Information Sciences 188 (2012) 151–169

Contents lists available at SciVerse ScienceDirect

Information Sciences

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

Group skyline computation

Hyeonseung Im, Sungwoo Park *

Pohang University of Science and Technology (POSTECH), Republic of Korea

ARTICLE INFO

Article history:
Received 22 October 2010
Received in revised form 13 October 2011
Accepted 10 November 2011
Available online 22 November 2011

Keywords:
Skyline computation
Dynamic algorithm

ABSTRACT

Given a multi-dimensional dataset of tuples, skyline computation returns a subset of tuples that are not dominated by any other tuples when all dimensions are considered together. Conventional skyline computation, however, is inadequate to answer various queries that need to analyze not just individual tuples of a dataset but also their combinations. In this paper, we study group skyline computation which is based on the notion of dominance relation between groups of the same number of tuples. It determines the dominance relation between two groups by comparing their aggregate values such as sums or averages of elements of individual dimensions, and identifies a set of skyline groups that are not dominated by any other groups. We investigate properties of group skyline computation and develop a group skyline algorithm GDynamic which is equivalent to a dynamic algorithm that fills a table of skyline groups. Experimental results show that GDynamic is a practical group skyline algorithm.

1. Introduction

Given a multi-dimensional dataset of tuples, skyline computation [4] returns a subset of tuples that are no worse than, or not dominated by, any other tuples when all dimensions are considered together. As an example, consider the dataset of six basketball players in Fig. 1. Each tuple lists the number of points and the number of rebounds. As we prefer higher values for both attributes, players p_1, p_2, and p_3 (marked with •) are the best choices which are no worse than any other players, while players p_4, p_5, and p_6 (marked with ○) are all inferior choices which are dominated by player p_1 or p_3. We say that players p_1, p_2, and p_3 are *skyline tuples*, and refer to the set of skyline tuples as the *skyline* of the dataset. A skyline algorithm performs comparisons between tuples, or *dominance tests*, to identify all skyline tuples, or equivalently, to eliminate all non-skyline tuples.

Because of its potential applications in decision making, skyline computation since its inception has drawn a lot of attention in the database community [2,4,5,9,12,13,18,28]. Researchers have also investigated its extensions such as parallel skyline computation for share-nothing architectures [8,25,27] and multicore architectures [19,21], probabilistic skyline computation on uncertain datasets and data streams [1,3,6,20], and skycube computation for finding all possible $2^d - 1$ subspace skylines of the given d-dimensional dataset [11].

Conventional skyline computation, however, is inadequate to answer various queries that require us to analyze not just individual tuples of a dataset but also their combinations. For example, it is not clear how to answer the following queries (on the dataset in Fig. 1) using only conventional skyline computation when we wish to form a team of three players:

- *Our team has currently two players p_1 and p_5. Who is the best player to recruit as a new member?*

* Corresponding author. Tel.: +82 54 279 2386.
E-mail addresses: genilhs@postech.ac.kr (H. Im), gla@postech.ac.kr (S. Park).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.11.014

Player	Points	Rebounds
p_1	3	3
p_2	0	4
p_3	4	1
p_4	2	3
p_5	2	2
p_6	2	1

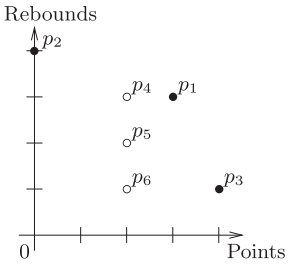

Fig. 1. A dataset of basketball players and the skyline.

- *Who is the best player to choose if the other two players in our team are unknown yet? In particular, is non-skyline player p_4 a better choice than skyline player p_2 or p_3?*

In essence, these queries call for an analysis of groups of three players, which is beyond the reach of conventional skyline computation.

In this paper, we study *group skyline computation* which is based on the notion of dominance relation between groups of the same number of tuples. It determines the dominance relation between two groups by comparing their aggregate values of elements of individual dimensions. We may use any aggregate function that is *strictly monotone in each argument* [7] such as sum and average. For the purpose of easy illustration, however, we use the aggregate function sum in all the motivating examples in the paper.

As an example of group skyline computation, consider again the dataset in Fig. 1. We wish to find groups of three players whose sums of points and rebounds are no worse than any other groups of three players. After examining a total of $C(6,3) = 20$ candidate groups, we obtain four such groups:

Group of players	Sum of points	Sum of rebounds
$\{p_1, p_2, p_3\}$	7	8
$\{p_1, p_2, p_4\}$	5	10
$\{p_1, p_3, p_4\}$	9	7
$\{p_1, p_4, p_5\}$	7	8

We refer to these groups as 3-*skyline groups* of the dataset where 3 indicates the number of tuples in each group, and refer to the set of 3-skyline groups as the 3-*skyline* of the dataset. Given a value of k specifying the number of tuples in each group, group skyline computation finds the k-skyline, or the set of k-skyline groups, of the dataset.

Group skyline computation has a number of potential applications in decision making that need to compare groups of tuples. For example, we can answer the two previous queries using group skyline computation:

- *Our team has currently two players p_1 and p_5. Who is the best player to recruit as a new member?*
  We perform group skyline computation on the dataset to find 3-skyline groups with both p_1 and p_5. Since there is only one such 3-skyline group $\{p_1, p_4, p_5\}$, we choose to recruit non-skyline player p_4, instead of remaining skyline players p_2 and p_3. If we perform conventional skyline computation after removing players p_1 and p_5, we obtain three players p_2, p_3, and p_4 to choose from.
- *Who is the best player to choose if the other two players in our team are unknown yet? In particular, is non-skyline player p_4 a better choice than skyline player p_2 or p_3?*
  The goal is to find a player who maximizes the probability that the team with the player belongs to the 3-skyline. We perform group skyline computation on the dataset and count the number of 3-skyline groups to which each player belongs. We rank all the players by these numbers and obtain the following list: p_1 (4), p_4 (3), p_2/p_3 (2), p_5 (1), p_6 (0). Note that we rank non-skyline player p_4 higher than skyline players p_2 and p_3.

Thus we may exploit group skyline computation to answer various queries that conventional skyline computation either cannot answer or answers incorrectly.

The main challenge in developing an algorithm for group skyline computation is to overcome its computational complexity. Unfortunately the space complexity of group skyline computation is exponential in general and a dataset of n tuples can have up to $C(n,k) = \frac{n!}{(n-k)!k!}$ k-skyline groups. For example, we can construct such a dataset by placing all tuples in the same hyperplane. This, in turn, means that its time complexity is also exponential because we have to generate all skyline groups during the computation. Since the set of intermediate candidate groups may not fit in memory, we have to generate all candidate groups in a progressive manner and update the resultant group skyline dynamically. As such, we thus cannot

exploit those techniques, such as index structures R-trees [18] and ZBtrees [13], developed for conventional skyline computation.

We develop a group skyline algorithm called GDynamic which exploits key properties of group skyline computation to reduce both the number of intermediate candidate groups and the number of dominance tests between groups. As a preliminary step, we study a simple group skyline algorithm called GIncremental which is based on the intuition that every k-skyline group is likely to contain at least one $(k-1)$-skyline group. (We prove that the intuition is valid for $k \leqslant 3$, but not for $k \geqslant 4$.) After analyzing the properties of GIncremental, we design GDynamic which is equivalent to a dynamic algorithm that fills a table of group skylines. Experimental results on both synthetic and real datasets show that GDynamic is a practical group skyline algorithm.

A recent proposal of top-k combinatorial skyline query (k-CSQ) processing by Su et al. [22] is similar in spirit to group skyline computation in that it also considers combinations of tuples and computes combinations that are not dominated by any others. However, k-CSQ processing is different in that it reports only those combinations whose aggregate values for a certain attribute are the highest, and thus may miss some important combinations. In contrast, group skyline computation generates all skyline groups, which may be exploited in more complicated decision making processes. Hence its main challenge is to find key properties for reducing both the number of intermediate candidate groups and the number of dominance tests between groups. Section 6 provides a more detailed discussion on k-CSQ processing.

The remainder of the paper is organized as follows. Section 2 introduces group skyline computation and studies its basic properties. Sections 3 and 4 develop the algorithm GIncremental and GDynamic, respectively. Section 5 presents experimental results and Section 6 discusses related work. Section 7 concludes.

2. Preliminaries

This section formally defines group skyline computation and studies its basic properties. Fig. 2 summarizes the key notations used throughout the paper. We begin by reviewing conventional skyline computation.

2.1. Skyline computation

Given a dataset, skyline computation retrieves its skyline which consists of tuples not dominated by any other tuples. Under the assumption that larger values are better, a tuple p dominates another tuple q if all elements of p are larger than or equal to their corresponding elements of q and there exists at least one element of p that is strictly larger than its corresponding element of q. Thus the skyline consists of those tuples that are no worse than any other tuples when all dimensions are considered together.

Let us formally define the skyline of a d-dimensional dataset D. We write $p[i]$ for the i-th element of tuple p where $1 \leqslant i \leqslant d$; hence we have $p = (p[1], \ldots, p[d])$. We write $p \succ q$ to mean that tuple p dominates tuple q, that is, $p[i] \geqslant q[i]$ holds for $1 \leqslant i \leqslant d$ and there exists a dimension j such that $p[j] > q[j]$. We also write $p \not\succ q$ to mean that p does not dominate q, and $p \prec\!\succ q$ to mean that p and q are incomparable ($p \not\succ q$ and $q \not\succ p$). Then the skyline $\mathcal{S}(D)$ of D is defined as

$$\mathcal{S}(D) = \{p \in D \mid \forall q \in D,\ q \not\succ p\}.$$

Note that $\mathcal{S}(D) \subset D$ and $\mathcal{S}(\mathcal{S}(D)) = \mathcal{S}(D)$ hold. We refer to those tuples in the skyline as skyline tuples.

The cost of skyline computation increases with the number of dominance tests performed to identify skyline tuples. A dominance test between two tuples p and q determines whether p dominates q ($p \succ q$), q dominates p ($q \succ p$), or p and q are incomparable ($p \prec\!\succ q$). Usually a skyline algorithm reduces the number of dominance tests by exploiting specific

Notation	Meaning
p, q, r	tuple
f	aggregate function that is strictly monotone in each argument
$\mathcal{S}(D)$	skyline of D
C_k	k-group
G_k	k-skyline group
$f(C_k)$	$(f(p_1[1], \cdots, p_k[1]), \cdots, f(p_1[d], \cdots, p_k[d]))$ where $C_k = \{p_1, \cdots, p_k\}$
$\mathcal{C}_k$	set of k-groups
$\mathcal{G}_k$	set of k-skyline groups
$\mathcal{C}_k(D)$	set of all k-groups of D
$\mathcal{G}_k(D)$	k-skyline of D
$\mathcal{T}_k(D)$	set of candidate tuples for computing $\mathcal{G}_k(D)$

Fig. 2. Summary of key notations.

properties of skyline tuples. For example, transitivity of $\succ$ allows us to eliminate from further consideration any tuple as soon as we find that it is dominated by another tuple:

Proposition 2.1. *If $p \succ q$ and $q \succ r$, then $p \succ r$.*

2.2. Group skyline computation

Group skyline computation extends the notion of dominance relation to groups of the same number of tuples and identifies a group skyline, or a set of groups that are no worse than any other groups. It determines the dominance relation between two groups by comparing their aggregate values of elements of individual dimensions. Hence we may think of group skyline computation on a dataset as conventional skyline computation on a derived dataset consisting of all its subsets of the same size.

For dominance tests, we may use any aggregate function that is strictly monotone in each argument [7]. An aggregate function f is *monotone* if $f(x_1,\ldots,x_m) \geqslant f(x'_1,\ldots,x'_m)$ whenever $x_i \geqslant x'_i$ for every i. f is *strictly monotone* if $f(x_1,\ldots,x_m) > f(x'_1,\ldots,x'_m)$ whenever $x_i > x'_i$ for every i. f is *strictly monotone in each argument* if an increase in any argument results in an increase of its value. That is, f is strictly monotone in each argument if $f(x_1,\ldots,x_i,\ldots,x_m) > f(x_1,\ldots,x'_i,\ldots,x_m)$ whenever $x_i > x'_i$ for some i. For example, sum and average are strictly monotone in each argument, whereas min and max are not. In the rest of the paper, we write "aggregate functions" to mean those that are strictly monotone in each argument.

Now let us formally define group skylines of a d-dimensional dataset D using an aggregate function f. We refer to a subset C_k of D containing k tuples as a k-group of D and write $\mathcal{C}_k(D)$ for the set of all k-groups of D:

$$\mathcal{C}_k(D) = \{C_k \mid C_k \subset D, |C_k| = k\}$$

As a special case, we have $\mathcal{C}_1(D) = \{\{p\} \mid p \in D\}$.

Given a k-group $C_k = \{p_1,\ldots,p_k\}$, we write $f(C_k)$ for the aggregate tuple $(f(p_1[1],\ldots,p_k[1]),\ldots,f(p_1[d],\ldots,p_k[d]))$. We define the dominance relation $C_k \succ C'_k$ between two k-groups C_k and C'_k as follows:

Definition 2.2. Suppose $C_k = \{p_1,\ldots,p_k\}$ and $C'_k = \{q_1,\ldots,q_k\}$. We say that C_k dominates C'_k and write $C_k \succ C'_k$ if $f(C_k) \succ f(C'_k)$ holds, that is, $f(p_1[i],\ldots,p_k[i]) \geqslant f(q_1[i],\ldots,q_k[i])$ holds for $1 \leqslant i \leqslant d$ and there exists a dimension j such that $f(p_1[j],\ldots,p_k[j]) > f(q_1[j],\ldots,q_k[j])$.

As in conventional skyline computation, we write $C_k \not\succ C'_k$ to mean that C_k does not dominate C'_k, and $C_k \prec\!\!\succ C'_k$ to mean that C_k and C'_k are incomparable ($C_k \not\succ C'_k$ and $C'_k \not\succ C_k$). Then the group skyline of size k, or k-skyline, of D is defined as

$$\mathcal{G}_k(D) = \mathcal{S}(\mathcal{C}_k(D)) = \{C_k \in \mathcal{C}_k(D) \mid \forall C'_k \in \mathcal{C}_k(D),\ C'_k \not\succ C_k\}.$$

We refer to those groups in the k-skyline as k-skyline groups and write G_k for a k-skyline group (*i.e.*, $G_k \in \mathcal{G}_k(D)$). As a special case, we have $\mathcal{G}_1(D) = \{\{p\} \mid p \in \mathcal{S}(D)\}$.

2.3. Basic properties of group skyline computation

A trivial but important property of group skyline computation is that its space complexity is exponential in general. This, in turn, means that its time complexity is also exponential because we have to generate all skyline groups during the computation.

Proposition 2.3. *For a value of k specifying the size of group skylines, a dataset with n tuples can have up to $C(n,k) = \frac{n!}{(n-k)!k!}$ k-skyline groups. That is, all its k-groups can be in the k-skyline.*

Proof. Consider the summation Σ as an aggregate function and a 2-dimensional dataset $D = \{(x,-x) \mid x = 1,\ldots,n\}$. Every k-group of D is written as $(\Sigma_{i=1}^{k} x_i,\ -\Sigma_{i=1}^{k} x_i)$ and belongs to the k-skyline. □

Since we cannot achieve a group skyline algorithm of polynomial space or time complexity, we focus on those properties of group skyline computation that enable us to consider fewer intermediate candidate groups on typical datasets not all of whose k-groups are in the k-skyline.

Since a k-skyline group collectively represents a single "good" tuple, it is likely to contain skyline tuples. Proposition 2.4, however, shows that a group of k skyline tuples does not necessarily form a k-skyline group:

Proposition 2.4. *Suppose $C_k \subset \mathcal{S}(D)$. Then we may have $C_k \notin \mathcal{G}_k(D)$.*

Proof. Consider the summation Σ as an aggregate function and a 2-dimensional dataset of 4 tuples: $p_1 = (0,4)$, $p_2 = (1,2)$, $p_3 = (2,1)$, and $p_4 = (4,0)$. All tuples are skyline tuples, but $\{p_1,p_4\}$ dominates $\{p_2,p_3\}$. □

Proposition 2.5 proves the converse of Proposition 2.4 that a k-skyline group may indeed contain non-skyline tuples:

Proposition 2.5. *Suppose $G_k \in \mathcal{G}_k(D)$. Then we may have $G_k \not\subset \mathcal{S}(D)$.*

Proof. Consider the summation Σ as an aggregate function and a 2-dimensional dataset of 3 tuples: $p_1 = (0,2)$, $p_2 = (1,0)$, and $p_3 = (2,1)$. p_2 is dominated by p_3 and thus is not a skyline tuple, but $\{p_2, p_3\}$ is a 2-skyline group. □

Hence group skyline computation needs to consider not only skyline tuples but also non-skyline tuples. This result, however, does not mean that we should treat all non-skyline tuples in the same manner – Proposition 2.6 shows that a k-skyline group may contain a non-skyline tuple, but along with all its dominating tuples:

Proposition 2.6. *Suppose $p \in G_k \in \mathcal{G}_k(D)$. Then $q \succ p$ with $q \in D$ implies $q \in G_k$.*

Proof. Suppose $q \notin G_k$ and $G_k = \{p_1, \ldots, p_k\}$. Without loss of generality, assume $p_k = p$. From $q \succ p$, we have $q[i] \geqslant p[i]$ for every i and there exists j such that $q[j] > p[j]$. Then, given an aggregate function f, we have $f(p_1[i], \ldots, p_{k-1}[i], q[i]) \geqslant f(p_1[i], \ldots, p_{k-1}[i], p[i])$ for every i by the monotonicity of f. Moreover, by the strict monotonicity in each argument and $q[j] > p[j]$, $f(p_1[j], \ldots, p_{k-1}[j], q[j]) > f(p_1[j], \ldots, p_{k-1}[j], p[j])$. Hence, by Definition 2.2, we have $(G_k - \{p\}) \cup \{q\} \succ G_k$, which contradicts $G_k \in \mathcal{G}_k(D)$. For example, in Fig. 1, player p_5 belongs to a 3-skyline group G_3 only if players p_1 and p_4 both belong to G_3. □

Hence a k-skyline group contains at least one skyline tuple.

An important corollary to Proposition 2.6 is that for computing a k-skyline, we may safely discard all non-skyline tuples that are dominated by k or more tuples. We say that a tuple p is k-dominated if it is dominated by k or more tuples in the same dataset, *i.e.*, if there exists C_k such that $q \in C_k$ implies $q \succ p$.

Corollary 2.7. *For a k-dominated tuple $p \in D$, there is no k-skyline group $G_k \in \mathcal{G}_k(D)$ such that $p \in G_k$.*

Proof. Suppose that a k-skyline group G_k contains a k-dominated tuple p. Then there exist k distinct tuples that dominate p. By Proposition 2.6, G_k should also contain all those k tuples. Since its size is k, G_k cannot contain all k dominating tuples and p at the same time, which contradicts the assumption. □

Proposition 2.6 and Corollary 2.7 are the basis for our group skyline algorithms. Given a dataset D and the size k of the group skyline, we first exploit Corollary 2.7 to remove all k-dominated tuples. Then we consider only those k-groups that satisfy the condition in Proposition 2.6. Without these two optimizations, even a dataset of moderate size with a small value of k generates an intractable number of candidate k-groups. For example, a dataset of 1000 tuples with $k = 5$ generates $C(1000, 5) = 8250291250200$ candidate 5-groups.

Below we write $\mathcal{T}_k(D)$ for the set of tuples obtained by removing all k-dominated tuples from D. $\mathcal{T}_k(D)$ is the set of candidate tuples that we need to consider when computing $\mathcal{G}_k(D)$. We have $\mathcal{S}(D) \subset \mathcal{T}_k(D)$.

3. Incremental group skyline computation

In designing an algorithm for group skyline computation, we need to devise a scheme for dealing with its exponential space complexity. In principle, the exponential space complexity is due to the worst-case memory requirement for storing skyline groups, but in practice, the main obstacle lies in storing intermediate candidate groups, which may not fit in memory. Hence a practical group skyline algorithm never generates all candidate groups at once; rather it generates candidate groups in a progressive manner and updates the resultant group skyline dynamically. This, in turn, means that we cannot exploit those techniques, such as index structures R-trees [18] and ZBtrees [13], developed for conventional skyline computation.

Since we generate candidate groups in a progressive manner, the speed of a group skyline algorithm depends to some extent on the order in which it examines candidate groups. As a preliminary step to developing our main group skyline algorithm in Section 4, we study an intuitive group skyline algorithm and investigate other subtle properties of group skyline computation.

3.1. Group skyline algorithm GIncremental

Recall from Section 2 that a k-skyline group consists only of skyline tuples or non-skyline tuples that are dominated by at most $(k-1)$ tuples. Since all these k tuples are likely to be "good" tuples, a k-skyline group is also likely to contain at least one $(k-1)$-skyline group as its subset. This intuition leads to a simple group skyline algorithm GIncremental which successively computes $\mathcal{G}_k(D)$ from $\mathcal{G}_{k-1}(D)$ beginning with $\mathcal{G}_1(D) = \{\{p\} \mid p \in \mathcal{S}(D)\}$.

Given $(i-1)$-skyline $\mathcal{G}_{i-1}$, GIncremental generates a candidate i-group by extending an $(i-1)$-skyline group G_{i-1} with a tuple p in $\mathcal{T}_k(D)$ while testing the condition in Proposition 2.6. (To be specific, GIncremental checks whether G_{i-1} contains every tuple that dominates p.) Then it attempts to insert the candidate i-group $G_{i-1} \cup \{p\}$ to the current i-skyline $\mathcal{G}_i$. GIncremental produces all candidate groups in a progressive manner and considers only one candidate group $G_{i-1} \cup \{p\}$ at a time.

$$
\begin{array}{lcl}
p_1 & = & (\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\quad 4,\ 1,\ 1,\ 1,\quad 3,\ 3,\ 3,\ 0,\ 0,\ 0\ ) \\
p_2 & = & (\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\quad 1,\ 4,\ 1,\ 1,\quad 3,\ 0,\ 0,\ 3,\ 3,\ 0\ ) \\
p_3 & = & (\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\quad 1,\ 1,\ 4,\ 1,\quad 0,\ 3,\ 0,\ 3,\ 0,\ 3\ ) \\
p_4 & = & (\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\quad 1,\ 1,\ 1,\ 4,\quad 0,\ 0,\ 3,\ 0,\ 3,\ 3\ ) \\
q_1 & = & (\ 2,\text{-}1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\quad 0,\ 4,\ 4,\ 4,\quad 1,\ 1,\ 1,\ 4,\ 4,\ 4\ ) \\
q_1' & = & (\text{-}1,\ 2,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\quad 0,\ 4,\ 4,\ 4,\quad 1,\ 1,\ 1,\ 4,\ 4,\ 4\ ) \\
q_2 & = & (\ 0,\ 0,\ 2,\text{-}1,\ 0,\ 0,\ 0,\ 0,\quad 4,\ 0,\ 4,\ 4,\quad 1,\ 4,\ 4,\ 1,\ 1,\ 4\ ) \\
q_2' & = & (\ 0,\ 0,\text{-}1,\ 2,\ 0,\ 0,\ 0,\ 0,\quad 4,\ 0,\ 4,\ 4,\quad 1,\ 4,\ 4,\ 1,\ 1,\ 4\ ) \\
q_3 & = & (\ 0,\ 0,\ 0,\ 0,\ 2,\text{-}1,\ 0,\ 0,\quad 4,\ 4,\ 0,\ 4,\quad 4,\ 1,\ 4,\ 1,\ 4,\ 1\ ) \\
q_3' & = & (\ 0,\ 0,\ 0,\ 0,\text{-}1,\ 2,\ 0,\ 0,\quad 4,\ 4,\ 0,\ 4,\quad 4,\ 1,\ 4,\ 1,\ 4,\ 1\ ) \\
q_4 & = & (\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 2,\text{-}1,\quad 4,\ 4,\ 4,\ 0,\quad 4,\ 4,\ 1,\ 4,\ 1,\ 1\ ) \\
q_4' & = & (\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\text{-}1,\ 2,\quad 4,\ 4,\ 4,\ 0,\quad 4,\ 4,\ 1,\ 4,\ 1,\ 1\ )
\end{array}
$$

Fig. 3. A dataset D in which a 4-skyline group $\{p_1,p_2,p_3,p_4\}$ contains no 3-skyline group.

As a minor optimization, we sort $\mathcal{T}_k(D)$ according to a monotone preference function based on entropy [5] so that p_j never dominates p_i if p_j is considered later than p_i. Hence GIncremental considers "good" tuples early and is likely to generate "good" skyline groups dominating many candidate groups early in the computation.

The correctness of the algorithm GIncremental is subject to the conjecture that every k-skyline group contains at least one $(k-1)$-skyline group. Proposition 2.6 implies that the conjecture is true for $k = 2$. Proposition 3.2 below shows that the conjecture is also true for $k = 3$ if the aggregate function f is restricted to sum or average. The proof of Proposition 3.2 allows multi-sets (*e.g.*, $\{p,p,r\}$) and implicitly uses Lemma 3.1, which also allows multi-sets and uses $\cup$ to denote the union of multi-sets, for example, $\{p,p\} \cup \{p,q\} = \{p,p,p,q\}$. Note that both proofs of Lemma 3.1 and Proposition 3.2 are valid regardless of the dimensionality of the dataset.

Lemma 3.1. *Suppose that the aggregate function f is sum or average. Then,*

1) *if $C_i \succ C_i'$ and $C_j \succ C_j'$, then $C_i \cup C_j \succ C_i' \cup C_j'$.*
2) *if $C_i \succ C_i'$ with $p \in C_i$ and $p \in C_i'$, then $C_i - \{p\} \succ C_i' - \{p\}$.*

Proof. The proof is straightforward by Definition 2.2 and the assumption that f is sum or average. □

Proposition 3.2. *Suppose that the aggregate function f is sum or average. Then every 3-skyline group G_3 contains a 2-skyline group G_2 (i.e., $G_2 \subset G_3$).*

Proof. See Appendix for the detailed proof. □

Unfortunately the conjecture fails to extend to the case of $k \geqslant 4$.

Proposition 3.3. *There exists a dataset D with a 4-skyline group $G_4 \in \mathcal{G}_4(D)$ such that no 3-skyline group is a subset of G_4, i.e., $G_3 \in \mathcal{G}_3(D)$ implies $G_3 \not\subset G_4$.*

Proof. Consider the summation Σ as an aggregate function. Fig. 3 shows such a dataset D (with dimensionality 18 and cardinality 12). $G_4 = \{p_1,p_2,p_3,p_4\}$ is a 4-skyline group, but no subset of size 3 is a 3-skyline group. For the detailed proof, see Appendix. □

Proposition 3.3 implies that for $k \geqslant 4$, the algorithm GIncremental is incorrect in general: it may fail to identify all k-skyline groups. In fact, we observe its incorrectness even on a real dataset. For example, on the three-dimensional NBA dataset (which we use in Section 5), GIncremental misses a 9-skyline group out of a total of 3635 9-skyline groups. Still, however, we regard GIncremental as a practical group skyline algorithm because it usually returns a correct answer unless k is extremely large (*e.g.*, $k > 8$). Even if it fails to return a correct answer, it misses only a very small number of k-skyline groups.

4. Dynamic group skyline computation

Besides its incorrectness, the algorithm GIncremental has another weakness that it generates only one candidate group at a time. Our experiments with GIncremental, however, reveal that it is safe to generate a set of candidate groups at once as far as the set is smaller than the current group skyline.

Below we develop another group skyline algorithm GDynamic which is an improvement over the algorithm GIncremental. Like GIncremental, GDynamic successively computes group skylines of different sizes, but selects a specific tuple p and generates all candidate groups containing p at once. The availability of such a set of candidate groups allows us to implement an optimization that would be too expensive if separately applied to individual candidate groups as in GIncremental. GDynamic is equivalent to a dynamic algorithm that fills a table of group skylines (of different sizes and with different input datasets), but its implementation updates just an array of group skylines.

4.1. Basic idea

The basic idea of the algorithm GDynamic is that given a dataset D containing a tuple p, we can compute $\mathcal{G}_k(D)$ from $\mathcal{G}_k(D-\{p\})$ and $\mathcal{G}_{k-1}(D-\{p\})$ (where $k>1$). Consider a k-skyline group G_k. There are two possibilities:

- If $p \notin G_k$, then G_k must be in $\mathcal{G}_k(D-\{p\})$.
- If $p \in G_k$, then $G_k-\{p\}$ must be in $\mathcal{G}_{k-1}(D-\{p\})$. Otherwise there exists a $(k-1)$-group G_{k-1} such that $G_{k-1} \succ G_k-\{p\}$, which implies $G_{k-1}\cup\{p\} \succ (G_k-\{p\})\cup\{p\}$, *i.e.*, $G_{k-1}\cup\{p\} \succ G_k$, contradicting the assumption that G_k is a k-skyline group.

Therefore, in order to compute $\mathcal{G}_k(D)$, we only have to consider those k-groups in $\mathcal{G}_k(D-\{p\})$ and new candidate k-groups built from $\mathcal{G}_{k-1}(D-\{p\})$:

$$\mathcal{G}_k(D) = \mathcal{S}(\mathcal{G}_k(D-\{p\}) \ \cup \ \{G_{k-1}\cup\{p\} \mid G_{k-1}\in\mathcal{G}_{k-1}(D-\{p\})\})$$

Note that new candidate k-groups built from $\mathcal{G}_{k-1}(D-\{p\})$ are all incomparable with each other. In this way, we try to generate a minimal number of intermediate candidate groups.

Building on this basic idea, GDynamic exploits Corollary 2.7 to examine only those tuples in $\mathcal{T}_k(D)$ and Proposition 2.6 to further reduce the number of intermediate candidate groups. Let us assume $\mathcal{T}_k(D)=\{p_1,\cdots,p_n\}$. We use $\mathcal{G}_i^j$ for the i-skyline of a dataset $\{p_1,\ldots,p_j\}$. For $i>1$, we compute $\mathcal{G}_i^j$ from $\mathcal{G}_i^{j-1}$ and $\mathcal{G}_{i-1}^{j-1}$ as follows:

$$\mathcal{G}_i^j = \mathcal{S}\Big(\mathcal{G}_i^{j-1}\cup\Big\{G_{i-1}\cup\{p_j\} \mid G_{i-1}\in\mathcal{G}_{i-1}^{j-1},\ q\succ p_j \text{ implies } q\in G_{i-1}\Big\}\Big)$$

For $i=1$, we compute $\mathcal{G}_i^j$ by inserting $\{p_j\}$ to $\mathcal{G}_i^{j-1}$ if no (singleton) group in $\mathcal{G}_i^{j-1}$ dominates $\{p_j\}$:

$$\mathcal{G}_i^j = \mathcal{S}\Big(\mathcal{G}_i^{j-1}\cup\{\{p_j\}\}\Big)$$

Thus, after initializing $\mathcal{G}_i^0$ to an empty set for $i=1,\ldots,k$, we compute $\mathcal{G}_i^j$ for $j=1,\ldots,n$ until we obtain $\mathcal{G}_k^n$. We may think of computing $\mathcal{G}_i^j$ by GDynamic as filling in a table of group skylines shown below:

	1	$\cdots$	$i-1$	i	$\cdots$	k
1	$\{p_1\}$		$\varnothing$	$\varnothing$		$\varnothing$
$\vdots$						
$j-1$			$\mathcal{G}_{i-1}^{j-1}$	$\mathcal{G}_i^{j-1}$		
j				$\searrow \downarrow$ $\mathcal{G}_i^j$		
$\vdots$						
n	?		?	?		$\mathcal{G}_k^n$

4.2. Group skyline algorithm GDynamic

Fig. 4 shows the pseudocode of the algorithm GDynamic. As in GIncremental, it begins by sorting candidate tuples in $\mathcal{T}_k(D)$ according to a monotone preference function based on entropy (line 1). The outer loop (lines 3–12) considers a tuple p_j in $\mathcal{T}_k(D)$ at each iteration and computes all group skylines (of sizes $1,\ldots,k$) of $\{p_1,\ldots,p_j\}$. The inner loop (lines 4–11) decreases the index variable i which specifies the size of the group skyline being computed. For $i>1$, the inner loop generates at once the set $\mathcal{C}_i$ of all candidate i-groups that include p_j (line 8) and incorporates $\mathcal{C}_i$ into an existing group skyline $\mathcal{G}_i$ (line 9). Note that decreasing i in the inner loop permits GDynamic to discard $\mathcal{G}_i^{j-1}$ after computing $\mathcal{G}_i^j$ and thus allocate just a single variable $\mathcal{G}_i$ for the entire i-th column in the table shown above.

Algorithm GDynamic (D, k)
Input: dataset D
size k for the group skyline
Output: k-skyline $\mathcal{G}_k(D)$
1: compute $\mathcal{T}_k(D) = \{p_1, \cdots, p_n\}$ sorted by entropy
2: $\mathcal{G}_i \leftarrow \varnothing$, $i = 1, \cdots, k$
3: **for** j **from** 1 **to** n **do**
4: **for** i **from** $\min(j, k)$ **down to** $\max(1, j - n + k)$ **do**
// $\mathcal{G}_{i-1} = \mathcal{G}_{i-1}^{j-1}$ and $\mathcal{G}_i = \mathcal{G}_i^{j-1}$
5: **if** $i = 1$ **then**
6: $\mathcal{G}_i \leftarrow \mathcal{S}(\mathcal{G}_i \cup \{\{p_j\}\})$
7: **else**
8: $\mathcal{C}_i \leftarrow \{G_{i-1} \cup \{p_j\} \mid G_{i-1} \in \mathcal{G}_{i-1},$
G_{i-1} contains every tuple that dominates $p_j\}$
9: $\mathcal{G}_i \leftarrow \mathcal{S}(\mathcal{G}_i \cup \mathcal{C}_i)$
10: **end if** // $\mathcal{G}_i = \mathcal{G}_i^j$
11: **end for**
12: **end for**
13: **return** $\mathcal{G}_k$

Fig. 4. Algorithm GDynamic.

	$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{G}_3$
p_1	$\{p_1\}$	$\mathcal{G}_2^1$ $\varnothing$	$\mathcal{G}_3^1$ $\varnothing$
p_2	$\{p_1\}$ $\{p_2\}$	$\mathcal{G}_2^2$ $\{p_1, p_2\}$ $(3, 7)$	$\mathcal{G}_3^2$ $\varnothing$
p_3	$\{p_1\}$ $\{p_2\}$ $\{p_3\}$	$\mathcal{G}_2^3$ $\{p_1, p_2\}$ $(3, 7)$ $\{p_1, p_3\}$ $(7, 4)$ $\{p_2, p_3\}$ $(4, 5)$ $\times$	$\mathcal{G}_3^3$ $\{p_1, p_2, p_3\}$ $(7, 8)$
p_4	?	$\mathcal{G}_2^4$ $\{p_1, p_2\}$ $(3, 7)$ $\{p_1, p_3\}$ $(7, 4)$ $\{p_1, p_4\}$ $(5, 6)$	$\mathcal{G}_3^4$ $\{p_1, p_2, p_3\}$ $(7, 8)$ $\{p_1, p_2, p_4\}$ $(5, 10)$ $\{p_1, p_3, p_4\}$ $(9, 7)$
p_5	?	?	$\mathcal{G}_3^5$ $\{p_1, p_2, p_3\}$ $(7, 8)$ $\{p_1, p_2, p_4\}$ $(5, 10)$ $\{p_1, p_3, p_4\}$ $(9, 7)$ $\{p_1, p_4, p_5\}$ $(7, 8)$

Fig. 5. Example of running GDynamic on the dataset of basketball players in Fig. 1.

Fig. 5 shows the result of running GDynamic on the dataset D in Fig. 1 to compute $\mathcal{G}_3(D)$ using the aggregate function sum. By sorting candidate tuples in $\mathcal{T}_3(D)$ by entropy, we obtain $\{p_1, p_2, p_3, p_4, p_5\}$ (which excludes player p_6).[1] Throughout the entire computation, GDynamic generates only one candidate group that is later eliminated, namely $\{p_2, p_3\}$ (marked with $\times$) which is included in $\mathcal{G}_2^3$, but not in $\mathcal{G}_2^4$. All other candidate groups turn out to be skyline groups. Note that GDynamic exploits Proposition 2.6 when generating candidate groups. For example, when computing $\mathcal{G}_3^5$, it does not generate $\{p_1, p_2, p_5\}$ and $\{p_1, p_3, p_5\}$ as candidate groups because neither group contains player p_4, who dominates player p_5.

To show the correctness of GDynamic, we observe that at the beginning of an iteration of the inner loop, variables $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$ store $(i-1)$-skyline $\mathcal{G}_{i-1}^{j-1}$ and i-skyline $\mathcal{G}_i^{j-1}$, respectively (line 4). Then the inner loop terminates the iteration by storing i-skyline $\mathcal{G}_i^j$ in variable $\mathcal{G}_i$ (line 10). Hence, when the outer loop terminates, variable $\mathcal{G}_k$ stores the k-skyline of $\mathcal{T}_k(D)$, or equivalently, of the input dataset D.

In addition to its correctness, GDynamic overcomes the weakness of GIncremental by generating at once the set of all candidate groups that include a specific tuple (line 8 in Fig. 4). The availability of a set of candidate groups allows us to implement an optimization that would be too expensive if separately applied to individual candidate groups as in GIncremental. For example, to update $\mathcal{G}_i$ in line 9 in Fig. 4, our implementation of GDynamic maintains temporary index structures (storing the result of sorting tuples along each individual dimension) for both $\mathcal{G}_i$ and $\mathcal{C}_i$. Then, for each i-skyline group G_i in $\mathcal{G}_i$,

[1] Players p_2 and p_3 have the same entropy because we take into consideration maximum and minimum values for each dimension.

we exploit the index structure for $\mathcal{C}_i$ in order to determine the minimal number of i-groups in $\mathcal{C}_i$ that should be compared with G_i. (Similarly for each i-group C_i in $\mathcal{C}_i$.)

The following example illustrates how our implementation of GDynamic exploits temporary index structures, where we use the aggregate function sum. It concerns the last step in Fig. 5 which merges two sets $\mathcal{C}_3$ and $\mathcal{G}_3^4$ to obtain $\mathcal{G}_3^5$ (in line 9 in Fig. 4).

	$\mathcal{C}_3$
C_3^1	$\{p_1,p_2,p_5\}$ (5,9)
C_3^2	$\{p_1,p_3,p_5\}$ (9,6)
C_3^3	$\{p_1,p_4,p_5\}$ (7,8)

	$\mathcal{C}_3$ index
1	2
C_3^2	C_3^1
C_3^3	C_3^3
C_3^1	C_3^2

	$\mathcal{G}_3^4$
G_3^1	$\{p_1,p_2,p_3\}$ (7,8)
G_3^2	$\{p_1,p_2,p_4\}$ (5,10)
G_3^3	$\{p_1,p_3,p_4\}$ (9,7)

	$\mathcal{G}_3^4$ index
1	2
G_3^3	G_3^2
G_3^1	G_3^1
G_3^2	G_3^3

GDynamic obtains $\mathcal{C}_3$ by adzding a new tuple p_5 into each 2-skyline group in $\mathcal{G}_2^4$ (in line 8 in Fig. 4). $\mathcal{G}_3^4$ is the 3-skyline of a smaller subset $\{p_1,p_2,p_3,p_4\}$ of the dataset. For each of $\mathcal{C}_3$ and $\mathcal{G}_3^4$, our implementation of GDynamic maintains a sorted list for each dimension as an index structure. For example, the first sorted list for $\mathcal{C}_3$ stores indexes to 3-groups in $\mathcal{C}_3$ in the decreasing order of their aggregate values (sums) of elements of the first dimension.

To obtain $\mathcal{G}_3^5$ from $\mathcal{G}_3^4$ and $\mathcal{C}_3$, we use each 3-group G_3 in $\mathcal{G}_3^4$ to remove from $\mathcal{C}_3$ all groups dominated by G_3, and vice versa. Suppose that we now examine G_3^2 in $\mathcal{G}_3^4$. We perform a binary search on each sorted list for $\mathcal{C}_3$ to find the minimum number of groups to be compared. By inspecting the first sorted list, we find that G_3^2 dominates at most one group, namely C_3^1. Note that G_3^2 cannot dominate C_3^2 and C_3^3 because its aggregate value for the first dimension is smaller than the corresponding values of C_3^2 and C_3^3. Similarly, by inspecting the second list, we find that G_3^2 dominates at most three groups C_3^1, C_3^2, and C_3^3. By combining these two results, G_3^2 dominates at most one group C_3^1. Hence GDynamic compares G_3^2 only with C_3^1 and dispenses with two dominance tests. When using each group C_3 in $\mathcal{C}_3$ to remove from $\mathcal{G}_3^4$ all groups dominated by C_3, GDynamic similarly exploits the index structures for $\mathcal{G}_3^4$.

In conjunction with the basic idea given in Section 4.1, the above optimization results in a considerable difference in speed and the number of dominance tests, as we will see in the next section.

5. Experiments

This section presents experimental results of running GIncremental and GDynamic on various datasets. In order to measure their relative performance, we use another group skyline algorithm GNaive:

- Given a dataset D, GNaive computes $\mathcal{G}_k(D)$ by exhaustively generating all possible candidate k-groups from $\mathcal{T}_k(D)$ and updating the set of k-skyline groups in a progressive manner. It does not compute the k-skyline at once after generating all candidate k-groups, which may not fit in memory. Rather it generates a k-group at a time and compares it with current k-skyline groups to update the k-skyline. When generating candidate k-groups, GNaive also exploits Proposition 2.6 and Corollary 2.7, and examines tuples in $\mathcal{T}_k(D)$ in the decreasing order of their entropy.

In all our experiments below, we use an aggregate function sum, and GIncremental misses no skyline groups and returns correct answers. We implement all the algorithms GIncremental, GDynamic, and GNaive as main-memory algorithms in C.

We do not compare our algorithms with RCA, the algorithm for computing top-k combinatorial skylines, proposed by Su et al. [22]. The reasons are twofold. First RCA requires the user to determine the ranking of every dimension of the dataset, and its performance varies depending on the user preference. Second, while our algorithms compare only groups of the same size, RCA compares groups of different sizes as well.

5.1. Setup

The experiments use both synthetic datasets and real datasets. We generate synthetic datasets according to independent distributions or anti-correlated distributions [4]. For independent datasets, we independently generate all tuple elements using a uniform distribution. For anti-correlated datasets, we generate tuples in such a way that a good tuple element in one dimension is likely to indicate the existence of bad elements in other dimensions. For real datasets, we use the NBA dataset which contains average records about 1,313 players from 1991 to 2005 (from http://www.nba.com). The NBA dataset roughly follows an anti-correlated distribution.

A dataset D determines two parameters d and n:

- d denotes the dimensionality of the dataset.
- n denotes the number of tuples in the dataset, or its size. When reporting the number of tuples, we use K for 1000.

For independent distributions, we use two groups of synthetic datasets. The first group varies the number of tuples in the dataset: we use $n = 1K$, $2K$, $8K$, $64K$, and $d = 4$. The second group varies the dimensionality of the dataset: we use $d = 2$, 3, 4, 5, and $n = 1K$. For anti-correlated distributions, we use $n = 50$, 100, 200, 400, and $d = 4$. (Thus we use a total of 11 synthetic datasets.) In each experiment, we use a parameter k to specify the size of group skylines.

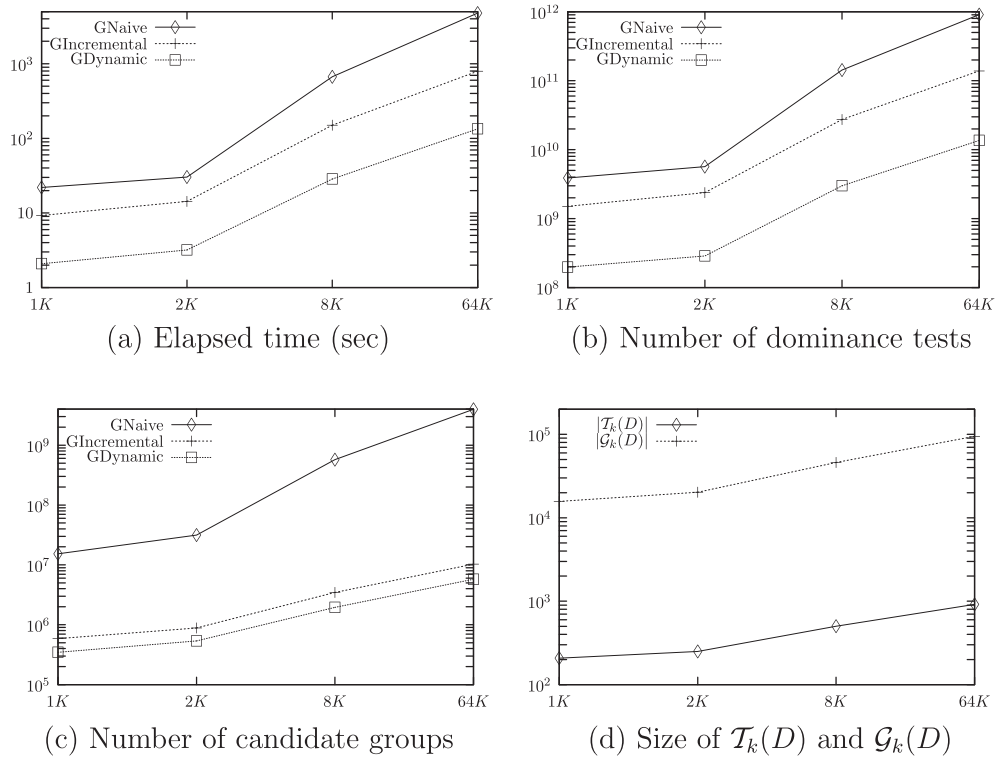

Fig. 6. Results on independent datasets with $d = 4$, $k = 5$ ($n = 1K, 2K, 8K, 64K$).

We compile all the algorithms using gcc 4.4.3 with -O3 option. We run all experiments on Ubuntu Linux with Intel Xeon X3363 2.83 GHz and 4 gigabytes of main memory. We use as main performance metrics the elapsed time measured in wall clock seconds, the number of dominance tests, and the number of intermediate candidate groups. All measurements are averaged over 4 sample runs. For each experiment, we also report the size of $\mathcal{T}_k(D)$ and $\mathcal{G}_k(D)$ of each dataset D.

5.2. Experimental results on synthetic datasets

Fig. 6 shows the effect of the size of independent datasets on (a) the elapsed time, (b) the number of dominance tests, and (c) the number of intermediate candidate groups when n varies from 1K to 64K while d is set to 4; we set k (the size of group skylines) to 5. All graphs are shown in a logarithmic scale and have a similar pattern. For each algorithm, the elapsed time is approximately proportional to the total number of dominance tests. Fig. 6(d) shows the size of $\mathcal{T}_k(D)$ and $\mathcal{G}_k(D)$ for each dataset D. Note that because of the use of independent distributions in generating datasets, $|\mathcal{T}_k(D)|$ and $|\mathcal{G}_k(D)|$ do not increase in proportion to n. For example, while n increases 8-fold from 8K to 64K, both $|\mathcal{T}_k(D)|$ and $|\mathcal{G}_k(D)|$ increase approximately 2-fold only.

In Fig. 6, GDynamic outperforms GNaive and GIncremental with respect to all three metrics. For example, in Fig. 6(a), it runs up to 35.59 and 5.93 times faster than GNaive and GIncremental, respectively (n = 64K). In Fig. 6(b), GDynamic significantly reduces the number of dominance tests: 96.56% of GNaive and 88.53% of GIncremental on average. In Fig. 6(c), it also significantly reduces the number of intermediate candidate groups: 98.12% of GNaive and 42.05% of GIncremental on average. Overall GDynamic is much more effective than GNaive and GIncremental in generating candidate groups and avoiding unnecessary dominance tests.

Fig. 7 shows the effect of the dimensionality of independent datasets when d varies from 2 to 5 while n is set to 1K; we set k to 5. All graphs are shown in a logarithmic scale. As in Fig. 6, GDynamic outperforms GNaive and GIncremental with respect to all three metrics. In Fig. 7(a), it runs up to 10.52 and 4.42 times faster than GNaive and GIncremental, respectively (d = 4). Fig. 7(b) and (c) show that GDynamic significantly reduces the number of dominance tests and intermediate candidate groups, respectively: for dominance tests, 94.91% of GNaive and 86.60 % of GIncremental on average (excluding d = 2); for candidate groups, 95.00% of GNaive and 40.10% of GIncremental on average (excluding d = 2). In Fig. 7(d), since a larger value of d means more skyline tuples, both $|\mathcal{T}_k(D)|$ and $|\mathcal{G}_k(D)|$ increase with d, but $|\mathcal{G}_k(D)|$ increases much faster than $|\mathcal{T}_k(D)|$. That is, the ratio $\frac{|\mathcal{G}_k(D)|}{|\mathcal{T}_k(D)|}$ increases with d. For example, $\frac{|\mathcal{G}_k(D)|}{|\mathcal{T}_k(D)|}$ is 2.35 for d = 2, but 270.47 for d = 5. This result indicates that combinations of candidate tuples (from $\mathcal{T}_k(D)$) are more likely to form skyline groups in high-dimensional datasets.

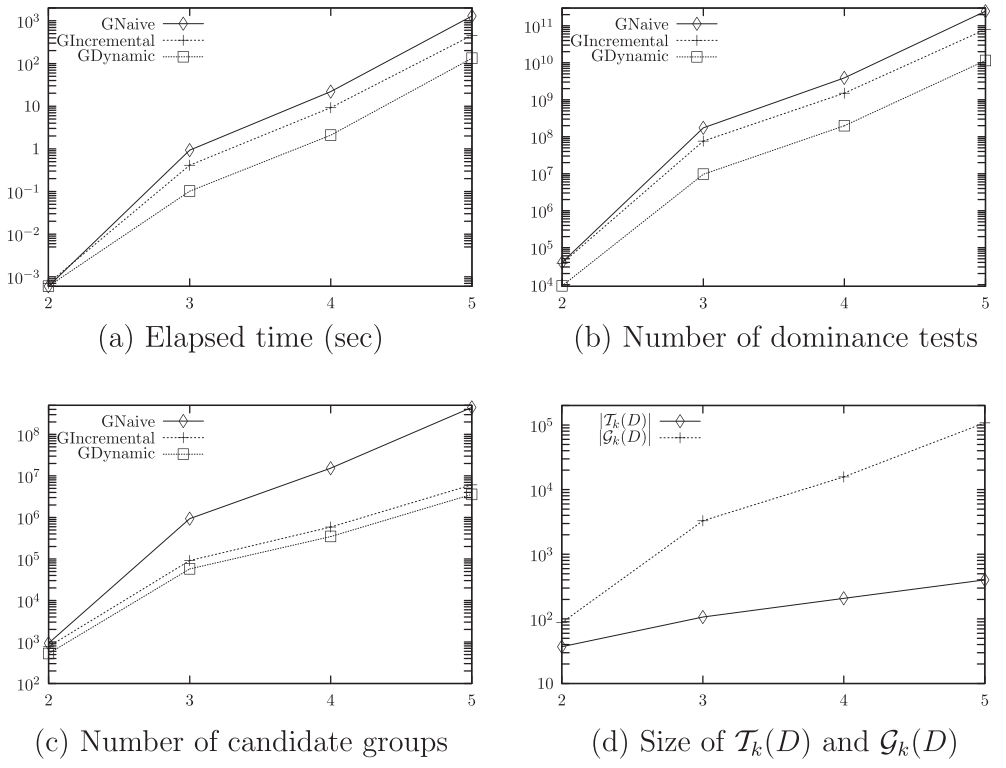

Fig. 7. Results on independent datasets with n = 1K, k = 5 (d = 2,3,4,5).

Fig. 8 shows the effect of the size of anti-correlated datasets when n varies from 50 to 400 while d is set to 4; we set k to 5. All graphs are shown in a logarithmic scale. Although we observe a similar pattern, the speedup of GDynamic over GIncremental is not as high as in Figs. 6 and 7: it runs only from 1.14 ($n = 400$) to 3.22 ($n = 50$) times faster. This is partly because combinations of candidate tuples in anti-correlated datasets are more likely to form aggregate tuples whose values are close to the median values for individual dimensions than in independent datasets. Hence the optimization in GDynamic is not so effective in anti-correlated datasets – when combining two sets of intermediate candidate groups (line 9 in Fig. 4), GDynamic performs more dominance tests in anti-correlated datasets than in independent datasets. GDynamic still reduces 66.60% of dominance tests and 18.83% of candidate groups on average when compared with GIncremental. In Fig. 8(d), the ratio $\frac{|\mathcal{G}_k(D)|}{|\mathcal{T}_k(D)|}$ ranges from 312.59 ($n = 200$) to 392.04 ($n = 50$), whereas it ranges only from 75.60 ($n = 1K$) to 103.94 ($n = 8K$) in Fig. 6(d). This result indicates that combinations of candidate tuples are more likely to form skyline groups in anti-correlated datasets than in independent datasets.

Fig. 9 shows the effect of the size of groups when k varies from 2 to 8; we consider the independent dataset with $n = 1K$ and $d = 4$. All graphs are shown in a logarithmic scale. We observe that even for a large value of k, GDynamic is much faster than GNaive and GIncremental and the most effective in generating candidate groups and avoiding unnecessary dominance tests. Note that $|\mathcal{G}_k(D)|$ increases exponentially with k (because the number of k-groups increases exponentially with k).

5.3. Effectiveness of Proposition 2.6

To show the pruning power of Proposition 2.6, we compare GDynamic with a variant that does not exploit it. For the variant, we simply remove the condition "G_{i-1} contains every tuple that dominates p_j" in line 8 in Fig. 4. We use the same datasets as in Section 5.2.

Fig. 10 shows the reduction percentage of dominance tests and candidate groups due to Proposition 2.6. Fig. 10(a) shows that the reduction percentage is more than 70% for all datasets, regardless of their size. Fig. 10(b) shows that the reduction percentage for independent datasets is almost irrelevant to their dimensionality (up to 5). Fig. 10(c) shows that the reduction percentage for anti-correlated datasets increases with n. Since anti-correlated datasets usually have more skyline tuples than independent datasets, the reduction percentage is not as high as in independent datasets. Fig. 10(d) shows that the reduction percentage increases with k. As the number of k-groups increases exponentially with k, the number of dominance tests and candidate groups reduced by Proposition 2.6 also increases exponentially with k. Overall Proposition 2.6 is highly effective in reducing the number of dominance tests and intermediate candidate groups.

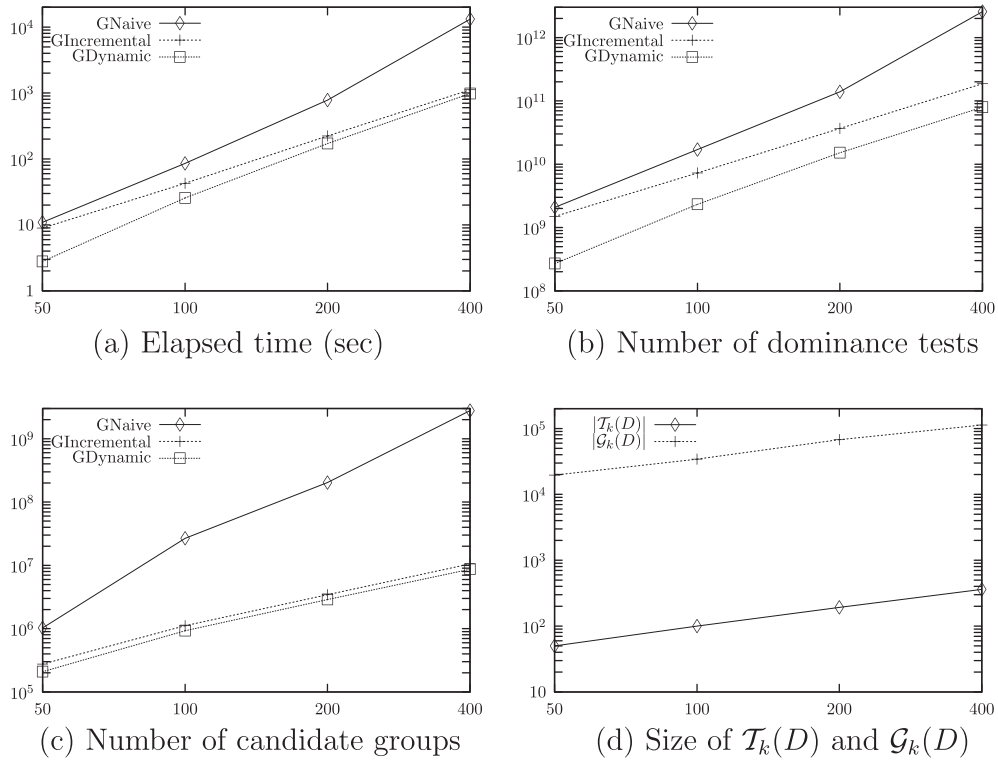

Fig. 8. Results on anti-correlated datasets with $d = 4$, $k = 5$ ($n = 50, 100, 200, 400$).

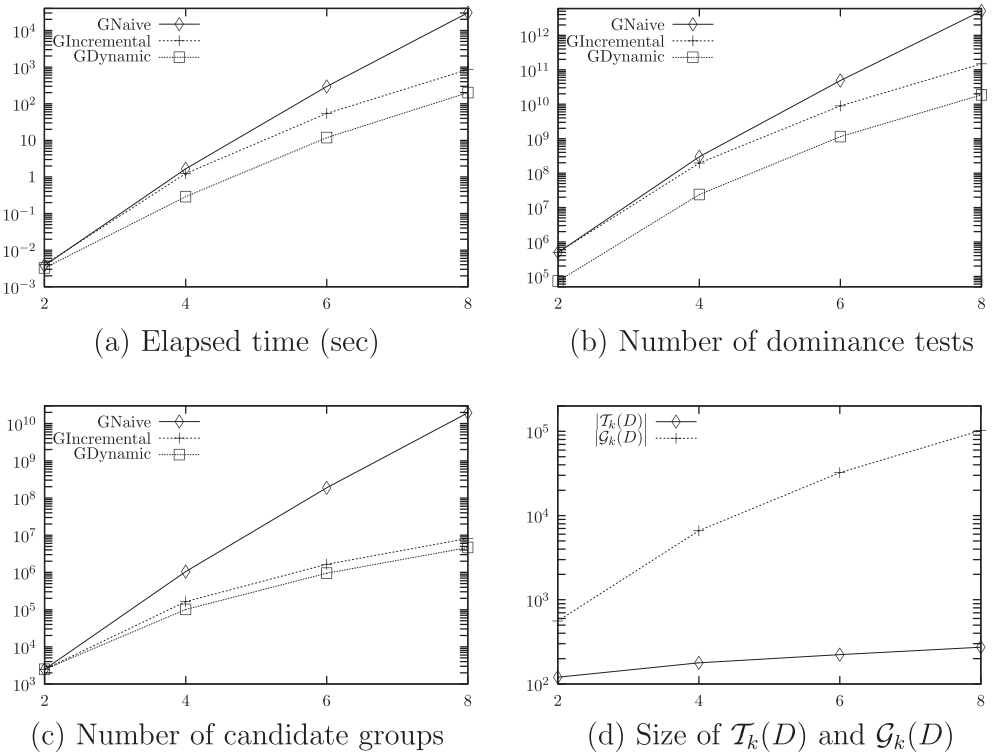

Fig. 9. Results on the independent dataset with $n = 1K$, $d = 4$ ($k = 2,4,6,8$).

n	dominance	groups
$1K$	71.33	71.47
$2K$	70.96	72.52
$8K$	70.97	74.02
$64K$	77.35	78.26
average	72.65	74.07

(a) independent, $d = 4$, $k = 5$

d	dominance	groups
2	53.78	72.93
3	69.31	68.09
4	71.33	71.47
5	68.20	69.69
average	65.66	70.55

(b) independent, $n = 1K$, $k = 5$

n	dominance	groups
50	22.01	21.68
100	25.64	25.67
200	44.11	45.24
400	49.59	50.17
average	35.34	35.69

(c) anti-correlated, $d = 4$, $k = 5$

k	dominance	groups
2	37.72	48.77
4	65.74	66.01
6	73.06	72.63
8	78.68	77.59
average	63.80	66.25

(d) independent, $n = 1K$, $d = 4$

Fig. 10. Reduction percentage (%) of dominance tests and candidate groups due to Proposition 2.6.

5.4. Experimental results on real datasets

This section shows the experimental results on the NBA dataset with two dimensionalities $d = 3, 5$. For $d = 3$, we use average points, assists, and rebounds for each player. For $d = 5$, we additionally use average steals and blocks.

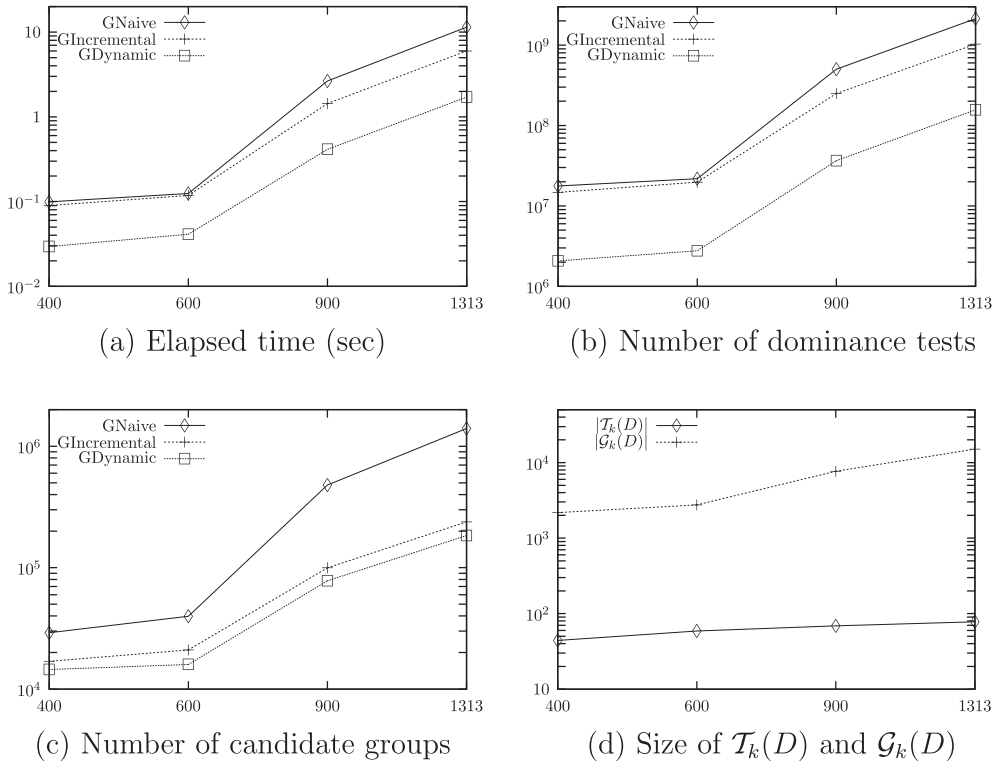

Fig. 11. Results on the NBA dataset with $d = 5$, $k = 5$ ($n = 400, 600, 900, 1313$).

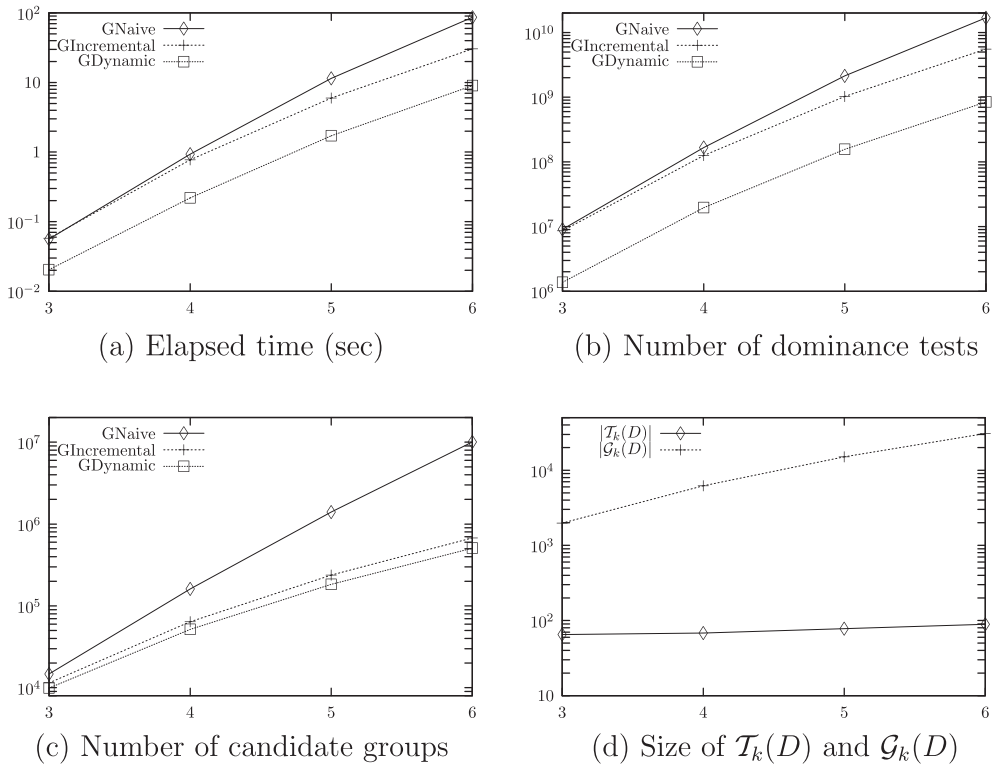

Fig. 12. Results on the NBA dataset with $n = 1313$, $d = 5$ ($k = 3, 4, 5, 6$).

d	GNaive	GIncremental	GDynamic	$\lvert\mathcal{T}_k(D)\rvert$	$\lvert\mathcal{G}_k(D)\rvert$
3	0.035	0.025	0.010	46	982
5	11.449	5.995	1.711	78	15071

(a) Elapsed time (sec) and size of $\mathcal{T}_k(D)$ and $\mathcal{G}_k(D)$

d	GNaive	GIncremental	GDynamic
3	6623797	4190388	797011
5	2130476306	1028816926	156831364

(b) Number of dominance tests

d	GNaive	GIncremental	GDynamic
3	23610	10392	8632
5	1398934	238326	183579

(c) Number of candidate groups

Fig. 13. Results on the NBA dataset with $n = 1313$, $k = 5$ ($d = 3,5$).

Fig. 11 shows the effect of the size of the NBA dataset: we consider $n = 400, 600, 900, 1313$ while setting both d and k to 5. We create datasets with $n < 1313$ by randomly choosing tuples from the NBA dataset with $n = 1313$. We observe a similar pattern to Fig. 6. In Fig. 11(a), GDynamic runs up to 9.61 times faster than GNaive and up to 3.50 times faster than GIncremental ($n = 1313$). In Fig. 11(b), it significantly reduces the number of dominance tests: 90.25% of GNaive and 85.51% of GIncremental on average. In Fig. 11(c), it also significantly reduces the number of intermediate candidate groups: 70.20% of GNaive and 20.83% of GIncremental on average. The ratio $\frac{|\mathcal{G}_k(D)|}{|\mathcal{T}_k(D)|}$ ranges from 46.58 ($n = 600$) to 193.22 ($n = 1313$).

Fig. 12 shows the effect of the size of groups when k varies from 3 to 6; we consider the NBA dataset with $n = 1313$ and $d = 5$. We observe a similar pattern to Fig. 9. The speedup of GDynamic over GNaive gradually increases with k, reaching 9.61 for $k = 6$. Its speedup over GIncremental also gradually increases with k and it runs 3.32 times faster than GIncremental on average. Furthermore it performs much fewer dominance tests and examines much fewer candidate groups than GNaive and GIncremental.

Fig. 13 shows the experimental results on the NBA dataset with $d = 3, 5$ when we set n to 1313 and k to 5. As in all the previous experiments, GDynamic outperforms both GNaive and GIncremental with respect to the elapsed time, the number of dominance tests, and the number of intermediate candidate groups.

To conclude, GDynamic is much more effective than GNaive and GIncremental for both synthetic and real datasets in generating candidate groups and avoiding unnecessary dominance tests. The next subsection gives an application of group skyline computation on the NBA dataset.

5.5. Analysis of the NBA dataset

As an example of decision making based on group skyline computation, let us consider the 3-dimensional NBA dataset which records average points, assists, and rebounds for each player. We are interested in recruiting five good players to form a team. We compute the 5-skyline of the dataset and obtain a total of 982 5-skyline groups. We rank each player by the number of 5-skyline groups to which he belongs, and obtain the following five players with the highest numbers:

Player	Average record	Number
LeBron James	(26.72, 6.53, 6.70)	676
Jason Kidd	(14.59, 9.18, 6.54)	635
Charles Barkley	(21.02, 4.18, 11.70)	376
Kevin Garnett	(20.35, 4.51, 11.25)	356
John Stockton	(13.51, 10.24, 2.88)	310

The five players are all skyline players and also form a 5-skyline group by themselves with a total average record of (96.19, 34.64, 39.06).

Now we wish to form another team to play against the previous team. We find a total of six teams that are incomparable with the previous team:

Team	Players
1	p_1, p_2, p_3, p_4, p_5
2	p_1, p_2, p_3, p_6, p_5
3	p_1, p_7, p_2, p_3, p_4
4	p_1, p_7, p_2, p_3, p_6
5	p_1, p_8, p_7, p_2, p_6
6	p_1, p_8, p_7, p_2, p_3

Player	Name
p_1	Shaquille O'Neal
p_2	Karl Malone
p_3	Tim Duncan
p_4	Dirk Nowitzki
p_5	Dennis Rodman
p_6	Chris Webber
p_7	Michael Jordan
p_8	Allen Iverson

We have to recruit Shaquille O'Neal who is the only player belonging to all the six teams. Suppose that Dennis Rodman, Chris Webber, and Allen Iverson refuse to join the new team. Then we choose the team 3 whose total average record is (123.07, 17.08, 48.41). Interestingly Dirk Nowitzki is not a skyline player as his average record (22.17, 2.54, 8.66) is dominated by Karl Malone's record (24.75, 3.99, 9.81). (Including Dirk Nowitzki, there are 17 such non-skyline players who belong to 5-skyline groups.)

6. Related work

In this section, we review conventional skyline algorithms and their connection with group skyline computation. Group-by skyline computation in [16] returns skyline sets of multiple groups of tuples and is fundamentally different from group skyline computation.

Depending on whether external index structures are used or not, the state-of-the-art skyline algorithms are usually classified into two categories. Those skyline algorithm that do no use external index structures include the block-nested-loop (BNL) algorithm [4] which performs a dominance test between every pair of tuples while maintaining a window of candidate skyline tuples, the sort-filter-skyline (SFS) algorithm [5] which presorts a dataset according to a monotone preference function so that no tuple is dominated by succeeding tuples in the sorted dataset, LESS (linear elimination sort for skyline) [9] which incorporates two optimization techniques into the external sorting phase of SFS, and SaLSa (sort and limit skyline algorithm) [2] which presorts a dataset, but unlike SFS and LESS, does not necessarily inspect every tuple in the sorted dataset. All these algorithms do not take streams of tuples as input and thus are difficult to adapt for group skyline computation, which needs to generate all candidate groups in a progressive manner.

There are also several skyline algorithms that exploit external index structures. The nearest-neighbor (NN) algorithm [12] and the branch-and-bound skyline (BBS) algorithm [18] use R-trees [10] as their index structures. Lee et al. [13] propose the ZSearch algorithm which uses a new variant of B^+-tree, called *ZBtree*, for maintaining the set of candidate skyline tuples in the increasing order of Z-addresses. Similarly to those algorithms using no external index structures, BBS and ZSearch are inappropriate for group skyline computation because they require an entire dataset in order to build an external index structure.

Since group skyline computation generates candidate groups in a progressive manner, we can incorporate into our group skyline algorithms any algorithm for continuous skyline computation on data streams such as [14,17,23]. For example, we can replace $\mathcal{G}_i \leftarrow \mathcal{S}(\mathcal{G}_i \cup \mathcal{C}_i)$ in line 9 in Fig. 4 with an algorithm for continuous skyline computation that regards $\mathcal{C}_i$ as new arrivals in the data stream. Such an algorithm can also take advantage of the property of group skyline computation that only new groups arrive and no existing groups expire (*i.e.*, the size of the sliding window can be set to infinite).

Recently Su et al. [22] propose top-k combinatorial skyline query (k-CSQ) processing which is similar in spirit to group skyline computation in that it also considers combinations of tuples. A combination is either a single tuple or a meta-tuple

that combines several tuples using monotone aggregate functions.[2] Combinatorial skyline tuples are then defined as the skyline of all combinations where the maximum number of tuples to be combined is specified by the user. k-CSQ processing returns k best combinatorial skyline tuples according to a preference order of attributes given by the user. In other words, it returns those k combinatorial skyline tuples whose aggregate values for the most preferred attribute are the highest. The preference order is crucial in reducing exponential search space. To find k best combinatorial skyline tuples, it suffices to first sort the dataset according to the user preference and then inspect combinations of only a small number of tuples that appear in the beginning of the sorted dataset.

Although Su et al. experimentally show the efficiency of their proposed algorithm for k-CSQ processing, it reports only those combinatorial skyline tuples whose aggregate values for a certain attribute are the highest. In contrast, group skyline computation considers only groups of the same number of tuples without considering the user preference. Furthermore we focus on various properties of group skyline computation and efficient generation of all skyline groups by exploiting such properties. As discussed in Section 5.5, we may also use skyline groups in more complicated decision making processes.

Another work superficially similar to but clearly different from group skyline computation concerns the problem of creating competitive products proposed by Wan et al. [26]. They also consider combinations of tuples but their objective is to create new products that are not dominated by already existing products. A product is a combination of tuples, or components, each of which comes from a different dataset called a source table and may have distinct attributes. For example, we may think of laptops as products, and CPUs, memories, and screens as their components. One important property in the problem of creating competitive products is that when we combine source tables to create new products, it suffices to consider only the skyline tuples in each source table, which reduces search space significantly. In contrast to creating competitive products which combines tuples of different kinds, group skyline computation combines tuples of the same kind. Hence group skyline computation is more appropriate for such problems as recruiting sport players or analyzing stock portfolios which deal with homogeneous datasets. Furthermore, as proved in Proposition 2.5, it needs to consider not only skyline tuples but also non-skyline tuples, which is the main difference from their work.

7. Conclusion

We have proposed the problem of group skyline computation which is based on dominance tests between groups of the same number of tuples. After studying its various properties, we have developed a group skyline algorithm GDynamic which uses the idea of dynamic programming in its design. Through various experiments, we have demonstrated that GDynamic is a practical group skyline algorithm.

Because of the exponential growth in the number of combinations of tuples, group skyline computation is inherently expensive in terms of both time and space. Like previous work on parallel skyline computation [8,19,21,25,27], we may consider the parallelization of group skyline computation on distributed or multicore architectures, which is left as future work. As another direction for future work, we may consider a problem of finding a small number of representative skyline groups, similarly to finding a small number of representative skyline tuples [15,24].

Acknowledgments

We thank Hyuk Jun Kweon for the example in Fig. 3 and Sunghwan Kim for the prototype implementation of the group skyline algorithms. We also thank the anonymous reviewers for their helpful comments. This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2009-0077543).

Appendix A

A.1. Proof of Proposition 3.2

Suppose that there exists a 3-skyline group $G_3 = \{p,q,r\}$ that contains no 2-skyline group (where $p \neq q$, $p \neq r$, $q \neq r$). We show by contradiction that such a 3-skyline group G_3 cannot exist.

Since G_3 contains no 2-skyline group, there exists a 2-skyline group that dominates $\{p,q\}$. Furthermore such a 2-skyline group G_2 must include r, since otherwise $G_2 \cup \{r\}$ would dominate G_3. Hence we assume a 2-skyline group $\{p',r\}$ such that $\{p',r\} \succ \{p,q\}$ with $p' \neq r$. Similarly we assume 2-skyline groups $\{q',p\}$ and $\{r',q\}$ such that $\{q',p\} \succ \{q,r\}$ with $q' \neq p$ and $\{r',q\} \succ \{p,r\}$ with $r' \neq q$.

By combining the three dominance relations $\{p',r\} \succ \{p,q\}$, $\{q',p\} \succ \{q,r\}$, and $\{r',q\} \succ \{p,r\}$, we obtain $\{p',q',r'\} \succ \{p,q,r\}$. Since $\{p,q,r\} = G_3$ is a 3-skyline group, $\{p',q',r'\}$ must contain duplicate tuples. Without loss of generality, we assume that p' and q' are the same tuple ($p' = q'$). Then $\{p',p\} \succ \{q,r\}$ follows from $\{q',p\} \succ \{q,r\}$. By combining $\{p',p\} \succ \{q,r\}$ with $\{p',r\} \succ$

[2] Incidentally their use of monotone aggregate functions is incorrect because Lemma 1 and Theorem 2 in [22] do not hold for the monotone aggregate function max. As in this paper, they should use aggregate functions that are strictly monotone in each argument.

$\{p,q\}$, we obtain $p' \succ q$ and consequently $\{p,p',r\} \succ \{p,q,r\}$. Below we show that $\{p,p',r\} \succ \{p,q,r\}$ contradicts one of the assumptions. We have two cases to consider: (1) $p \neq p'$ and (2) $p = p'$.

Suppose $p \neq p'$. Since $p \neq r$ and $p' \neq r$ already hold, $\{p,p',r\}$ contains no duplicate tuples and $\{p,p',r\} \succ \{p,q,r\}$ contradicts the assumption that $\{p,q,r\}$ is a 3-skyline group.

Suppose $p = p'$. From $p' \succ q$, we obtain $p \succ q$. From $\{p',r\} \succ \{p,q\}$, we obtain $r \succ q$. Since $p \succ q$ and $r \succ q$, either $\{r',q\}$ is not a 2-skyline group by Corollary 2.7 or $p = r$ holds. The former contradicts the assumption that $\{r',q\}$ is a 2-skyline group and the latter the assumption $p \neq r$.

Therefore a 3-skyline group always contains a 2-skyline group. □

A.2. Proof of Proposition 3.3

We construct D in such a way that q_i and q_i' form a pair and correspond to p_i for $1 \leqslant i \leqslant 4$. Every tuple in D is a skyline tuple and we have $\mathcal{T}_4(D) = D$. We show that $G_4 = \{p_1,p_2,p_3,p_4\}$ is a 4-skyline group, but no subset of size 3 is a 3-skyline group.

First we observe that none of $\{p_1,p_2,p_3\}$, $\{p_1,p_2,p_4\}$, $\{p_1,p_3,p_4\}$, and $\{p_2,p_3,p_4\}$ is a 3-skyline group:

$$\{p_1,q_1,q_1'\} \succ \{p_2,p_3,p_4\}$$
$$\{p_2,q_2,q_2'\} \succ \{p_1,p_3,p_4\}$$
$$\{p_3,q_3,q_3'\} \succ \{p_1,p_2,p_4\}$$
$$\{p_4,q_4,q_4'\} \succ \{p_1,p_2,p_3\}$$

For example, $\{p_1,q_1,q_1'\} \succ \{p_2,p_3,p_4\}$ holds as follows:

$$\Sigma\{p_1,q_1,q_1'\} = (1,1,0,0,0,0,0,0,4,9,9,9,5,5,5,8,8,8)$$
$$\Sigma\{p_2,p_3,p_4\} = (0,0,0,0,0,0,0,0,3,6,6,6,3,3,3,6,6,6)$$

The other three cases are all symmetric because of the way that we construct D. Hence G_4 contains no 3-skyline group.

Next we show that G_4 is indeed a 4-skyline group, that is, no $C_4 \subset D$ dominates G_4. There are three mutually exclusive cases to consider where we assume $1 \leqslant i \leqslant 4$ and $1 \leqslant j \leqslant 4$.

- For some pair $\{q_i,q_i'\}$, only one of $q_i \in C_4$ and $q_i' \in C_4$ holds. We assume $q_1 \in C_4$ and $q_1' \notin C_4$ without loss of generality.
- There is only one pair $\{q_i,q_i'\}$ such that both $q_i \in C_4$ and $q_i' \in C_4$ hold. In this case, $|C_4 \cap G_4| = 2$; otherwise, this case is covered by the first one. We assume $q_1 \in C_4$ and $q_1' \in C_4$ without loss of generality.
- C_4 consists of two pairs $\{q_i,q_i'\}$ and $\{q_j,q_j'\}$. Without loss of generality, we assume $C_4 = \{q_1,q_1',q_2,q_2'\}$.

In the first case, we have $(\Sigma C_4)[2] = -1$ and $(\Sigma G_4)[2] = 0$ and thus $C_4 \not\succ G_4$. In the second case, we have $(\Sigma C_4)[9] \leqslant 5$ and $(\Sigma G_4)[9] = 7$ and thus $C_4 \not\succ G_4$. In the third case, we have $(\Sigma C_4)[13] = 4$ and $(\Sigma G_4)[13] = 6$ and thus $C_4 \not\succ G_4$. Hence G_4 is a 4-skyline group. □

References

[1] M. Atallah, Y. Qi, Computing all skyline probabilities for uncertain data, in: Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, 2009, pp. 279–287.
[2] I. Bartolini, P. Ciaccia, M. Patella, Efficient sort-based skyline evaluation, ACM Transactions on Database Systems 33 (4) (2008) 1–49.
[3] C. Böhm, F. Fiedler, A. Oswald, C. Plant, B. Wackersreuther, Probabilistic skyline queries, in: Proceeding of the 18th ACM Conference on Information and Knowledge Management, ACM, 2009, pp. 651–660.
[4] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: Proceedings of the 17th International Conference on Data Engineering, IEEE Computer Society, 2001, pp. 421–430.
[5] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, Skyline with presorting, in: Proceedings of the 19th International Conference on Data Engineering, IEEE Computer Society, 2003, pp. 717–719.
[6] X. Ding, X. Lian, L. Chen, L. Jin, Continuous monitoring of skylines over uncertain data streams, Information Sciences 184 (1) (2012) 196–214.
[7] R. Fagin, A. Lotem, M. Naor, Optimal aggregation algorithms for middleware, Journal of Computer and System Sciences 66 (4) (2003) 614–656.
[8] A. Cosgaya-Lozano, A. Rau-Chaplin, N. Zeh, Parallel computation of skyline queries, in: Proceedings of the 21st Annual International Symposium on High Performance Computing Systems and Applications, IEEE Computer Society, 2007, p. 12.
[9] P. Godfrey, R. Shipley, J. Gryz, Maximal vector computation in large data sets, in: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB Endowment, 2005, pp. 229–240.
[10] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, ACM, 1984, pp. 47–57.
[11] G.T. Kailasam, J.-S. Lee, J.-W. Rhee, J. Kang, Efficient skycube computation using point and domain-based filtering, Information Sciences 180 (7) (2010) 1090–1103.
[12] D. Kossmann, F. Ramsak, S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB Endowment, 2002, pp. 275–286.
[13] K.C.K. Lee, B. Zheng, H. Li, W.-C. Lee, Approaching the skyline in Z order, in: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB Endowment, 2007, pp. 279–290.
[14] X. Lin, Y. Yuan, W. Wang, H. Lu, Stabbing the sky: efficient skyline computation over sliding windows, in: Proceedings of the 21st International Conference on Data Engineering, IEEE Computer Society, 2005, pp. 502–513.
[15] X. Lin, Y. Yuan, Q. Zhang, Y. Zhang, Selecting stars: the k most representative skyline operator, in: Proceedings of the 23rd International Conference on Data Engineering, IEEE Computer Society, 2007, pp. 86–95.

[16] M-H. Luk, M.L. Yiu, E. Lo, Group-by skyline query processing in relational engines, in: Proceeding of the 18th ACM Conference on Information and Knowledge Management, ACM, 2009, pp. 1433–1436.
[17] M. Morse, J.M. Patel, W.I. Grosky, Efficient continuous skyline computation, Information Sciences 177 (17) (2007) 3411–3437.
[18] D. Papadias, Y. Tao, G. Fu, B. Seeger, Progressive skyline computation in database systems, ACM Transactions on Database Systems 30 (1) (2005) 41–82.
[19] S. Park, T. Kim, J. Park, J. Kim, H. Im, Parallel skyline computation on multicore architectures, in: Proceedings of the 25th International Conference on Data Engineering, IEEE Computer Society, 2009, pp. 760–771.
[20] J. Pei, B. Jiang, X. Lin, Y. Yuan, Probabilistic skylines on uncertain data, in: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB Endowment, 2007, pp. 15–26.
[21] J. Selke, C. Lofi, W.-T. Balke, Highly scalable multiprocessing algorithms for preference-based database retrieval, in: Proceedings of the 15th International Conference on Database Systems for Advanced Applications (DASFAA), 2010, pp. 246–260.
[22] I.-F. Su, Y.-C. Chung, C. Lee. Top-k combinatorial skyline queries, in: Proceedings of the 15th International Conference on Database Systems for Advanced Applications (DASFAA), 2010, pp. 79–93.
[23] Y. Tao, D. Papadias, Maintaining sliding window skylines on data streams, IEEE Transactions on Knowledge and Data Engineering 18 (3) (2006) 377–391.
[24] Y. Tao, L. Ding, X. Lin, J. Pei, Distance-based representative skyline, in: Proceedings of the 25th International Conference on Data Engineering, IEEE Computer Society, 2009, pp. 892–903.
[25] A. Vlachou, C. Doulkeridis, Y. Kotidis, Angle-based space partitioning for efficient parallel skyline computation, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, 2008, pp. 227–238.
[26] Q. Wan, R.C.-W. Wong, I.F. Ilyas, M.T. Özsu, Y. Peng, Creating competitive products, Proceedings of the VLDB Endowment 2 (1) (2009) 898–909.
[27] P. Wu, C. Zhang, Y. Feng, B.Y. Zhao, D. Agrawal, A.E. Abbadi, Parallelizing skyline queries for scalable distribution, in: Proceedings of the 10th International Conference on Extending Database Technology, 2006, pp. 112–130.
[28] S. Zhang, N. Mamoulis, D.W. Cheung, Scalable skyline computation using object-based space partitioning, in: Proceedings of the 35th SIGMOD International Conference on Management of Data, ACM, 2009, pp. 483–494.