. Pei, and W. Wang, "Threshold-based probabilistic top- k dominating queries," *VLDB J.*, vol. 19, no. 2, pp. 283–305, 2010.
[22] L. Zhan, Y. Zhang, W. Zhang, and X. Lin, "Identifying top k dominating objects over uncertain data," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2014, pp. 388–405.
[23] X. Miao, Y. Gao, B. Zheng, G. Chen, and H. Cui, "Top-k dominating queries on incomplete data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 252–266, Jan. 2016.
[24] Y. Dong, H. Chen, J. X. Yu, K. Furuse, and H. Kitagawa, "Grid-index algorithm for reverse rank queries," in *Proc. Int. Conf. Extending Database Technol.*, 2017, pp. 306–317.
[25] Y. Qian, H. Li, N. Mamoulis, Y. Liu, and D. W. Cheung, "Reverse k-ranks queries on large graphs," in *Proc. Int. Conf. Extending Database Technol.*, 2017, pp. 37–48.
[26] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang, "Extracting top-k insights from multi-dimensional data," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 1509–1524.
[27] G. Yang and Y. Cai, "Querying improvement strategies," in *Proc. Int. Conf. Extending Database Technol.*, 2017, pp. 294–305.
[28] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-based representative skyline," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 892–903.
[29] M. Magnani, I. Assent, and M. L. Mortensen, "Taking the big picture: Representative skylines based on significance and diversity," *VLDB J.*, vol. 23, no. 5, pp. 795–815, 2014.
[30] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. J. Xu, "Regret-minimizing representative databases," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 1114–1124, 2010.
[31] S. Chester, D. Sidlauskas, I. Assent, and K. S. Bogh, "Scalable parallelization of skyline computation for multi-core processors," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 1083–1094.

Haoyang Zhu received PhD degree in computer science and technology from the College of Computer Science, National University of Defense Technology, China, in 2017. Now, he is a research associate with the Academy of Military Sciences, People's Liberation Army. His current research interests include massive data processing, and query processing and optimization. He is a member of the CCF and ACM.

Xiaoyong Li received the PhD degree in computer science and technology from the College of Computer Science, National University of Defense Technology, China, in 2013. Now he is a research associate with the National University of Defense Technology. His current research interests include data stream management, parallel computing, uncertain queries, and cloud computing. He is a member of the CCF, IEEE, and ACM.

Qiang Liu received the PhD degree in computer science and technology from the College of Computer Science, National University of Defense Technology, China, in 2014. Now, he is an assistant professor with the National University of Defense Technology. His current research interests include 5G network, internet of things, and machine learning. He currently serves on the editorial review board of the *Artificial Intelligence Research* journal. He is a member of the CCF and IEEE.

Zichen Xu received the PhD degree from the Ohio State University. He is a full professor with Nanchang University, China. His research spans in the area of data-conscious complex system, including profile analysis, system optimization, and storage system design/implementation. He is a member of the IEEE and ACM.