# Towards Multidimensional Subspace Skyline Analysis

JIAN PEI
Simon Fraser University
YIDONG YUAN and XUEMIN LIN
The University of New South Wales and NICTA
WEN JIN and MARTIN ESTER
Simon Fraser University
QING LIU
The University of Queensland
WEI WANG
The University of New South Wales
YUFEI TAO and JEFFREY XU YU
The Chinese University of Hong Kong
and
QING ZHANG
E-Health Research Centre/CSIRO ICT

The skyline operator is important for multicriteria decision-making applications. Although many recent studies developed efficient methods to compute skyline objects in a given space, none of them considers skylines in multiple subspaces simultaneously. More importantly, the fundamental

The research of J. Pei was partially supported by NSERC Discovery Grants, NSERC Collaborative Research and Development Grants, NSF Grant IIS-0308001, and IBM Eclipse Innovation Awards. The research of Y. Yuan and Q. Lin was partially supported by ARC Discovery Grant (DP0666428). The research of M. Ester was partially supported by NSERC. The research of Y. Tao was partially supported by the RGC grant CityU 1163/04E from the HKSAR government. The research of J. X. Yu was partially supported by RGC grant (CUHK418205) of HKSAR. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.
Authors' addresses: J. Pei, W. Jin, M. Ester, Simon Fraser University, Canada; email: {jpei,wjin,ester}@cs.sfu.ca; Y. Yuan (also NICTA, Australia), X. Lin (also NICTA, Australia), W. Wang, The University of New South Wales, Australia; email: {yyidong,lxue,weiw}@cse.unsw.edu.au; Q. Liu, The University of Queensland, Australia; email: qing@itee.uq.edu.au; Y. Tao, J. X. Yu, The Chinees University of Hong Kong; email: taofy@cs.cityu.edu.hk, yu@se.cuhk.edu.hk; Q. Zhang, E-Health Research Centre/CSIRO ICT Centre, Australia; email: qing.zhang@csiro.au.

problem on the *semantics* of skylines remains open: Why and in which subspaces is (or is not) an object in the skyline? Practically, users may also be interested in the skylines in any subspaces. Then, what is the relationship between the skylines in the subspaces and those in the super-spaces? How can we effectively analyze the subspace skylines? Can we efficiently compute skylines in various subspaces and answer various analytical queries?

In this article, we tackle the problem of multidimensional subspace skyline computation and analysis. We explore skylines in subspaces. First, we propose the concept of SKYCUBE, which consists of skylines of all possible nonempty subspaces of a given full space. Once a SKYCUBE is materialized, any subspace skyline queries can be answered online. However, SKYCUBE cannot fully address the semantic concerns and may contain redundant information. To tackle the problem, we introduce a novel notion of *skyline group* which essentially is a group of objects that coincide in the skylines of some subspaces. We identify the *decisive subspaces* that qualify skyline groups in the subspace skylines. The new notions concisely capture the semantics and the structures of skylines in various subspaces. Multidimensional roll-up and drill-down analysis is introduced. We also develop efficient algorithms to compute SKYCUBE, skyline groups and their decisive subspaces. A systematic performance study using both real data sets and synthetic data sets is reported to evaluate our approach.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Skyline query, multidimensional data analysis, data cubing

## 1. INTRODUCTION

It has been well recognized that the *skyline operator* is important for multicriteria decision making applications. A (classic) illustrative example of skyline queries is to search for hotels in Nassau (Bahamas) which are cheap and close to the beach [Börzsönyi et al. 2001]. Suppose each hotel has two attributes: the price and the distance to the beach. Hotel $p$ *dominates* hotel $q$ (or, $p$ is a better choice than $q$ in the context of this example) if $p.price \leq q.price$, $p.distance \leq q.distance$ and at least one inequality holds. Those hotels not dominated by others in terms of price and distance to the beach form the skyline. In other words, the skyline hotels are all possible trade-offs between price and distance to the beach that are superior to other hotels not in the skyline.

There are many recent studies on efficient methods for skyline computation. Please see Section 6 for a brief review. However, the fundamental questions about the semantics of skyline remain open.

*Example* 1 (*Intuition*). Consider a set of 5 objects in 2D space $(A, B)$ as shown in Figure 1. It is easy to verify that objects $P_1$, $P_2$, and $P_3$ are in the skyline in space $(A, B)$ since each of them is not dominated by any other objects.

In the same figure, we also plot the projections of the objects on dimensions $A$ and $B$, respectively. In subspace $A$, the projections of $P_1$ and $P_4$ collapse. Both are in the subspace skyline of $A$. In subspace $B$, the projection of $P_3$ is in the subspace skyline. Since all objects collapse in the trivial subspace $\emptyset$, hereafter, we use the term *subspace* to refer to only nonempty ones except for specifically mentioned ones.

Although $P_1$, $P_2$, and $P_3$ all are skyline objects in the full space $(A, B)$, there are some subtle differences among them. Both $P_1$ and $P_3$ have some projections which are in some subspace skylines (i.e., subspaces $A$ and $B$, respectively), but

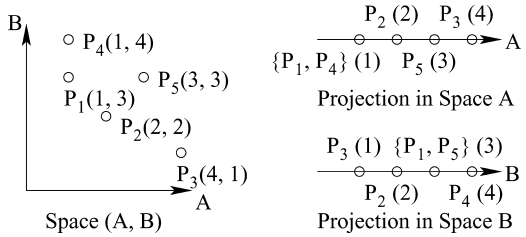



Fig. 1. An example showing the intuition.

no projection of $P_2$ is in the subspace skyline of any proper subspace. A closer look finds that $P_1$ taking value 1 on dimension $A$ is already sufficient to qualify $P_1$ as a skyline object. Similarly, the value 1 of $P_3$ on dimension $B$ is critical for the skyline membership of $P_3$. On the other hand, $P_2$ is a skyline object only if both dimensions $A$ and $B$ are considered—it needs two dimensions to qualify. In other words, subspaces $A$ and $B$ are decisive to $P_1$ and $P_3$, respectively, and subspace $(A, B)$ is decisive to $P_2$.

Although both $P_4$ and $P_5$ are not in the skyline in space $(A, B)$, they are still subtly different if we look at the subspaces. The projection of $P_4$ is in the skyline in subspace $A$ but $P_5$ has no projection belonging to a subspace skyline. $P_4$ is dominated by $P_1$, nevertheless, the dominance is partial, that is, $P_4$ takes the same value as $P_1$ in dimension $A$ and thus has the chance to be in the skyline in subspace $A$. In other words, subspace $A$ is also decisive to $P_4$ since $A$ enables $P_1$ as a skyline object in some subspace (subspace $A$ itself in this example).

While the skyline in Example 1, which involves a two-dimensional space and only a few objects, is simple and easily perceived, the general situation may be much more complicated when many dimensions and many objects are involved. Nevertheless, the observations in Example 1 illustrate one important intuition. Skylines in various subspaces are interesting and meaningful. Whether an object is in the skylines of the full space or of some subspaces is determined by the values of the object in some *decisive subspaces*. The decisive subspaces and the values in those subspaces vary from object to object in the skyline. For a particular object, the values in its decisive subspaces justify why and in which subspaces the object is in the skyline—the *semantics* of the object with respect to skyline.

Why should we care about the semantics of skylines? Semantics is important in order to understand data. For example, Section 7 analyzes a real data set which contains 17,226 records of Great NBA Players' seasonal performance from 1960 to 2001. Wilt Chamberlain's performance in 1960 is in the skyline of the full space, which can be identified by the conventional skyline computation methods. However, one may wonder which merits really make Chamberlain that outstanding. The semantics analysis in Section 7 shows that Chamberlain was outstanding in total rebounds in the season of 1960 by achieving the record of 2,149 in the NBA history. The attribute of total rebounds is the decisive subspace that establishes his superior status. In fact, he was not exceptional in any other factors such as total assists. As another example, Michael Jordan does not hold any record in any single attribute. However, his performance in

1988 is in the skyline of subspaces (total points, total rebounds, total assists) and (games played, total points, total assists), and also in the skyline of the full space. The two subspaces are decisive and explain why Michael Jordan is an outstanding player. Clearly, such information cannot be captured by the traditional skyline computation and analysis in the full space.

The concepts of skyline groups and decisive subspaces can also be used immediately for efficient query answering. For example, given an object or a group of objects, a *skyline membership query* determines the subspaces where the object(s) are in the subspace skylines. On the other hand, given a subspace, the *subspace skyline query* finds the set of objects whose projections are in the subspace skyline.

The investigation of skylines in subspaces naturally introduces the problem of subspace skyline analysis and computation: for a set of subspaces, find the objects and their projections that are in the skylines of the subspaces, and analyze their relationship. These types of queries are interesting and useful in practice since, more often than not, a user may want to interactively examine the skylines with respect to different combinations of attributes.

Motivated by these observations, in this article, we study the problem of multidimensional subspace skyline computation and analysis. We make the following contributions.

—*We propose the concept of* SKYCUBE*, which consists of skylines of all possible nonempty subspaces of a given full space*. Once a SKYCUBE is materialized, any subspace skyline queries can be answered online.

—*We develop a theoretical framework to answer the question about semantics of skyline: Why and in which subspaces is an object in the skyline?* The semantics of skyline objects is concisely captured by the novel notions of *skyline groups* and the corresponding *decisive subspaces*. The subspaces where an object (or a set of objects) is in the skyline can be effectively determined by the skyline groups that the object belongs to and their decisive subspaces.

—*We investigate the problem of subspace skyline analysis.* Skylines in subspaces can be concisely summarized by skyline groups. Moreover, skyline objects in the full space can be selected as the representatives in skyline groups. They catch the contour (i.e., technically, the projections) of the skylines. The multidimensional roll-up and drill-down analysis is useful to support the online analytic processing of skylines.

—*We present efficient algorithms for subspace skyline computation.* We develop two algorithmic frameworks of different styles which can compute both a SKYCUBE and the set of skyline groups as well as their decisive subspaces effectively. The algorithms make use of several important computation sharing principles which are unique in the context of SKYCUBE computation.

—*A systematic performance study using both synthetic and real data sets is conducted to evaluate our approach.* We showcase some interesting findings in the skyline semantic analysis using the real data set about technical statistics of NBA players and justify why they are meaningful in practice. Moreover, we use benchmark synthetic data sets to test the efficiency and the scalability of our algorithms.

The rest of the article is organized as follows. In Section 2, we extend the concept of skyline to multidimensional subspaces and introduce the concept of SKYCUBE. In Section 3, we introduce the notions of skyline groups and decisive subspaces and justify how the new notions capture the semantics of objects with respect to skylines. In Section 4, we tackle the problem of subspace skyline analysis. In Section 5, we present the algorithmic frameworks for subspace skyline computation, including the specific algorithms to compute SKYCUBE and skyline groups as well as their decisive subspaces. We review related work in Section 6. An extensive performance study is reported in Section 7. The article is concluded in Section 8.

## 2. MULTIDIMENSIONAL SUBSPACE SKYLINES AND SKYCUBE

In this section, we extend the concept of skyline to multidimensional subspaces, and introduce the concept of SKYCUBE.

Hereafter, by default we consider a set $S$ of objects in a $d$-dimensional space $\mathcal{D} = (D_1, \ldots, D_d)$, where dimensions $D_1, \ldots, D_d$ are in the domain of numbers. For the sake of brevity, we often do not explicitly mention $S$ and $\mathcal{D}$ when they are clear in the context.

For objects $p, q \in S$, $p$ is said to *dominate* $q$ if $p.D_i \leq q.D_i$ for $1 \leq i \leq d$, and there exists at least one dimension $D_{i_0}$ such that $p.D_{i_0} < q.D_{i_0}$. Object $p$ is a *skyline object* if $p$ is not dominated by any other objects in $S$.

The notion of skyline can be intuitively extended to subspaces.

*Definition* 2.1 (*Subspace skyline*). A subset of dimensions $\mathcal{B} \subseteq \mathcal{D}$ ($\mathcal{B} \neq \emptyset$) forms a (nontrivial) $|\mathcal{B}|$-*dimensional subspace* of $\mathcal{D}$. For an object $p$ in space $\mathcal{D}$, the projection of $p$ in subspace $\mathcal{B}$, denoted by $p_{\mathcal{B}}$, is a $|\mathcal{B}|$-tuple $(p.D_{i_1}, \ldots, p.D_{i_{|\mathcal{B}|}})$, where $D_{i_1}, \ldots, D_{i_{|\mathcal{B}|}} \in \mathcal{B}$ and $i_1 < \cdots < i_{|\mathcal{B}|}$.

The projection of an object $p$ ($p \in S$) in subspace $\mathcal{B} \subseteq \mathcal{D}$ is in the *subspace skyline* (of $\mathcal{B}$) if $p_{\mathcal{B}}$ is not dominated by any $q_{\mathcal{B}}$ in $\mathcal{B}$ for any other object $q \in S$. $p$ is also called a *subspace skyline object (of $\mathcal{B}$)*.

For example, in Figure 1, the projections of both $P_1$ and $P_4$ are in the subspace skyline in subspace $A$, and the projection of $P_3$ is in the subspace skyline in subspace $B$.

Skylines in all possible nonempty subspaces form a SKYCUBE.

*Definition* 2.2 (SKYCUBE). The set of all skyline objects in subspace $\mathcal{B}$ is denoted by $SKY_{\mathcal{B}}(S)$. A SKYCUBE is the set of all skylines in all possible nonempty subspaces, that is,

$$\text{SKYCUBE}(S, \mathcal{D}) = \{(\mathcal{B}, SKY_{\mathcal{B}}(S)) | \mathcal{B} \subseteq \mathcal{D}, \mathcal{B} \neq \emptyset\}.$$

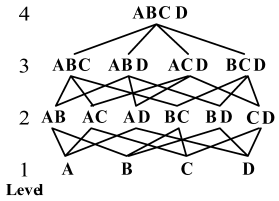
$SKY_{\mathcal{B}}(S)$ is called a *cuboid* of subspace $\mathcal{B}$.

*Example* 2 (SKYCUBE). Consider the objects in Figure 2(a) as our running example. The SKYCUBE is shown in Figure 2(c).

The structure of the SKYCUBE can be visualized in a lattice structure similar to that of the data cube in the data warehouse (see the example in Figure 2(b)). From the bottom to the top of the SKYCUBE, we number each level of the cuboid increasingly. For two cuboids $SKY_{\mathcal{U}}(S)$ of subspace $\mathcal{U}$ and $SKY_{\mathcal{V}}(S)$ of

| | A | B | C | D |
|---|---|---|---|---|
| $P_1$ | 1 | 4 | 5 | 7 |
| $P_2$ | 1 | 3 | 6 | 7 |
| $P_3$ | 2 | 3 | 5 | 8 |

(a) Dataset $S$




(b) Lattice Structure of a SKYCUBE

| Subspace | Skyline |
|---|---|
| (A, B, C, D) | {$P_1$, $P_2$, $P_3$} |
| (A, B, C) | {$P_1$, $P_2$, $P_3$} |
| (A, B, D) | {$P_2$} |
| (A, C, D) | {$P_1$} |
| (B, C, D) | {$P_1$, $P_2$, $P_3$} |
| (A, B) | {$P_2$} |
| (A, C) | {$P_1$} |
| (A, D) | {$P_1$, $P_2$} |
| (B, C) | {$P_3$} |
| (B, D) | {$P_2$} |
| (C, D) | {$P_1$} |
| (A) | {$P_1$, $P_2$} |
| (B) | {$P_2$, $P_3$} |
| (C) | {$P_1$, $P_3$} |
| (D) | {$P_1$, $P_2$} |

(c) Skycube

Fig. 2. SKYCUBE of the running example.

subspace $\mathcal{V}$ in a SKYCUBE, if $\mathcal{U} \subset \mathcal{V}$, we call $SKY_{\mathcal{V}}(S)$ an *ancestor* of $SKY_{\mathcal{U}}(S)$, and $SKY_{\mathcal{U}}(S)$ a *descendant* of $SKY_{\mathcal{V}}(S)$. Specially, if $\mathcal{U} \subset \mathcal{V}$ and $|\mathcal{V} - U| = 1$, we also call $SKY_{\mathcal{V}}(S)$ a *parent cuboid* of $SKY_{\mathcal{U}}(S)$, and $SKY_{\mathcal{U}}(S)$ a *child cuboid* of $SKY_{\mathcal{V}}(S)$.

A subspace skyline query asks for the set of skyline objects in a given subspace. Clearly, once a SKYCUBE is materialized, any subspace skyline queries can be answered efficiently since all subspace skylines are precomputed.

A SKYCUBE may contain redundant information. For example, in Figure 2(c), we can observe that the skylines in subspaces $A$, $D$, and $(A, D)$ have the identical set of objects. As another example, the skylines in subspaces $(A, C)$, $(C, D)$, and $(A, C, D)$ consist of the same set of objects. Why may a group of objects appear together in the skylines of some subspaces? Can we eliminate the redundancy in skyline representations and also in multidimensional skyline analysis? These questions motivate our exploration of the semantics of subspace skylines.

## 3. SEMANTICS

For an object or a set of objects, in which subspaces is the object (or are the objects) in the skylines? We answer this question in this section. We first illustrate the ideas. Then, we introduce the new notions. Last, we elaborate on how the new notions capture the semantics of skyline objects and answer skyline membership queries.

### 3.1 Ideas

In general, if an object $p$ is in the skylines of subspaces $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $\mathcal{B}_1 \subset \mathcal{B}_2$, can we declare that $p$ is also in the skyline of any subspace $\mathcal{B}$ in between,

that is, $\mathcal{B}_1 \subset \mathcal{B} \subset \mathcal{B}_2$? This property is appealing since it may considerably simplify the determination of skyline membership in subspaces. Unfortunately, the general situation is far from being so simple.

*Example* 3 (*Ideas*). Consider the objects in Figure 2(a) again, and the subspace skylines in Figure 2(c). We obtain the following two observations.

First, object $P_1$ is in the skylines of full space $(A, B, C, D)$ and of subspace $A$. However, it is not in the skyline of subspace $(A, B)$ since its projection, $(1, 4, *, *)$, is dominated by $(1, 3, *, *)$, the projection of $P_2$. This demonstrates that, in general, for subspaces $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}$ such that $\mathcal{B}_1 \subset \mathcal{B} \subset \mathcal{B}_2$, even though an object is in the subspace skylines of $\mathcal{B}_1$ and $\mathcal{B}_2$, it may not be in the subspace skyline of $\mathcal{B}$ in between.

Second, objects $P_1$ and $P_2$ collapse in subspace $(A, D)$. The projection $(1, *, *, 7)$ is in the subspace skyline of $(A, D)$. Thus, any dimension values in subspaces $A$, $D$ or $(A, D)$ qualifying $P_1$ as a subspace skyline object in those subspaces also establish the same qualification for $P_2$, and vice versa. In other words, if a group of objects collapse in a subspace $\mathcal{B}$, then the objects in the same group share the skyline membership in all subspaces of $\mathcal{B}$.

The observations in Example 3 can be summarized as follows.

—Generally, the skyline membership is not monotonic, that is, being in the skyline of subspace $\mathcal{B}$ does not automatically qualify an object in the skyline of superspaces of $\mathcal{B}$. The object may be dominated in the superspaces of $\mathcal{B}$ by some other objects which have the same values in $\mathcal{B}$.

—Objects coincide and form groups in subspaces. The skyline memberships in subspaces are shared by all objects in the same group. The convergence and divergence of groups from subspace to subspace play critical roles in forming the skylines of various subspaces. Therefore, it is critical to capture groups of objects whose shared projections are in the skylines of the subspaces.

### 3.2 Skyline Groups and Decisive Subspaces

Now, let us consider objects that collapse in subspaces. They form groups that are critical in our multidimensional subspace skyline analysis.

*Definition* 3.1 (*C-group*). Let $G \subseteq S$ be a subset of objects and $\mathcal{B} \subseteq D$ be a subspace. $(G, \mathcal{B})$ is a *coincident group* (or *c-group* for short) if, for each dimension in $\mathcal{B}$, all objects in $G$ share a same value. The projection of the group in $\mathcal{B}$, denoted by $G_{\mathcal{B}}$, is $u_{\mathcal{B}}$ where $u \in G$.

A c-group $(G, \mathcal{B})$ is *maximal* if no any other objects $p \in (S - G)$ share the same values as those in $G$ on dimensions in $\mathcal{B}$, and objects in $G$ do not share a same value on any other dimension $D \in (\mathcal{D} - \mathcal{B})$. $\mathcal{B}$ is called the *signature subspace* of $G$.

*Example* 4 (*C-group*). Consider objects $P_1$ and $P_2$ in Figure 2(a). They share the same value on dimension $A$. Thus, $P_1$ and $P_2$ form a coincident group (or c-group for short) on $A$.

No other objects have the same value on $A$. Although we cannot add new objects into the group $(\{P_1, P_2\}, A)$, it can be expanded by including more dimensions. $P_1$ and $P_2$ share the same values in $A$ and $D$ but no other dimensions. Thus, we can maximize the group to include dimension $D$. $(\{P_1, P_2\}, (A, D))$ is a maximal c-group.

Given a subset of objects $G$, we define $\mathcal{I}(G)$ as the maximum set of dimensions that all objects in $G$ share the same values. That is,

$$\mathcal{I}(G) = \{D_i | D_i \in \mathcal{D}, \forall p, q \in G : u.D_i = v.D_i\}.$$

Moreover, for a subspace $\mathcal{B}$ and a set of objects $G$, we define $\mathcal{O}(G, \mathcal{B})$ as the maximum set of objects that have the same values on dimensions in $\mathcal{B}$ as objects in $G$. That is,

$$\mathcal{O}(G, \mathcal{B}) = \{p | p \in S, \forall D_i \in \mathcal{B}\ \forall q \in G : p.D_i = q.D_i\}.$$

For example, on the data set in Figure 2, $\mathcal{I}(\{P_1, P_2\} = AD$, and $\mathcal{O}(\{P_3\}, C) = \{P_1, P_3\}$.

The previous two operators can be used to derive maximal c-groups for any given subset of objects or a subset of objects and a subspace. Generally, given a set of objects $G$, we can get a maximal c-group in two steps. First, we can find the maximum subspace $\mathcal{I}(G)$ where the objects in $G$ share the same projection. Then, we can insert into $G$ all other objects sharing the same projection $G_{\mathcal{I}(G)}$. On the other hand, if we want to derive a group in a subspace $\mathcal{B}$ or its superspace, we can first insert into $G$ all other objects sharing the projection $G_{\mathcal{B}}$ (i.e., the set of objects becomes $\mathcal{O}(G, \mathcal{B})$), and then find the maximum subspace where the objects in $\mathcal{O}(G, \mathcal{B})$ share the same projection.

The following lemma formalizes these derivations and can be shown using the related definitions immediately there after.

LEMMA 3.2 (C-GROUP). *For a given subset of objects $G$, $(\mathcal{O}(G, \mathcal{I}(G)), \mathcal{I}(G))$ is a maximal c-group. For a given c-group $(G, \mathcal{B})$, $(\mathcal{O}(G, \mathcal{B}), \mathcal{I}(\mathcal{O}(G, \mathcal{B})))$ is a maximal c-group.*

We are particularly interested in maximal c-groups whose projections are in the skylines of some subspaces. Intuitively, we want to capture the subsets of values in their projections that are decisive to their skyline memberships.

*Definition* 3.3 (*Skyline group and decisive subspace*). Maximal c-group $(G, \mathcal{B})$ is called a *skyline group* if $G_{\mathcal{B}}$ is in the subspace skyline of $\mathcal{B}$.

For skyline group $(G, \mathcal{B})$, a subspace $\mathcal{C} \subseteq B$ is called *decisive* if (1) $G_{\mathcal{C}}$ is in the subspace skyline of $\mathcal{C}$; (2) $\mathcal{O}(G, \mathcal{C}) = G$; and (3) there exists no proper subspace $\mathcal{C}' \subset C$ such that conditions (1) and (2) also hold for $\mathcal{C}'$.

The *signature* of skyline group $(G, \mathcal{B})$ is written as $Sig(G, \mathcal{B}) = \langle G_{\mathcal{B}}, \mathcal{C}_1, \ldots, \mathcal{C}_k \rangle$, where $\mathcal{C}_1, \ldots, \mathcal{C}_k$ are all decisive subspaces of the skyline group.

Conditions (1) and (3) are straightforward. Condition (2) requires that the decisive subspaces are exclusive to the group $G$. This reflects our intension to catch the decisive factors for a group of objects that are in the (subspace) skylines. We will revisit this point soon when we discuss the semantics.

*Example* 5 (*Skyline group*). Consider the objects in Figure 2(a) again. In the full space, $P_1$, $P_2$, and $P_3$ are unique and each of them is a skyline object, therefore, each of them forms a maximal c-group in space $(A, B, C, D)$. Each group contains only one object.

For group $(P_1, ABCD)$, where $P_1$ and $ABCD$ are shorthand for set $\{P_1\}$ and subspace $(A, B, C, D)$, respectively, subspace $CD$ is decisive. Please note that $AD$ is not a decisive subspace for the group since $P_1$ collapses with $P_2$ in $AD$, and the maximal c-group in $AD$ contains two objects, that is, $\mathcal{O}(P_1, AD) = P_1P_2$. In other words, condition (2) in the definition is violated. Another decisive subspace for this group is $AC$. Thus,

$$Sig(P_1, ABCD) = \langle (1, 4, 5, 7), AC, CD \rangle.$$

As another example, for group $(P_1P_2, AD)$, its projection is in the subspace skyline of $AD$. The group has two decisive subspaces, namely $A$ and $D$. Thus,

$$Sig(P_1P_2, AD) = \langle (1, *, *, 7), A, D \rangle.$$

Similarly, we have $Sig(P_1P_3, C) = \langle (*, *, 5, *), C \rangle$.

### 3.3 Semantics of (Subspace) Skyline Objects

The question about the semantics[1] asks: For a given object or a group of objects, can we determine the subspaces where the projections of the object(s) are in the subspace skyline?

THEOREM 3.4 (DECISIVE SUBSPACE). *For skyline group $(G, \mathcal{B})$, if $\mathcal{C}$ is a decisive subspace, then for any subspace $\mathcal{C}'$ such that $\mathcal{C} \subseteq \mathcal{C}' \subseteq B$, $G_{\mathcal{C}'}$ is in the subspace skyline.*

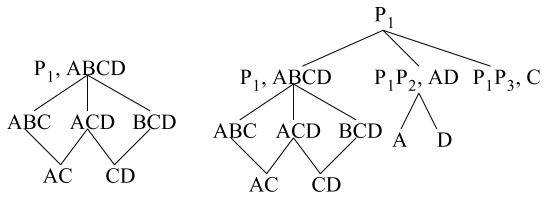
PROOF. We prove by contradiction. Suppose $G_{\mathcal{C}'}$ is not in the subspace skyline, and is dominated by an object $p_{\mathcal{C}'}$ in subspace $\mathcal{C}'$. Then, $p \notin G$. For each dimension $D_i \in \mathcal{C}'$, $p.D_i \leq G_{\mathcal{B}}.D_i$ and the inequality holds on at least one dimension. On the other hand, since $\mathcal{C}$ is decisive, $G_{\mathcal{C}}$ is not dominated by the projections of any other objects. Thus, $G_{\mathcal{C}} = p_{\mathcal{C}}$. That means, $\mathcal{O}(G, \mathcal{C}) \supset G$, which violates condition (2) in Definition 3.3. □

Theorem 3.4 indicates how decisive subspaces capture the semantics of skyline objects: The skyline membership of an object or a group of objects is established by its decisive subspaces.

*Example* 6 (*Semantics*). As shown in Example 5, $Sig(P_1, ABCD) = \langle (1, 4, 5, 7), AC, CD \rangle$. Thus, $P_1$ is in the skylines of subspaces inclusively bordered by $ABCD$, $AC$ and $CD$, as shown in Figure 3(a). This also explains why we opt for the representation of signature.

The signature of skyline group $(P_1, ABCD)$ explains why and in which subspaces $P_1$ is in the skyline without any accompanying coincident objects. $P_1$ coincides with $P_2$ and $P_3$ in some subspaces and thus may jointly be in some subspace skylines. This is captured by the corresponding skyline groups.

[1]Here, we use the term *semantics* to refer to the meaning and the explanation of data.




(a) Subspaces determined by group ($P_1$, ABCD)

(b) Subspace where $P_1$ is in the subspace skyline

Fig. 3. The subspaces where object $P_1$ in Figure 2(a) Is in the skyline.

THEOREM 3.5 (SEMANTICS). *An object $p$ is in the skyline of subspace $\mathcal{C}$ if and only if there exists a skyline group $(G, \mathcal{B})$ and its decisive subspace $\mathcal{C}'$ such that $p \in G$ and $\mathcal{C}' \subseteq C \subseteq B$.*

PROOF. (Direction if). Following Theorem 3.4, $G_{\mathcal{C}}$ is in the subspace skyline. Since $p \in G$, $p_{\mathcal{C}}$ is also in the subspace skyline.

(Direction only-if). Consider the group of objects $\mathcal{O}(p, \mathcal{C})$. All objects in the group are in the subspace skyline of $\mathcal{C}$ since they share the same values as $p$ on dimensions in $\mathcal{C}$. Following Lemma 3.2, $(\mathcal{O}(p, \mathcal{C}), \mathcal{I}(\mathcal{O}(p, \mathcal{C})))$ is a maximal c-group. Furthermore, it is easy to see that the group must be in the skyline of subspace $\mathcal{I}(\mathcal{O}(p, \mathcal{C}))$. Thus, the group is a skyline group. We notice that subspace $\mathcal{C}$ satisfies conditions (1) and (2) of Definition 3.3. Thus, if $\mathcal{C}$ is minimal, then $\mathcal{C}$ itself is decisive, i.e., $\mathcal{C}' = C$. Otherwise, there must exist a $\mathcal{C}' \subset C$ that $\mathcal{C}'$ is decisive. □

Theorem 3.5 can be extended to the cases of a set of objects that appear in subspace skylines simultaneously.

COROLLARY 3.6 (A SET OF SKYLINE OBJECTS). *A set of objects $S$ are together in the skyline of subspace $\mathcal{C}$ if and only if there exists a skyline group $(G, \mathcal{B})$ and its decisive subspace $\mathcal{C}'$ such that $S \subseteq G$ and $\mathcal{C}' \subseteq \mathcal{C} \subseteq \mathcal{B}$.*

### 3.4 Answering Skyline Membership Queries

Theorem 3.5 leads to a generic yet simple framework of answering skyline membership queries using skyline groups and their signatures: given an object or a group of objects, determine the subspaces where the object(s) are in the subspace skylines.

Suppose the set of skyline groups and their signatures are materialized. (The algorithm for computing skyline groups and their signatures will be given in Section 5.) Then, instead of searching all possible subspaces, we only need to check the skyline groups in which the object is a member. This is effective since only the signatures of the skyline groups are needed. Moreover, the skyline groups can be indexed by their signatures to speed up the search. To illustrate, we give an intuitive example here.

*Example* 7 (*Semantics continued*). Continued from Example 6, $P_1$ is a member of skyline group $(P_1P_2, AD)$ which has decisive subspaces $A$ and $D$. Thus, $P_1$ is also in the subspace skylines of $A$, $D$, and $AD$. Similarly, as a

member of group $(P_1P_3, C)$, $P_1$ is in the subspace skyline of $C$. The complete set of subspaces where $P_1$ is in the skyline is shown in Figure 3(b).

## 4. SUBSPACE SKYLINE ANALYSIS

The notion of skyline groups naturally leads us to explore skylines in subspaces. When skylines in all subspaces are considered, it is imperative to ask: How are the subspace skylines formed and what is the relationship among them?

### 4.1 Intuition

We try to decipher some elegant structures embedded in the subspace skylines.

Skylines in subspaces consist of projections of objects. For a projection that is in the skyline of a subspace, the set of objects that share the same projection form a c-group. By the c-group containment relationship, the projections in subspace skylines form a lattice called the *skyline projection lattice* (Theorem 4.1 in Section 4.2), which is a concise structure.

The projection lattice may contain redundant information. The critical point here is that some projections in skylines of different subspaces may be made by the same maximal group of objects. Conceptually, a skyline group is a maximal group of objects that coincide in some subspaces and whose projections are also in the subspace skyline. Therefore, we can use skyline groups to derive a concise representation. Later in this section, we will show that the lattice of skyline groups, called the *skyline group lattice*, is a quotient lattice of the skyline projection lattice (Theorem 4.2 in Section 4.2).

Manipulating groups of objects all the time is still inconvenient. Ideally, we would like to select some representatives for the skyline groups. Fortunately, this is achievable since each skyline group must contain at least one object that is in the skyline of the full space. This indicates that the full space skyline casts the contours of skylines in subspaces.

### 4.2 Skyline Group Lattice

A projection $p_{\mathcal{B}}$ where $p \in S$ is called a *skyline projection* if it is in the skyline of $\mathcal{B}$. We can define a relation $\sqsubseteq$ on the set $\mathcal{P}$ of all skyline projections: for $u, v \in \mathcal{P}$ that are in the subspace skylines of $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively, $u \sqsubseteq v$ if $\mathcal{B}_1 \supseteq \mathcal{B}_2$ and $u_{\mathcal{B}_2} = v$.

For example, on the dataset in Figure 2, the projection of $P_1$ on $A$, $(1, *, *, *)$ and the projection of $P_2$ on $AD$, $(1, *, *, 7)$ are skyline projections since they are in the subspace skylines. $(1, *, *, 7) \sqsubseteq (1, *, *, *)$.

THEOREM 4.1 (SKYLINE PROJECTION LATTICE). *Let $\mathcal{P}$ be the set of all skyline projections with respect to a set of objects $S$. $(\mathcal{P}, \sqsubseteq)$ is a complete lattice if $(*, *, \ldots, *)$ and $\emptyset$ are treated as the two trivial skyline projections for the unit element and the zero element, respectively.*

PROOF. Obviously, $\sqsubseteq$ is a partial order on $\mathcal{P}$. We also notice that $\emptyset$ is the projection of any objects in subspace $\emptyset$ (the trivial subspace), and $(*, *, \ldots, *)$ is the projection of an empty set of objects on all dimensions. They are trivial

and just technically make up the lattice. The completeness of the lattice follows from the fact that the number of skyline projections is finite. □

To characterize that multiple skyline projections may be made by one maximal group of objects, we define an equivalence among skyline projections as follows. For any projection $u$ in subspace $\mathcal{B}$, define the *pre-image* of $u$ as the set of objects that have $u$ as the projection in $\mathcal{B}$, denoted by $pre(u) = \{p | p \in S, p_{\mathcal{B}} = u\}$. For two skyline projections $u$ and $v$ in subspaces $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively, they are equivalent (in terms of being generated by the same group of objects), denoted by $u \sim v$, provided $pre(u) = pre(v)$.

THEOREM 4.2 (SKYLINE GROUP LATTICE). *Let $\mathcal{SG}$ be the set of all skyline groups. $(\mathcal{SG}, \sqsubseteq)$ forms a complete lattice where $\sqsubseteq$ is on the projections in the groups. Moreover, $(\mathcal{SG}, \sqsubseteq) = (\mathcal{P}, \sqsubseteq)/\sim$.*

PROOF. The claim follows from the fact that a skyline group also expands to include all possible dimensions where the objects share the same projections. For any skyline projections $u$ in subspace $\mathcal{B}_1$ and $v$ in subspace $\mathcal{B}_2$ such that $u \sim v$, let $G = pre(u) = pre(v)$. We show $u$ and $v$ are in fact in the same skyline group.

According to Lemma 3.2, $(\mathcal{O}(G, \mathcal{I}(G)), \mathcal{I}(G))$ is a maximal c-group. From the definition of pre-image, we know $\mathcal{O}(G, \mathcal{I}(G)) = G$, $\mathcal{B}_1 \subseteq \mathcal{I}(G)$ and $\mathcal{B}_2 \subseteq \mathcal{I}(G)$. Now, we show that $(G, \mathcal{I}(G))$ is a skyline group by contradiction.

Suppose $(G, \mathcal{I}(G))$ is not a skyline group, that is, $G_{\mathcal{I}(G)}$ is not in the skyline of subspace $\mathcal{I}(G)$. Then, there must exist an object $p \notin G$ such that $p_{\mathcal{I}(G)}$ dominates $G_{\mathcal{I}(G)}$, i.e., for each dimension $D_i \in \mathcal{I}(G)$, $p.D_i \leq q.D_i$ where $q \in G$. However, since $G_{\mathcal{B}_1}$ is in the skyline of subspace $\mathcal{B}_1$, $p_{\mathcal{B}_1} = q_{\mathcal{B}_1}$. In other words, $p \in pre(u) = G$. That leads to a contradiction.

Thus, $u$ and $v$ are in fact the projection of skyline group $(\mathcal{O}(G, \mathcal{I}(G)), \mathcal{I}(G))$ on $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively. That means $u$ and $v$ belong to the same skyline group. □

Theorem 4.2 shows that skyline groups capture skyline projections in subspace skylines effectively, and the signatures of skyline groups serve as the summarization. Immediately, we know that the number of skyline groups is at most the number of skyline projections.

Practically, is the summarization using skyline groups meaningful? In practice, data is more or less correlated. Thus, objects may share values in some dimensions and form groups. In addition to capturing the semantics of skyline objects, skyline groups also summarize data records collapsing in some subspaces and appearing in some subspace skylines.

### 4.3 Skyline Groups and Skyline Objects

Although skyline groups provide a succinct summarization of the skylines in various subspaces, it still can be inconvenient and costly to manage all group members if many objects exist in a dataset. Can we select some representative objects from the skyline groups?

Encouragingly, we observe the following interesting fact.

THEOREM 4.3 (SKYLINE OBJECT). *For any skyline group $(G, \mathcal{B})$, there exists at least one object $u \in G$ such that $u$ is in the skyline of full space $\mathcal{D}$.*

PROOF. Let $p$ be an object in $G$ such that, in the full space $\mathcal{D}$, $p$ is not dominated by any other objects in $G$. In other words, $p$ is a skyline object with respect to objects in $G$. Clearly, for any $G \neq \emptyset$, we can find at least one object that is in the skyline with respect to objects in $G$. We show that $p$ is a skyline object in $\mathcal{D}$ with respect to the set of all objects $S$.

Suppose $p$ is dominated by $q \notin G$ in the full space $\mathcal{D}$, then two cases may arise. First, $p_{\mathcal{B}} = q_{\mathcal{B}}$, then $q \in G$, and it contradicts the assumption that $p$ is not dominated by any other objects in $G$. Second, if there exists a dimension $D_i \in \mathcal{B}$ that $p.D_i > q.D_i$, then given that $p$ is dominated by $q$ in the full space, $p_{\mathcal{B}}$ is dominated by $q_{\mathcal{B}}$. That leads to a contradiction to the assumption that $p_{\mathcal{B}}$ is in the subspace skyline. □

Theorem 4.3 indicates that the skyline objects in the full space play critical roles in the construction of subspace skylines, their projections are sufficient to represent the *contour* of the skyline, that is, the dimension values of the projections in the subspace skyline. In other words, an object that is not in the skyline of full space can be in the skyline of some subspace only if it collapses to some full space skyline object(s).

Please note that the other direction of Theorem 4.3 does not hold. Generally, a maximal c-group that is not a skyline group still may have a skyline object in the full space as a member. For example, in Figure 1, the group $(P_1P_5, B)$ is a maximal c-group and $P_1$ is a skyline object in the full space $(A, B)$, but the group itself is not a skyline group.

For a dataset $S$, we can obtain the set $SK$ of skyline objects in the full space. An object $p$ is in the skyline of subspace $\mathcal{B}$ in $S$ if and only if there exists an object $q$ that is in the skyline of the full space such that $p_{\mathcal{B}} = q_{\mathcal{B}}$ and $q$ is also in the skyline of the same subspace $\mathcal{B}$ in $SK$.

For example, on the dataset in Figure 1, $P_4$ is not in the skyline of the full space. However, it is in the skyline of subspace $A$ since it shares the same value on $A$ with $P_1$, a full space skyline object, and $P_1$ is also in the skyline of subspace $A$.

Moreover, we have the following result on the relationship between subspace skyline groups in a dataset and the subspace skyline groups in the subset of full space skyline objects.

THEOREM 4.4 (SKYLINE GROUP LATTICES AND FULL SPACE SKYLINE OBJECTS). *For a dataset $S$, let $SK$ be the complete set of skyline objects in the full space. Furthermore, let $\mathcal{SG}_S$ and $\mathcal{SG}_{SK}$ be the skyline group lattices on datasets $S$ and $SK$, respectively. Then, $\mathcal{SG}_{SK}$ is a quotient lattice of $\mathcal{SG}_S$.*

PROOF. According to Theorem 4.2, both $\mathcal{SG}_S$ and $\mathcal{SG}_{SK}$ are complete lattices. For any objects $p, q \in SK$, if $p$ and $q$ are in a skyline group $(G, \mathcal{B})$ in $\mathcal{SG}_S$, then we have $p_{\mathcal{B}} = q_{\mathcal{B}}$. Therefore, $p$ and $q$ must also be in the skyline on set $SK$. On set $SK$, let $G' = \mathcal{O}(\{p, q\}, \mathcal{B})$ and $\mathcal{B}' = \mathcal{I}(G')$. $(G', \mathcal{B}')$ is a maximal c-group. Since $p$ and $q$ are in the skyline of $\mathcal{B}$, $(G', \mathcal{B}')$ is also a skyline group on $SK$. According to Theorem 4.3, every skyline group on $S$ must contain at least one skyline object in the full space. Thus, any skyline group $(G, \mathcal{B})$ on $S$ can be mapped to a skyline group $(\mathcal{O}(\{p, q\}, \mathcal{B}), \mathcal{I}(G'))$ on $SK$.

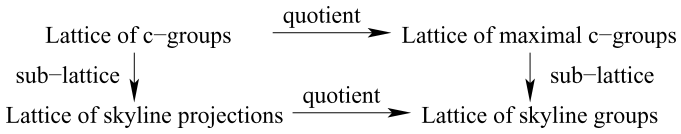



Fig. 4. Relationships of the four types of lattices.

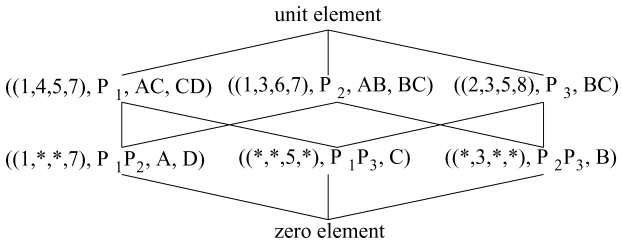



Fig. 5. Skyline group lattice for Figure 2(a).

On the other hand, for any skyline group $(G'', \mathcal{B}'')$ on $SK$, skyline group $(\mathcal{O}(G'', \mathcal{B}''), \mathcal{B}'')$ on $S$ is mapped to the group on $SK$. Thus, the mapping is a surjection from $\mathcal{SG}_S$ to $\mathcal{SG}_{SK}$. The claim that $\mathcal{SG}_{SK}$ is a quotient lattice of $\mathcal{SG}_S$ follows. □

On the other hand, if we are only concerned with the projections in the subspaces skyline, only the skyline objects in the full space are necessary for the analysis. In such a case, we do not need to manipulate all objects. This potentially leads to a significant reduction in the computational cost.

Theorems 4.3 and 4.4 immediately have two practicaly useful applications. First, they give rise to efficient algorithms for subspace skyline computation which will be discussed in Section 5. Second, they can also lead to a novel multidimensional analysis of subspace skylines which will be showcased in Section 4.4.

In summary, we discuss four types of lattices in this section, namely, the lattice of c-groups, the lattice of maximal c-groups, the lattice of skyline projections, and the lattice of skyline groups. Their relationships are summarized in Figure 4.

### 4.4 Multidimensional Analysis on Skylines

Since the skyline groups form a complete lattice, it is natural to introduce the multidimensional roll-up and drill-down analysis on skyline groups.

*Example* 8 (*Multidimensional Analysis*). Figure 5 shows the skyline group lattice for the objects in our running example (Figure 2(a)). For each node in the lattice, the projection, the skyline objects, and the decisive subspaces are shown.

By browsing Figure 5, the following structural information about the subspace skylines can be presented.

—Subspace skylines. The information is recorded in the signatures.

—Relationships between skylines in subspaces. For example, from the figure, we know that an object is in the subspace skyline of $C$ if it has value 5 on $C$.

There are two ways to further qualify the object as a skyline object in the full space: either having value 3 on $B$ (i.e., object $P_3$), or having value 1 on $A$ or 7 on $D$ (i.e., object $P_1$). The latter two values ($A = 1$ and $D = 7$) always come together.

—Closure information. From the figure, we can learn that it is impossible to have an object in the subspace skyline of $BCD$, but not in the subspace skyline of $ABCD$. While a naïve method to derive this information has to check all objects in the dataset, we derive this information from only the skyline groups.

In practice, why are such roll-up and drill-down operations useful? When a user examines subspace $C$, she finds that both $P_1$ and $P_3$ are subspace skyline objects. This is interesting to her, but she would like to find out further in what other subspaces $P_1$ is also good and is better than $P_3$. Then, she finds $AC$ and $CD$ and their super-spaces through a roll-up.

Clearly, the online roll-up and drill-down analysis is not available in the traditional skyline analysis.

## 5. MULTIDIMENSIONAL SUBSPACE SKYLINE COMPUTATION

Given a data set, the problem of *multidimensional subspace skyline computation* is to compute the skyline for each nonempty subspace. At the same time, we also compute the complete set of skyline groups and their signatures as the summarization of the skylines.

In this section, we first provide the theoretical analysis of SKYCUBE computation. Then, we identify a special condition, termed *distinct value condition*, under which the key ideas of our proposed SKYCUBE computation algorithms are easily illustrated. Finally, we present two novel SKYCUBE computation algorithms, Bottom-Up and Top-Down algorithms.

### 5.1 Complexity of SKYCUBE Computation

As shown in previous study, with the dimensionality fixed, the skyline computation is polynomial with respect to the number of objects. Thus, with the dimensionality fixed, the complexity of skyline cubes and skyline groups computation is also polynomial with respect to the number of objects.

Similar to data cube computation and many other OLAP problems, the major challenge is the dimensionality curse. In this section, we study the complexity of the computation with respect to the increase of the dimensionality.

The following theorems show that the problem of SKYCUBE computation is NP-hard.

THEOREM 5.1 (COMPLEXITY OF SKYCUBE COMPUTATION). *The problem of computing a* SKYCUBE *is NP-hard.*

PROOF. We prove the complexity by reducing the problem of mining frequent itemsets [Agrawal et al. 1993] to computing a SKYCUBE.

Let $\mathcal{ITEM}$ be a set of items. An itemset $X$ is a subset of $\mathcal{ITEM}$. A transaction database is a multiset of itemsets. For an itemset $X$ and a transaction database

$T$, the support of $X$ in $T$ is the number of transactions in $T$ that contain $X$ as a subset. For a given support threshold, an itemset is called frequent if its support passes the threshold. The problem of frequent itemset mining is to find all frequent itemsets.

To convert the problem, we create a database $DB$ of $|\mathcal{ITEM}|$ dimensions, such that a dimension is created for each item in $\mathcal{ITEM}$. Each transaction is transformed to an object in the following way: if item $x$ appears in the transaction, then the object takes value 0 on dimension $D_x$; otherwise, it takes value 1. We denote the object for transaction $t$ by $O(t)$. We also add one object $O_0$ into the database which takes value 0 in every dimension.

For any itemset $X = \{i_{j_1}, \ldots, i_{j_k}\}$, we consider subspace $\mathcal{S}(X) = (D_{j_1}, \ldots, D_{j_k})$. Clearly, transaction $t$ contains $X$ if and only if $O(t)$ is in the skyline of the subspace. In other words, the support of $X$ is the number of skyline object in subspace $\mathcal{S}(X)$ minus 1 since $O_0$ is always in the skyline of every nonempty subspace.

Apparently, the reduction is of linear time. As shown in Angiulli et al. [2004], mining categorical frequent itemsets is NP-hard. Thus, the problem of computing a SKYCUBE is also NP-hard. □

THEOREM 5.2 (COMPLEXITY OF SKYLINE GROUP COMPUTATION). *The problem of computing the complete set of skyline groups is NP-hard.*

PROOF. We reduce the problem of mining frequent closed itemsets [Pasquier et al. 1999] to computing the set of skyline groups. An itemset $X$ is closed if and only if there exists any proper superset $X' \supset X$ such that $sup(X) = sup(X')$. The problem of mining frequent closed itemsets was proved NP-hard by Yang [Yang 2004].

The problem transformation is the same as in Theorem 5.1. For any itemset $X = \{i_{j_1}, \ldots, i_{j_k}\}$, we consider subspace $\mathcal{S}(X) = (D_{j_1}, \ldots, D_{j_k})$. Clearly, transaction $t$ contains $X$ if and only if $O(t)$ is in the skyline of the subspace. In other words, the support of $X$ is the number of skyline object is subspace $\mathcal{S}(X)$ minus 1, since $O_0$ is always in the skyline of every nonempty subspace.

Let $\{t_1, \ldots, t_n\}$ be the set of transactions containing $X$. Consider object set $G = \{O(t_1), \ldots, O(t_n)\}$. Now, we only need to show $X$ is closed if and only if $(G, \mathcal{S}(X))$ is a skyline group.

All objects in $G$ share the same values on all dimensions in $\mathcal{S}(X)$. Moreover, there exists no dimension on which all objects in $G$ share a same value, otherwise $X$ is not closed. Therefore, $(G, \mathcal{S}(X))$ is a skyline group.

To show the other direction, assume $(G, \mathcal{S}(X))$ is skyline group, but $X$ is not closed. There must exist an itemset $X' \supset X$ such that $sup(X) = sup(X')$. That is, every transaction containing $X$ also contain $X'$. Under the problem transformation, all objects in $G$ also share the same values on dimensions in $\mathcal{S}(X' - X)$. That means $(G, \mathcal{S}(X))$ is not a skyline group, a contradiction.

The problem transformation is of linear time. Thus, the theorem is proved. □

## 5.2 Distinct Value Condition

Before we present our algorithms to compute the SKYCUBE and skyline groups for a dataset, we discuss the relationship of skyline objects between an

ancestor and a descendant subspace in this section. In particular, we identify the *distinct value condition*, under which the computations can be greatly simplified. We also identify a theorem which guides our algorithms to deal with the general case.

One difficulty of SKYCUBE computation lies in the fact that, in general, the skyline of an ancestor subspace does not contain that of a descendant subspace. That is, for two subspaces $\mathcal{U}$ and $\mathcal{V}$ such that $\mathcal{U} \subset \mathcal{V}$, $SKY_{\mathcal{V}}(S)$ does not contain *all* the skyline objects in $SKY_{\mathcal{U}}(S)$ ($S$ is the set of objects) as illustrated in Example 3 and Figure 2.

Nevertheless, we have identified Theorem 5.3 which accurately characterizes the relationship between skyline objects in the descendant subspace and its ancestor subspace.

THEOREM 5.3. *Given a set $S$ of objects in the full space $\mathcal{D}$, $\mathcal{U}$, and $\mathcal{V}$ are two subspaces where $\mathcal{U} \subset \mathcal{V}$. In subspace $\mathcal{V}$, each skyline object $q$ in $SKY_{\mathcal{U}}(S)$ is either dominated by another skyline object $p$ in $SKY_{\mathcal{U}}(S)$ such that $p_{\mathcal{U}} = q_{\mathcal{U}}$ (and thus $q$ is not a skyline object in subspace $\mathcal{V}$), or a skyline object in $SKY_{\mathcal{V}}(S)$.*

PROOF. For each skyline object $q$ in $SKY_{\mathcal{U}}(S)$, if there is another object $p$ such that $p_{\mathcal{U}} = q_{\mathcal{U}}$, $p$ may dominate $q$ in subspace $\mathcal{V}$ if $p$ dominates $q$ in the subspace $\mathcal{V} - \mathcal{U}$. Obviously, $p$ is a skyline object in $SKY_{\mathcal{U}}(S)$ as well. Otherwise, if such a skyline object $p$ does not exist, $q$ is a skyline object in $SKY_{\mathcal{V}}(S)$ because no other objects can dominate it in subspace $\mathcal{V}$. □

While Theorem 5.3 guides us in developing SKYCUBE and skyline group computation algorithms for the general case, we have also identified a special situation, termed as *distinct value condition* (defined in Definition 5.4) where the containment relationship between an ancestor and descendant subspace holds.

*Definition* 5.4 (*Distinct Value Condition*). In a given set $S$ of objects in the full space $\mathcal{D}$, the distinct value condition holds if for any two objects $p$ and $q$, $\forall D \in \mathcal{D}$, $p.D \neq q.D$.

The distinct value condition dictates that no two objects share the same projection in any subspace. The following two corollaries guarantee the containment relationship between an ancestor and a descendant subspace as well as the property of skyline groups under the distinct value condition.

COROLLARY 5.5 (CUBOIDS UNDER DISTINCT VALUE CONDITION). *Under the distinct value condition, given a set $S$ of objects, $SKY_{\mathcal{U}}(S) \subseteq SKY_{\mathcal{V}}(S)$, for any two subspaces $\mathcal{U}$ and $\mathcal{V}$ such that $\mathcal{U} \subset \mathcal{V}$.*

PROOF. We prove by contradiction. Suppose there exists an object $p \in SKY_{\mathcal{U}}(S)$ but $p \notin SKY_{\mathcal{V}}(S)$. There must be another object $q$ such that $q$ dominates $p$ in $\mathcal{V}$. In other words, for any dimension $D \in \mathcal{U} \subset \mathcal{V}$, $q.D \leq p.D$. Under the distinct value condition, $q.D \neq p.D$. Thus, $q.D < p.D$. That means $q$ dominates $p$ in $U$, which contradicts the assumption. □

COROLLARY 5.6 (SKYLINE GROUP UNDER DISTINCT VALUE CONDITION). *Under the distinct value condition, given a set of objects on the full space $\mathcal{D}$, for any subset*

*of objects* $G \neq \emptyset$,

$$\mathcal{I}(G) = \begin{cases} \mathcal{D} & \text{if } |G| = 1 \\ \emptyset & \text{if } |G| > 1 \end{cases}$$

*For any subset of objects* $G \neq \emptyset$ *and subspace* $\mathcal{B} \neq \emptyset$,

$$\mathcal{O}(G, \mathcal{B}) = \begin{cases} G & \text{if } |G| = 1 \\ \emptyset & \text{if } |G| > 1 \end{cases}$$

*For any skyline group* $(G, \mathcal{B})$, $|G| = 1$ *and* $\mathcal{B} = D$ *if* $\mathcal{B} \neq \emptyset$.

PROOF. The corollary follows the definitions of $\mathcal{I}(G)$ and $\mathcal{O}(G)$. □

Intuitively, with such good properties, SKYCUBE and skyline group computation under the distinct value condition is easier. For example, the skyline of a child subspace can be computed from that of its parent subspace instead of the original dataset which might be much larger. In the sequent sections, we will discuss algorithms that assume the distinct value conditions and then present modifications and extensions when such assumptions are dropped.

### 5.3 Bottom-Up SKYCUBE Computation

In this section, we present our Bottom-Up Skycube algorithm (*BUS*). The BUS algorithm computes the skyline results of all possible nonempty subspaces as well as the complete set of skyline groups and their signatures as the summarization of the skylines. As the BUS algorithm adopts the basic idea of nested-loop-based skyline algorithm to compute the skyline for each subspace, we first introduce the two classical nested-loop-based skyline algorithms: the Block-nested-loop (BNL) algorithm and the Sort-filter-skyline (SFS) algorithm.

For the ease of illustration of the BUS algorithm, we first assume that the distinct value condition holds. The BUS algorithm takes advantages of two computation sharing strategies: *sharing result* and *sharing sorting*. To save unnecessary pairwise comparisons between objects, a filter-based heuristic is developed. This heuristic greatly improves the performance of BUS, which is confirmed by our extensive experiment evaluations. Finally, we discuss the modifications on BUS to deal with cases where the distinct value condition does not hold.

5.3.1 *The BNL and SFS Algorithms.* To compute the skyline, BNL scans the dataset and compares each object $p$ with the current list of candidate skyline objects. Initially, the candidate list is empty. If $p$ is dominated by any object in the list, it is discarded. If $p$ dominates some of objects in the list, it is inserted into the list and all dominated objects are deleted from the list. If $p$ is neither dominated nor dominates any objects in the list, it is inserted into the list as a new candidate. After examining all the objects, BNL outputs all the candidates in the list as the skyline result.

In order to reduce the number of pairwise comparisons between objects in BNL, SFS introduces the *entropy value* [Chomicki et al. 2003]. The entropy value of an object $p$ on subspace $\mathcal{U}$ is $E_{\mathcal{U}}(p) = \sum_{\forall D_i \in \mathcal{U}} \ln(p'.D_i + 1)$, where $p'.D_i$ is the normalized value of $p.D_i$. Intuitively, the smaller the entropy value is,

the less likely the object is dominated by others. Based on this observation, SFS presorts the data set in nondecreasing order of the entropy values. Then, SFS examines the objects in this order in a similar way to BNL. As the objects with smaller entropy values are examined first, in general, skyline objects can be found earlier. Therefore, the number of unnecessary comparisons between objects and non-skyline objects in the candidate list is reduced.

5.3.2 *The Bottom-Up* SKYCUBE *Algorithm (BUS).* The basic idea of the BUS algorithm is to compute each cuboid in the SKYCUBE in a levelwise and bottom-up manner. Each cuboid is computed by a nested-loop-based algorithm similar to SFS [Chomicki et al. 2003]. Note that a naïve generalization of the SFS algorithm for the SKYCUBE computation does not have good performance for two reasons: (1) each cuboid is computed from the original data set independently; and (2) it requires sorting of the original dataset $(2^d - 1)$ number of times, as the orders among objects with respect to the entropy values defined vary according to the subspace in question.

In our BUS algorithm, we identify two computation-sharing strategies which address the first issue; to handle the second issue, we propose a sorting-and-filtering technique that effectively reduces the number of sorting operations from $(2^d - 1)$ to $d$. We also optimize our algorithm by employing a filtering function which avoids many dominance tests, which determine the dominance relationship between two objects by comparing the $d$ attribute values of them.

*Sharing Result.* According to Corollary 5.5, we can easily derive that the union of the child cuboids belongs to the parent cuboid. Therefore, during computing a cuboid, the objects, which are in one of its child cuboids, are guaranteed to be skyline objects. The advantages of this "result sharing" are twofold: on the one hand, it reduces the size of input to the individual skyline computation process; on the other hand, a fewer number of dominance tests are performed because those objects do not need to be examined again. We call this strategy *sharing result*.

In addition, since we compute the SKYCUBE in a levelwise and bottom-up fashion, we can exploit this result sharing on all the child cuboids of the cuboid in question. In other words, we can union all the child cuboids as the starting point of the computation of the parent cuboid. In terms of implementation, we use efficient bitmap operations so that the union can be done in linear time.

*Sharing Sorting.* To avoid the explosion of the number of sorting operations required by SKYCUBE computation based on the SFS algorithm, we propose to change the sorting criteria as follows: when computing the skyline on subspace $\mathcal{V}$, we accept input sorted on any dimension $D_i$ $(D_i \in \mathcal{V})$ (or in general, any $\mathcal{U} \subseteq \mathcal{V}$). Because of the nested loop nature of the skyline computation algorithms used in our BUS algorithm, changing the input sorting order only affects the performance, not the correctness. We make such a change for two reasons.

—It may reduce the number of sorting required to compute the SKYCUBE from $2^d - 1$ to $d$. Consequently, we only need to sort the source dataset $d$ times, once per dimension. Moreover, we do not keep $d$ copies of each object in the

dataset; instead, only the pointers to the objects are sorted and stored for $d$ times.

—Such input ordering still guarantees the correctness and efficiency of the block-nested loop-based skyline computation algorithm. An object $p$ cannot be dominated by another object $q$ if $p$ is ordered before $q$ and they do not share the same value on the dimension being sorted. The reason is that $p$ must have a smaller value than $q$ on at least one dimension, which makes $p$ before $q$ in the sorted list.

Furthermore, we complement this sorting scheme by a filtering process during the skyline computation which further reduces the cost of dominance test. We will cover this part shortly in Section 5.3.3.

In our implementation, we use the following heuristic: when computing the skyline on subspace $\mathcal{V}$, we always pick the input sorted on the dimension $D_i \in \mathcal{V}$ where the domain of $D_i$ is the largest among all $D_j \in \mathcal{V}$. We note that this heuristic is similar in spirit to that adopted in the bottom-up data cube computation algorithm [Beyer and Ramakrishnan 1999].

*BUS Algorithm.* Based on the previous two sharing strategies, we develop our BUS algorithm, which computes the SKYCUBE in a levelwise and bottom-up fashion. The algorithm is listed in Algorithm 1. To compute every cuboid $SKY_{\mathcal{V}}(S)$, it first computes a list of skyline objects based on the skyline objects in all of its child cuboids by the **UnionChildSkylines** method. Under the distinct value condition, this method simply unions the skylines of all child cuboids of $SKY_{\mathcal{V}}(S)$. We will discuss necessary modifications to this method in the general case in Section 5.3.5. BUS then examines each object in the nondecreasing order of their projection on dimension $D_i$. All objects in the core are skyline objects in $\mathcal{V}$. We also compare the objects not in the core against the skyline objects in the subspace $\mathcal{V}$ computed so far (Lines 6–7 in Algorithm 1). Otherwise it is compared with the current skyline to determine whether it is a new skyline object by calling the function **Evaluate**, that is, doing a *dominance test* by comparing the $d$ attribute values of two objects (Line 9).

**Algorithm 1. BUS ($S$)**

**Input:**
$S$: a set of $d$-dimensional objects
**Output:**
cuboid $SKY_{\mathcal{V}}(S)$ for every subspace $\mathcal{V}$;
the complete set of skyline groups and their signatures
**Description:**
1: sort $S$ on every dimension $D_i$ (in nondecreasing order) to form $d$ sorted lists $l_{D_i}$ $(1 \le i \le d)$
2: **for all** subspace $\mathcal{V}$ in a levelwise and bottom-up manner **do**
3: *Core* := **UnionChildSkylines**($\mathcal{V}$)
4: choose a sorted list $l_{D_i}$ $(D_i \in \mathcal{V})$ $\mathcal{V}$
5: **for all** object $q$ in $l_{D_i}$ **do**
6: **if** $q \in$ *Core* **then**
7: insert $q$ into $SKY_{\mathcal{V}}(S)$
8: **else**
9: **Evaluate**($q$, $SKY_{\mathcal{V}}(S)$)

Table I. An Example Dataset

| Objects | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $P_1$ | 4 | 3 | 2 | 2 |
| $P_2$ | 5 | 1 | 1 | 2 |
| $P_3$ | 1 | 4 | 4 | 1 |
| $P_4$ | 2 | 2 | 3 | 1 |

5.3.3 *Optimizing Dominance Test via Filtering.* The simplest implementation of the **Evaluate** function (Line 9) in Algorithm 1 is to perform a dominance test between $q$ and every skyline object in $SKY_{\mathcal{V}}(S)$. To further optimize the performance, our BUS algorithm is integrated with a filtering function, $f$, which drastically reduces the number of such dominance tests.

We define a *filter function* as a multivariate monotonic nondecreasing function that takes a multidimensional object as the parameter; the function value is called the *filter value* of the object. In the BUS algorithm, we use the following filter function defined on subspace $\mathcal{U}$ because it outperforms other filter functions in our experiments with various parameters.

$$f_{\mathcal{U}}(p) = \sum_{\forall D_i \in \mathcal{U}} p.D_i$$

For example, in Table I, $f_{ABCD}(P_1) = 11$, and $f_{ABCD}(P_4) = 8$. Based on the property of the multivariate monotonic nondecreasing function, it is easy to derive that, given two objects $p$ and $q$, if $f_{\mathcal{U}}(p) \leq f_{\mathcal{U}}(q)$, then $q$ does not dominate $p$ on subspace $\mathcal{U}$. In the previous example, based on their filter values, without an exhaustive comparison on every dimension, we know that $P_1$ cannot dominate $P_4$ on space $ABCD$ because the filter value of $P_4$ is smaller.

We note that the complexity of the filtering function is still $O(d)$ (where $d$ is dimensionality of the space) asymptotically; consequently, the complexity of calculating the filter values of all $n$ objects on every subspace of $d$-dimensional space is $O(nd\,2^d)$. Nevertheless, the filtering function accelerates the BUS algorithm significantly due to the following facts: (1) the lexicographical comparison between two objects on multidimensional space is more expensive than doing a comparison based on such a filter function; and (2) by maintaining the filtering values of skyline objects in sorted order, an incoming object, $p$, only needs to be compared to skyline objects whose filter value is smaller than $p$'s filter value. We found a 40% to 2700% speedup by adopting the filtering heuristics in our experiments.

Algorithm 2 presents the implementation of the **Evaluate** function using the filter-based heuristic. The algorithm requires us to maintain the skyline $SP$ such that objects inside are in a nondecreasing order of their filter values. When evaluating object $q$ against skyline object $p$, we first compare their filter values. If $q$'s filter value is smaller than $p$'s, $p$ and all the skyline objects after $p$ in the candidate list cannot dominate $q$. Therefore, it is known immediately that $q$ is a new skyline object. Otherwise, $p$ and $q$ are further compared on each dimension to determine whether $q$ is a new skyline object.

**Algorithm 2. Evaluate ($q$, $SP$)**

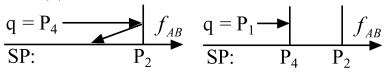
**Input:**
$q$: an object to be evaluated
$SP$: the skyline of subspace $\mathcal{U}$ computed so far (the full space is $\mathcal{D}$), maintained in the increasing order of filter values.

**Output:**
insert $q$ into $SP$ and construct/update the skyline group for $q$ if it is a skyline object of $\mathcal{U}$

**Description:**
1: **for all** object $p$ in $SP$ in the increasing order of their filter values **do**
2: **if** $f_{\mathcal{U}}(q) < f_{\mathcal{U}}(p)$ **or** $f_{\mathcal{U}}(q) < f_{\mathcal{U}}(\text{LastObj}(SP))$ **then**
3: insert $q$ into $SP$
4: **if** calculating skygroups **then**
5: **UpdateSkyGroup**($\{q\}$, $\mathcal{U}$) `/* to compute skyline groups, details will be covered in Section 5.3.5 */`
6: **break**
7: **else if** $(p \prec q)_{\mathcal{U}}$ **then**
8: discard $q$;
9: **break**

| Sort on B | $P_2$ | $P_4$ | $P_1$ | $P_3$ |
|---|---|---|---|---|
| $f_{AB}$ | 6 | 4 | 7 | 5 |

(a) Access Order and Filter Values




(b) Evaluate $P_4$ (c) Evaluate $P_1$

Fig. 6. Example of the filter-based heuristic.

Here is an example to illustrate the filter-based heuristic. Consider computing $SKY_{AB}(S)$ for the dataset shown in Table I. The objects are already sorted on dimension $B$. The access order and filter value of each object are listed in Figure 6(a). As $P_2$ is a skyline object of the child cuboid $SKY_B(S)$, it is directly inserted into the skyline $SP$. Figure 6(b) shows the scenario when $P_4$ is evaluated. Since $f_{AB}(P_4) < f_{AB}(P_2)$, $P_4$ is inserted into the $SP$ immediately without comparison with $P_2$ on subspace $AB$. Then, $P_1$ is evaluated (Figure 6(c)). $P_1$ is first compared with $P_4$. Since $f_{AB}(P_4) < f_{AB}(P_1)$ and $P_4$ dominates $P_1$, $P_1$ is discarded. Similarly, $P_3$ is evaluated. The skyline of subspace $AB$ is $\{P_4, P_3, P_2\}$.

5.3.4 *Computing the Skyline Groups under the Distinct Value Condition.* In this section, we consider computing skyline groups (and their signatures) assuming the datasets conforms to the distinct value condition. We shall drop this assumption and discuss the generalization needed to handle the general case in Section 5.3.5.

Recall that a skyline group is formed as $\langle G, \mathcal{B}, \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k \rangle$. Here, $G$ is a set of objects in the skyline group; $\mathcal{B}$ is the signature subspace such that all the objects in $G$ share the same projection on $\mathcal{B}$; and all $\mathcal{C}_i$ $(1 \leq i \leq k)$ are decisive spaces that informally indicate the skyline group. According to the definition of the decisive subspace, objects in $G$ are also in the skylines of the subspace $\mathcal{C}_i$.

A naïve way to compute the skyline groups once the SKYCUBE has been computed is to organize skyline objects in each of the subspaces $\mathcal{B} \subseteq \mathcal{D}$ according to their definition (i.e., Definition 3.3). For each object $p$ in the subspace skyline $SKY_{\mathcal{B}}(S)$, we find all objects $q \in S$, such that $p_{\mathcal{B}} = q_{\mathcal{B}}$. However, it is costly to compute the decisive spaces for the skyline group. This is because we have to check whether any of the subspaces of the subspace in question is decisive or not.

In the following, we will show an efficient method to compute the skyline groups within the BUS algorithm framework. The algorithm is based on the observation that decisive spaces of any skyline group in subspace $\mathcal{B}$ can be computed easily if we have computed all the skyline groups in all the subspace $\mathcal{B}' \subset \mathcal{B}$. As the BUS algorithm works in a levelwise and bottom-up manner, skyline group computation can therefore be easily supported by enhancing the BUS Algorithm.

Specifically, we can make the following proposition regarding the skyline groups under the distinct value condition.

PROPOSITION 5.7. *The skyline groups for a dataset satisfying the distinct value condition have the following properties:*

(1) *all skyline groups, $\mathcal{G}$, consist of only a single object and this object is a skyline object in the full space;*
(2) *the signature subspace $\mathcal{B}$ of all skyline groups is the full space $\mathcal{D}$.*

The proposition follows from Corollary 5.6.

5.3.4.1 *Modification to the BUS Algorithm.* In the enhanced BUS algorithm, we need to maintain an additional global hash table, $SkyGroups$, which associates skyline group ($G$) with its current signature subspace ($\mathcal{B}$) and decisive subspace(s) (i.e., $\mathcal{C}_1, \ldots, \mathcal{C}_k$). This choice is based on the observation that the signature subspace and decisive subspaces of the group may be updated every time the same group of objects is identified as skyline objects in a new subspace.

When computing skyline objects for single dimensions, according to Proposition 5.7, it is clear that all the skyline objects form skyline groups of their own, and the signature subspaces of such skyline groups are the full space $\mathcal{D}$. In addition, we also know that the current single dimension is one of the decisive subspaces of the skyline group. Our subsequent goal is to update and maintain the decisive subspaces for the groups in the subsequent skyline computation.

It can be shown that in the subsequent iterations, we only need to

—create new skyline groups;
—or update the signature subspace[2] and decisive subspaces for an existing group.

[2]Note that we can skip maintaining the signature subspace of skyline groups under the distinct value condition because it must be the full space $\mathcal{D}$. We still include this operation here for the sake of consistency.

**Algorithm 3. UpdateSkyGroup ($G$, $\mathcal{V}$)**

**Description:**

```
 1: if SkyGroups.has(G) then
 2:   Let ⟨G, B, C⟩ := SkyGroups[G]
 3:   B := B ∪ V
 4:   for all C_i in C do
 5:     contained := false
 6:     if V ⊇ C_i then
 7:       contained := true /* This happens in both the bottom-up and the
           top-down computation */
 8:       break /* goto Line 11 */
 9:     else if V ⊂ C_i then
10:       C := C − C_i /* This happens only in the top-down computation */
11:   if contained = false then
12:     C := C ∪ V
13:   SkyGroups[G] := ⟨G, B ∪ V, C⟩
14: else
15:   SkyGroups[G] := ⟨G, V, {V}⟩
```

These operations are captured in the **UpdateSkyGroup** function (Algorithm 3). This function is called from the BUS algorithm when a new skyline object of the current subspace is identified (Lines 2–5). Inside the **UpdateSkyGroup** function (Algorithm 3), we will either create a new sky group for this skyline object (Line 15), or update its existing skyline group information (Lines 2–13). Assume that the group is $\langle G, \mathcal{B}, \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k \rangle$. We add the fact that the same skyline group is still valid for skyline objects in the current subspace $\mathcal{U}$ by:

(1) updating the signature subspace from $\mathcal{B}$ to $\mathcal{B} \cup \mathcal{U}$. This is to record the fact that all the objects in the group coincide on all dimensions in $\mathcal{U}$. As an optimization for efficiency, this step can be skipped under the distinct value condition as the signature subspaces of all skyline groups are already the full space $\mathcal{D}$.

(2) inserting the current subspace $\mathcal{U}$ into the signature subspaces of the group if $\mathcal{U}$ is not a superset of an existing signature subspace (Lines 4–12). It is immediate from the definition of decisive space that if $\exists i, 1 \leq i \leq k$ such that $\mathcal{C}_i \subset \mathcal{U}$, then $\mathcal{C}_i$, rather than $\mathcal{U}$, is the decisive subspace for the group. Note that Lines 6–8 of Algorithm 3 are codes for skyline group computation in the top-down algorithm described later in Section 5.4; they will not be reached in the BUS algorithm.

5.3.5 *BUS in the General Case.* In this section, we extend the BUS algorithm to handle the general case where the distinct value condition does not hold.

*Computing the* SKYCUBE. The major challenge to compute the SKYCUBE in the general case is that, as stated in Section 5.2, only part of child cuboid $SKY_{\mathcal{U}}(S)$ belongs to the parent cuboid $SKY_{\mathcal{V}}(S)$ (See Example 3 and Figure 2).

Therefore, the **UnionChildSkylines** function in Algorithm 1 will not only union all the skyline objects in the child cuboids, but also calculate the skyline of

the union of objects to remove non-skyline objects in the parent cuboid $SKY_{\mathcal{V}}(S)$. Several optimizations are used to accelerate this step.

—According to Theorem 5.3, to examine object $q$ in $SKY_{\mathcal{U}}(S)$, we only need to compare $q$ with those object $p$ in $SKY_{\mathcal{U}}(S)$ whose projection on subspace $\mathcal{U}$ is identical to that of $q$, and the dominance test only needs to consider their projected values on the subspace $(\mathcal{V} - \mathcal{U})$. If such an object $p$ does not exist, $q$ is a skyline object of cuboid $SKY_{\mathcal{V}}(S)$.

—As the skyline objects in all subspaces are always maintained to have a non-decreasing order of their filter values, we only need to scan each child cuboid once and process skyline objects with the same filter values at a time.

—As a by-product of this computation, we can identify those objects that are definitely not the skyline objects in the parent cuboid $SKY_{\mathcal{V}}(S)$ (as they are dominated by other objects with the same filter values). These objects can be marked and will not enter the **FOR** loop in Lines 5–9 in Algorithm 1. This avoids redundant comparisons.

*Computing Skyline Groups.* The computation of skyline groups and their signatures is more complicated in the general case as well. Specifically, neither of the properties for the skyline groups and their signatures in the distinct value case (see Proposition 5.7) holds in the general case, as

(1) a skyline group, $G$, may consist of one or more than one object;

(2) the signature subspace $\mathcal{B}$ of all skyline groups may not be the full space $\mathcal{D}$.

Therefore, our tasks are to keep track of the changes of skygroups and maintain their signature subspace as well as decisive subspaces during the BUS computation.

The first issue is how to maintain skyline groups when integrating skyline objects from child cuboids in the **UnionChildSkylines** function. Depending on whether a skyline group $G$ in the child cuboid changes or not with respect to the parent cuboid $SKY_{\mathcal{V}}(S)$, we have two possible scenarios.

—All the objects in $G$ are still the skyline objects in the parent cuboid $SKY_{\mathcal{V}}(S)$. In this case, we only need to update signature subspace and decisive subspaces of the group by calling **UpdateSkyGroup(**$G$**,** $\mathcal{V}$**)** (shown in Algorithm 3).

—Some object in $G$ (that are skyline objects in the child cuboid $SKY_{\mathcal{U}}(S)$) is no longer a skyline object in the parent cuboid $SKY_{\mathcal{V}}(S)$. While the original skyline group $G$ remains the same, we need to construct a new skyline group $G' = G_{\mathcal{V}}$, set its temporary signature subspace as $\mathcal{V}$, and set its decisive space as $\mathcal{V}$.

The second issue is how to create new skyline groups for an object $p$ which has been identified as a skyline object in subspace $\mathcal{V}$ but not in any of the child subspace of $\mathcal{V}$ as identified in Lines 2–5 in Algorithm 2. The subtlety here is that we must construct a skyline group for all objects sharing the same projection on the current subspace $\mathcal{V}$ that is, we need to set $G = \mathcal{O}(p, \mathcal{V})$. As we process all the objects in the increasing order of their values on dimension $D_i$, we just search for such objects for all objects $q$ such that $q.D_i = p.D_i$. Note that the

| Subspace | Cuboid | Skyline Groups | | | | |
|---|---|---|---|---|---|---|
| A | { $P_3$} | < {$P_3$}, A, A > | | | | |
| C | { $P_2$} | < {$P_3$}, A, A > | < {$P_2$}, C, C > | | | |
| D | { $P_3$, $P_4$} | < {$P_3$}, A, A > | < {$P_2$}, C, C > | < {$P_3$, $P_4$}, D, D > | | |
| AC | { $P_1$, $P_2$, $P_3$, $P_4$} | < {$P_3$}, AC, A > | < {$P_2$}, AC, C > | < {$P_3$, $P_4$}, D, D > | < {$P_1$}, AC, AC > | < {$P_4$}, AC, AC > |
| AD | { $P_3$} | < {$P_3$}, ACD, A > | < {$P_2$}, AC, C > | < {$P_3$, $P_4$}, D, D > | < {$P_1$}, AC, AC > | < {$P_4$}, AC, AC > |
| CD | { $P_2$, $P_4$} | < {$P_3$}, ACD, A > | < {$P_2$}, ACD, C > | < {$P_3$, $P_4$}, D, D > | < {$P_1$}, AC, AC > | < {$P_4$}, ACD, AC > |
| ACD | { $P_1$, $P_2$, $P_3$, $P_4$} | < {$P_3$}, ACD, A > | < {$P_2$}, ACD, C > | < {$P_3$, $P_4$}, D, D > | < {$P_1$}, ACD, AC > | < {$P_4$}, ACD, AC > |

Fig. 7. Running BUS on the example dataset. (light shaded cells indicate the skyline groups were newly created dark shaded cells indicate the skyline groups were updated.)

signature subspace of the group is indeterminate until all the cuboids have been computed. However, it is easy to show that the signature space of a skyline group $(G, \mathcal{B})$ is the union of all subspaces of $\mathcal{B}$ on which $G$ are skyline objects. Similarly, the decisive subspaces are not known yet, although the current subspace, $\mathcal{V}$, is one of them. As the BUS algorithm traverses the cube lattice in a bottom-up manner, we can create a new skyline group with partial information about the signature subspace and decisive spaces and maintain them in the subsequent computation. Therefore, when a new skyline object $p$ is identified, we will create a new skyline group $\langle \mathcal{O}(\{p\}, \mathcal{V}), \mathcal{V}, \mathcal{V} \rangle$.

*Example* 9. Consider the example dataset shown in Table I. This dataset does not conform to the distinct value condition as there are more than one objects sharing the same value on dimension $D$. In this example, for ease of presentation, we ignore the dimension $B$. As a result, the full space is $\{A, C, D\}$. All cuboids and the snapshots of skyline groups are shown in Figure 7. We will focus on the computation of the skyline groups here.

Initially, there is no skyline group. $P_3$ is identified as a skyline object in subspace $A$. Since there is no skyline group identified by $P_3$, a new skyline group, $(\{P_3\}, A, A)$, is created.

Next we compute the skyline objects for subspace $AC$. After comparing objects with the same filter values in child cuboids $SKY_A(S)$ and $SKY_C(S)$, we find $P_3$ and $P_2$ to be the skyline objects of cuboid $SKY_{AC}(S)$. As a result, we call the **UpdateSkyGroup** function for them, knowing that the objects in the groups remains the same and only their signature subspaces and decisive subspaces need to be updated. We then evaluate the rest of the objects with a list of skyline objects computed so far with the help of the filter function. Eventually, we identify that both $P_1$ and $P_4$ are also skyline objects in subspace $AC$. Therefore, two new skyline groups are created for $P_1$ and $P_4$, respectively.

When we compute subspace $AD$, we need to filter skyline objects on the child cuboids $SKY_A(S)$ and $SKY_D(S)$ in the function **UnionChildSkylines**. We identify that although both $P_3$ and $P_4$ are skyline objects on $D$, only $P_3$ is the skyline object on $AD$. We only need to update skygroup information for (the existing group) $P_3$.

### 5.4 Top-Down SKYCUBE Computation

In this section, we present our Top-Down Skycube algorithm (*TDS*). TDS relies on a novel Shared-Divide-and-Conquer skyline algorithm (*SDC*), which adopts

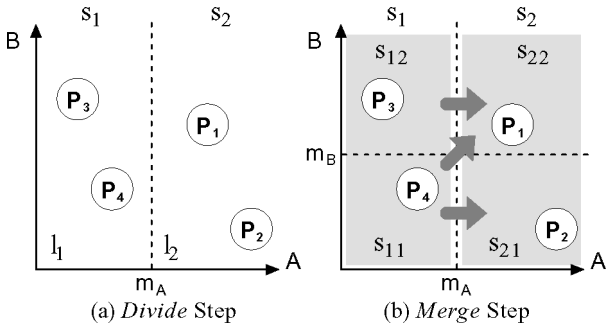



Fig. 8. Divide-and-conquer algorithm.

the basic idea of the Divide-and-Conquer skyline algorithm (*DC*) [Börzsönyi et al. 2001] while it computes multiple related skyline queries simultaneously with little additional overhead. In TDS, two new computation-sharing strategies: *sharing-partition-and-merging* and *sharing-parent* are developed.

In the following parts, we first illustrate the basic idea of the DC algorithm. Then, we propose the SDC algorithm, which is followed by the detailed TDS algorithm under the distinct value condition. Finally, we discuss the necessary modifications on the TDS algorithm in the general case.

5.4.1 *The Divide-and-Conquer Skyline Algorithm (DC).* To compute the skyline for a set of objects, DC first divides them into several parts and computes the skyline over each part. Then DC merges these skylines to obtain the final one. Consider the example to compute the skyline of subspace $AB$ on the dataset in Figure 8.

—First, DC calculates the median $m_A$ of all objects on dimension $A$ and divides the dataset into two parts, $l_1$ and $l_2$ (*divide* step). $l_1$ contains the objects whose values on dimension $A$ are less than $m_A$. $l_2$ contains all others.

—Then the skyline of $l_1$ ($l_2$) is computed. This is done by recursively applying the divide step until one object is left. In that case, to compute skyline is trivial. The skyline result $s_1$ ($s_2$) is shown in Figure 8(a).

—To obtain the overall skyline, DC eliminates the objects in $s_2$ which are dominated by those in $s_1$ (*merge* step). In the merge step, the median $m_B$ of objects in $s_1$ on dimension $B$ is calculated. $s_1$ and $s_2$ are further divided into $s_{11}$, $s_{12}$, $s_{21}$, and $s_{22}$ with $m_B$, shown in Figure 8(b). Clearly, the objects in $s_{21}$ have smaller value on dimension $B$ than that of objects in $s_{12}$. Therefore, the objects in $s_{21}$ are not dominated by any one in $s_{12}$. As a result, after further partition, DC only needs to merge $s_{11}$ and $s_{21}$, $s_{12}$ and $s_{22}$, $s_{11}$ and $s_{22}$, respectively (shown as arrows in Figure 8(b)). Each merge applies the merge step recursively until one object remains in either part, then to merge them is trivial. In this example, $P_1$ is eliminated from $s_2$ after merge. The final skyline of $AB$ is $\{P_3, P_4, P_2\}$.

5.4.2 *Computation Sharing Opportunities for the DC Algorithm.* Although DC is one of the most efficient skyline computation algorithms, it computes one cuboid only. However, we observe that both the divide step and the merge step

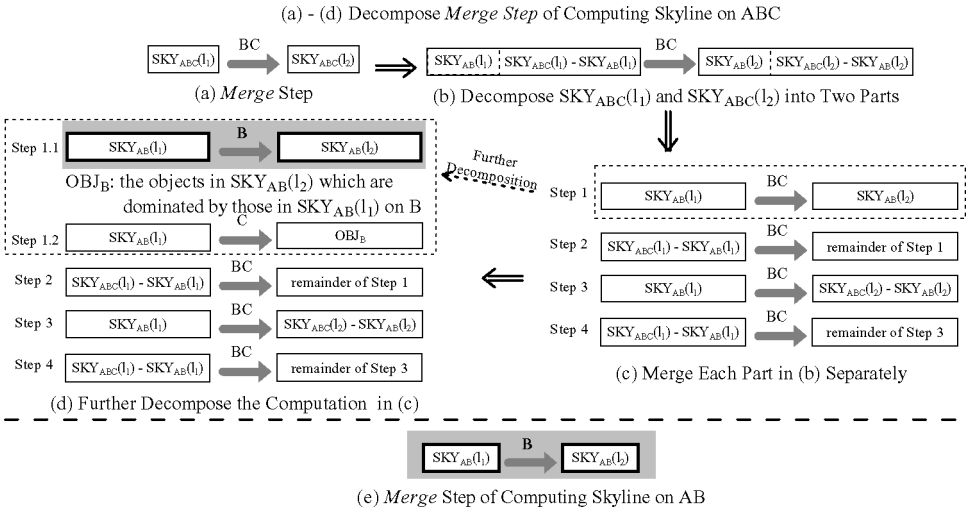



Fig. 9. Share merging. (In the figure, the merge operation indicated by the shaded background can be shared among the computation of cuboid $ABC$ and cuboid $AB$).

can be shared between a parent cuboid and its child cuboid. We call such sharing principles *sharing-partition-and-merging* collectively.

As described in detail in Section 5.4.1, the DC algorithm always chooses the median point with respect to the same dimension to divide the input set in the divide phase. We call such a dimension the *partitioning dimension* (e.g., $A$ is the example shown in Figure 8(a)). Obviously, to compute a skyline for child cuboid $SKY_{\mathcal{U}}(S)$ and parent cuboid $SKY_{\mathcal{V}}(S)$, if we divide the dataset using the same partitioning dimension, the partition results in both computations being the same. Hence, if we compute $SKY_{\mathcal{U}}(S)$ and $SKY_{\mathcal{V}}(S)$ simultaneously using the DC algorithm, the divide step can be shared.

In the *merge* step, DC merges the skylines of each part to obtain the final skyline. We observe that in this step the computation for the parent cuboid $SKY_{\mathcal{V}}(S)$ can be shared with its child cuboid $SKY_{\mathcal{U}}(S)$ ($\mathcal{U}$ is a prefix of $\mathcal{V}$). Therefore, the merge step for $\mathcal{U}$ can be saved.

Let us use the example in Figure 9 to illustrate the intuition of such sharing. We consider computing two cuboids on subspaces $ABC$ (Figure 9(a)–(d)) and $AB$ (Figure 9(e)), respectively. After the *divide* step with the partitioning dimension $A$, the dataset is partitioned into two parts $l_1$ and $l_2$. Then for the subspace $ABC$, the skyline of each part is calculated as $SKY_{ABC}(l_1)$ and $SKY_{ABC}(l_2)$. After that, DC needs to eliminate the objects in $SKY_{ABC}(l_2)$ which are dominated by those of $SKY_{ABC}(l_1)$ on subspace $BC$ as shown in Figure 9(a). According to Corollary 5.5, it is easy to derive that the skyline for the subspace $ABC$ contains that for the subspace $AB$. So, we split the skylines of $l_i$ ($i = 1, 2$) on $ABC$, e.g., $SKY_{ABC}(l_i)$, into two parts as $SKY_{AB}(l_i)$ and $SKY_{ABC}(l_i) - SKY_{AB}(l_i)$ (Figure 9(b)).

One key observation is that, after the split on both skyline objects, the merge operation can be decomposed into four steps, as shown in Figure 9(c). Step 1 is to eliminate objects in $SKY_{ABC}(l_2)$ which are dominated by $SKY_{ABC}(l_1)$. After Step 1, we call the remaining objects in $SKY_{ABC}(l_2)$ the *remainder* of Step (1).

Step (2) is to further eliminate the objects in the remainder of Step (1) which are dominated by $SKY_{ABC}(l_1) - SKY_{AB}(l_1)$. Step (3) and Step (4) follow the similar steps but are applied on $SKY_{ABC}(l_2) - SKY_{AB}(l_2)$.

In the previous decomposition, all the merge operations are performed on the subspace $BC$. The other key observation is that such dominance test can be further decomposed into checking dominance on dimension $B$ first, and then $C$. Applying this idea to Step (1) will result in Step (1.1) and Step (1.2), as shown in Figure 9(d). Step (1.1) is to merge $SKY_{AB}(l_1)$ and $SKY_{AB}(l_2)$ on the subspace $B$ (i.e., filter out objects in $SKY_{AB}(l_2)$ those are dominated by $SKY_{AB}(l_1)$ on dimension $B$). Instead of eliminating those objects, we keep them as $OBJ_B$. Then, in the next step, Step (1.2), each object $q$ in $OBJ_B$ is further examined against the objects $p$ in $SKY_{AB}(l_1)$ which dominate $q$ on the subspace $B$ whether $q$ is still dominated by $p$ on the subspace $C$. If so, $q$ is eliminated. All the remaining objects (in both Step (1.1) and Step (1.2)) are returned as the remainder of Step (1).

Now, if we consider the skyline computation on $AB$ using the same divide methods (i.e., obtain the same partition as $l_1$ and $l_2$), we need to merge $SKY_{AB}(l_1)$ with $SKY_{AB}(l_2)$ on dimension $B$. Clearly, this merge step is identical to Step (1.1) when computing the skyline on $ABC$ (see, the shaded areas in Figure 9(d) and (e)). This means that the merge step for the parent cuboid can be shared with that of its child cuboid.

By further generalizing this observation, we found that the divide steps and merge steps in DC can be shared by the two cuboids as long as the subspaces of one cuboid contains that of another (i.e., ancestor/descendant or parent/child relationship in the cube lattice). This motivates our SDC algorithm.

5.4.3 *Shared-Divide-and-Conquer Algorithm (SDC).* The basic idea of the Shared-Divide-and-Conquer algorithm (SDC) is to compute a number of related cuboids at a time-based on the aforementioned sharing principles. More specifically, SDC can compute a set of cuboids on a path in the cube lattice at a time. We call the set of subspaces associated with the cuboids *path of subspaces*, or simply *path* when there is no ambiguity. Without loss of generality, we can always represent a path in a canonical form by (1) arranging the subspaces in ascending order of their level values; and (2) two adjacent subspaces share the common prefix. For instance, the canonical form of the path $\{ABCD, ABD, AB\}$ is $\langle AB, ABD, ABDC\rangle$.

*Skylist.* The key to the SDC algorithm is a novel data structure, *skylist*, that concisely represents skylines for subspaces belonging to a path. Given a path $c$, a skylist consists of a number of elements, each of which stores the skyline objects for the corresponding subspace in $c$. Recall Corollary 5.5, for two subspaces $\mathcal{U}$ and $\mathcal{V}$, $SKY_{\mathcal{U}}(S) \subseteq SKY_{\mathcal{V}}(S)$ if $\mathcal{U} \subset \mathcal{V}$. A skylist organizes the skyline objects of each subspace in $c$ in the following accumulative way: the first element stores the skyline objects of the first subspace; and the $i$th element stores the difference between the skyline objects of the $i$th subspace and the $(i-1)$th one. For example, consider objects in Table I on subspace $ABC$ where the distinct value condition holds. For the path $c = \langle A, AB, ABC\rangle$, the skylist is $\langle\{P_3\}, \{P_2, P_4\}, \{P_1\}\rangle$.

The skylist data structure can benefit our SDC algorithm in the following ways. First, skylist stores the skyline objects for a path in a compact way which saves much storage space. Second, by storing the objects accumulatively, skylist enables our SDC algorithm to decompose the divide step and the merge step during the computation such that they can be shared by computing skylines for all subspace in the path simultaneously.

Here are some basic operations on the skylist data structure which are used in the SDC algorithm.

—*split($l$, $v$, $D_i$)*. Given a skylist $l$ and a value $v$ on dimension $D_i$, the split operation divides $l$ into two skylists $s_1$ and $s_2$. For each object $p$ in $l$, if $p.D_i < v$, it is moved to the corresponding element in $s_1$. Otherwise, $p$ is moved to the corresponding element in $s_2$.

—*union($s_1$, $s_2$)*. Unite the objects in the corresponding elements of the two skylists $s_1$ and $s_2$ to build a new skylist.

—*filter($c$, $s_1$, $s_2$)*. Given a path $c$ and two skylists, $s_1$ and $s_2$, the filter operation updates $s_2$ such that the remaining objects in $s_2$ are still skyline objects on the corresponding subspace even if $s_1$ and $s_2$ are united. Implementation-wise, for each object in every (the $i$th) element of $s_2$, if it is dominated by any objects in $s_1$ on the subspace corresponding to that element, it is moved to the next element of $s_2$.

For example, consider the skylists of the path $\langle A, AB\rangle$ on the top left ($\{P_3\}$) and the top right ($\{P_1\}$) datasets in Figure 8(b). Two skylists are constructed, $s_{12} = \langle\{P_3\}, \{\}\rangle$ and $s_{22} = \langle\{P_1\}, \{\}\rangle$. To filter $s_{22}$ by $s_{12}$, we first examine $P_1$ against $P_3$ on subspace $A$. As $P_1$ is dominated by $P_3$ on $A$, it is moved to the next element. Then the second element of $s_{22}$ is examined. Because $P_3$ does not dominate $P_1$ on subspace $AB$, $P_1$ is kept in the second element. So after filter, $s_{22} = \langle\{\}, \{P_1\}\rangle$.

*The SDC Algorithm.* To answer skyline queries on a path, SDC processes the objects in the divide-and-conquer manner similar to DC. The detailed SDC is presented in Algorithm 4. To compute skylines on a path $c$, an initial skylist $l$ corresponding to $c$ is constructed, where all objects are stored in the first element of $l$.

**Algorithm 4. SDC ($c$, $l$)**

**Input:**
$c$: a path, the last subspace in this path is $\mathcal{V}$
$l$: a skylist corresponding to $c$, initially, all objects are stored in the first element of $l$
**Output:**
a skylist corresponding to $c$ stores the skyline result
**Description:**

```
1: if |l| = 1 then /* |l|: the number of objects in l */
2:    return l
3: else
4:    D1 := the first dimension of V
5:    mD1 := the median of l on dimension D1
6:    (l1, l2) := split(l, mD1, D1)
7:    s1 := SDC(c, l1)
8:    s2 := SDC(c, l2)
```

```
 9:    s3 := SDC_Merge(c, s1, s2, 2)
10:    return union(s1, s3)
```

**Algorithm 5. SDC_Merge** ($c$, $s_1$, $s_2$, $i$)

**Input:**
$c$: a path, the last subspace in this path is $\mathcal{V}$
$s_1$, $s_2$: two skylists
$i$: the $i$th dimension of subspace $\mathcal{V}$ is used to split $s_1$ and $s_2$

**Output:**
the revised $s_2$ that eliminates the object in each element which is dominated by those in $s_1$ on the corresponding subspace in $c$

**Description:**

```
 1:  if |s1| = 1 or |s2| = 1 or i = |V| then /* |V|: the number of dimensions in V */
 2:    return filter(c, s1, s2)
 3:  else
 4:    Di := the ith dimension of V
 5:    v := the median on dimension Di
 6:    (s11, s12) := split(s1, v, Di)
 7:    (s21, s22) := split(s2, v, Di)
 8:    r1 = SDC_Merge(c, s11, s21, i)
 9:    r2 = SDC_Merge(c, s12, s22, i)
10:    r3 = SDC_Merge(c, s11, r2, i + 1)
11:    return union(r1, r3)
```

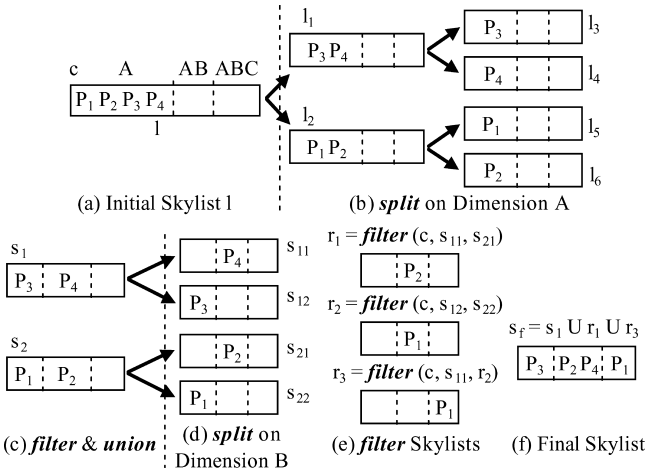



Fig. 10. An example of the SDC algorithm.

We use the following example to further illustrate SDC. Consider the skyline queries on a path $c = \langle A, AB, ABC \rangle$ over the dataset in Table I. Initially, a skylist $l$ is constructed according to $c$, and all objects are stored in the first element of $l$ as shown in Figure 10(a). After a recursive split on dimension $A$, $l$ is split into 4 skylists, each of which contains one object only (Figure 10(b)). Then we merge the skylist $l_3$ and $l_4$ first. As both of them contain only, one object in this merge, we directly call filter operation on them. In the filter process, $P_4$ is compared

to $P_3$. Since $P_3$ dominates $P_4$ on subspace $A$ and does not on subspace $AB$, $P_4$ is moved to the second element. Then, we union them, and the merge result $s_1$ is shown in Figure 10(c). Similarly, $l_5$ and $l_6$ are merged and the result is $s_2$. In order to merge $s_1$ and $s_2$ to obtain the final skylines, we split $s_1$ and $s_2$ with the median of objects in $s_1$ on dimension $B$. Figure 10(d) shows the split results. After that, three filter operations are processed. The filter results $r_1$ and $r_3$ are shown in Figure 10(e). Finally, we union $s_1$ and the merge results in the final skylist. To retrieve the skylines of all the subspaces in $c$, we use the procedure described in Algorithm 6. Essentially, we decompress the skylines of the $i$th subspace by unioning objects in all the preceding elements in the final skylist. For example, $SKY_A(S) = \{P_3\}$ and $SKY_{AB}(S) = \{P_3, P_2, P_4\}$.

5.4.4 *The Top-Down Skyline Algorithm (TDS).* In this section, we present our TDS algorithm which employs the SDC algorithm to compute SKYCUBE in a top-down manner. The pseudocode of the TDS algorithms is listed in Algorithm 7.

**Algorithm 6. RetrieveSkylist ($l_i$)**

**Input:**
$l_i$: a skylist
**Output:**
$SKY_{\mathcal{V}}(S)$ for every subspace $\mathcal{V}$ in the path $c_i$ corresponding to the skylist $l_i$
**Description:**
1: **for all** element $e_i$ (from the head to the tail) in the skylist $l_i$ **do**
2: $\mathcal{V}_i$ := the subspace corresponding to $e_i$
3: **if** $e_i$ is the first element **then**
4: $SKY_{\mathcal{V}_i}(S)$ := all objects in $e_i$
5: **else**
6: $SKY_{\mathcal{V}_i}(S)$ := **UnionAndFilter**($SKY_{\mathcal{V}_{i-1}}(S)$, all objects in $e_i$) `/* Perform set union when distinct value condition holds; otherwise, need to perform filtering after union. */`

**Algorithm 7. TDS ($S$)**

**Input:**
$S$: a set of $d$-dimensional objects
**Output:**
cuboid $SKY_{\mathcal{V}}(S)$ for every subspace $\mathcal{V}$;
the complete set of skyline groups and their signatures
**Description:**
1: compute the minimal set of paths that cover all the subspaces
2: **for all** path $c_i$ in the path set **do**
3: **SDC**($c_i, l_i$) `/* ` $l_i$ `is the result skylist corresponding to the path` $c_i$ `*/`
4: **RetrieveSkylist**($l_i$)
5: **if** calculating skygroups **then**
6: **for all** object $p$ in each cuboid $SKY_{\mathcal{V}}(S)$ where $\mathcal{V}$ is in the path $c_i$ **do**
7: **UpdateSkyGroup**($\{p\}$, $\mathcal{V}$) `/* Skyline groups are maintained in a global hash table */`

As analyzed in Section 5.4.3, SDC can compute subspace skylines on a path simultaneously. Thus, in order to compute SKYCUBE using SDC efficiently, we

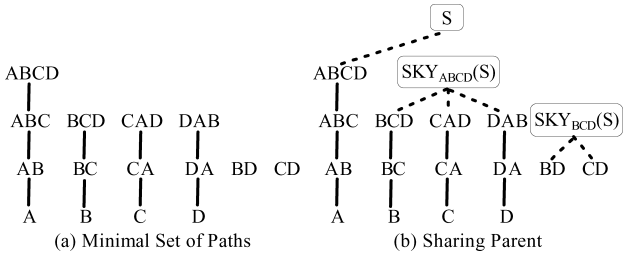



Fig. 11. An example of the TDS algorithm.

need to find a minimal set of paths such that every node in the cube lattice is covered by one path. In TDS, we apply the path finding algorithm in Ross and Srivastava [1997] to find such minimal sets of paths. In Ross and Srivastava [1997], it is shown that, on $d$-dimensional full space, the size of the minimal set of such paths is $\binom{d}{\lceil d/2 \rceil}$. Figure 11(a) shows a minimal set of paths for a 4-dimensional dataset.

After finding a minimal set of paths (Line 1 in Algorithm 7), TDS calls SDC to compute cuboids on each of the paths and computes the skyline groups on the skyline results of each path. Note that we only need to invoke SDC for the top cuboid of every path once, and all cuboids on the path can be computed with minimal additional cost. We adopt a further optimization here: when calling SDC on a path $c_i$, instead of computing the skylists from the original dataset $S$, we feed the already-compute cuboid $SKY_X(S)$ if $SKY_X(S)$ is the ancestor of the top cuboid along the path $c_i$. For example, in Figure 11(b), the computation of the second, $\langle B, BC, BCD \rangle$, is based on the cuboid of $SKY_{ABCD}(S)$. The correctness of this optimization follows directly from Corollary 5.5. In practice, this leads to a substantial saving of computation because (1) the whole dataset might be much larger than its skyline on a subspace; and (2) the reduction in the input size is most beneficial to the divide-and-conquer style algorithms as its time complexity is $O(n \log^{d-2} n)$. We call this optimization the *sharing parent* sharing strategy.

5.4.5 *TDS in the General Case.* Now we address necessary modifications to TDS in order to deal with the general case scenarios.

*Computing the* SKYCUBE. In the general case, there is no containment relation between a parent cuboid and its child cuboid. As a result, the straightforward sharing parent strategy may miss some skyline objects. For example, for the dataset shown in Table I, if we compute skylines for the path $\langle D, DB \rangle$ from the skyline of subspace $BCD$(i.e., $\{P_2, P_4\}$), we will miss the object $P_3$, which is the skyline object in the subspace $D$. Nevertheless, according to Theorem 5.3, these missing objects (e.g., $P_3$) must have the same values on some dimensions as some skyline objects of the parent cuboid. Therefore, the modification to the TDS algorithm is that, when invoking SDC for a path, the union of the chosen ancestor cuboid and those missing objects are passed as input. These missing objects can be easily retrieved as we have presorted the dataset on every single dimension.

Another modification is related to the restoration of skyline objects from skylists, that is, the **UnionAndFilter** function in Algorithm 6. Under the distinct value condition, for the skyline queries on a path $c$, SDC returns a skylist. To retrieve skyline for the $i$th subspace in $c$, the objects in all the preceding (i.e., $0 < j \leq i$) elements in the skylist are also written to the output. However, this again does not hold in the general case. According to Theorem 5.3, in the general case, the objects in the $j$th ($j < i$) element might not be the skyline objects of the $i$th subspace in $c$. As a result, we cannot output them directly (as Lines 4 and 6 in Algorithm 6). In order to determine whether these objects are still the skyline objects of the $i$th subspace in $c$ efficiently, we take advantages of their filter values as is done similarly in the BUS algorithm. More specifically, objects in each element are maintained in the nondecreasing order of their filter values. Then, the same method to union the results of child cuboids in the general case of the BUS algorithm is adopted to obtain the skyline result.

*Computing Skyline Groups.* The method to compute skyline groups in addition to the SKYCUBE is similar to that in the BUS algorithm. The key procedure is the **UpdateSkyGroup** function (Algorithm 3). This is shown in Lines 5–7 in Algorithm 7: when skyline groups computation is needed, all skyline objects are checked against existing skyline groups via the **UpdateSkyGroup** function. However, since we do not compute all the descendant cuboids before computing the cuboid in question, it is possible that the subspace, $\mathcal{V}$ of the current cuboid $SKY_{\mathcal{V}}(S)$ might be a subset of an existing decisive space. For example, as shown in Figure 11, when computing skyline objects in subspace $ABC$, skyline objects in subspace $C$ has not been computed yet. This means there might be some *false* decisive subspaces associated with the current skyline groups. Lines 9–10 in Algorithm 3 are designed to remove such false decisive subspaces. It invalidates all decisive subspaces that are a proper superset of the current subspace $\mathcal{V}$ on which all objects in the group $G$ are skyline objects.

### 5.5 Update Maintenance

Our work in this article is focused on static datasets. To maintain the efficiency of our techniques, we handle occassional updates in a straightforward way. We rerun our SKYCUBE computation algorithms to generate the updated SKYCUBE from scratch. However, as stated before, since several sharing strategies are employed in our algorithms, we do not have to recompute the complete SKYCUBE for the updates. Specifically, in the BUS algorithm, when computing one cuboid $\mathcal{U}$ during recomputation, if all its child cuboids $\mathcal{V}$ are not changed after recomputation, it is immediately clear that the cuboid $\mathcal{U}$ will also not be changed. Therefore, recomputation of this cuboid $\mathcal{U}$ is saved. Similarly, during recomputation in the TDS algorithm, if the input of the SDC algorithm for a path is the same as the one before the updates, none of the all the cuboids in this path will be changed. Consequently, the recomputation of this path is skipped.

Very recently, Xia and Zhang [2006] proposed a new structure, *Compressed SkyCube* (CSC), to compute a skyline cube against dynamic datasets (i.e., data objects are frequently updated). Instead of storing the complete SKYCUBE results

(i.e., all subspace skylines), CSC represents the SKYCUBE in a concise way: (1) for each skyline object $p$, it is only stored in the minimum subspaces, $mss(p)$, which is a set of subspaces, such that $\forall \mathcal{U} \in mss(p)$, $p \in SKY_{\mathcal{U}}(S)$ and $\forall \mathcal{V} \subset \mathcal{U}$, $p \notin SKY_{\mathcal{U}}(S)$; and (2) only the nonempty cuboids $\mathcal{U}$ (i.e., $\mathcal{U} \in mss(p)$, where $p$ is some skyline object) are kept in CSC. Then, an object-aware update schema is developed to effectively examine and update this small portion of SKYCUBE maintained in CSC upon the updates of the dataset. Refer to Xia and Zhang [2006] for more details.

## 6. RELATED WORK

To the best of our knowledge, Pei et al. [2005] and Yuan et al. [2005] are the first studies on the semantics and structure of subspace skylines and their computation. However, as two initial and independent studies, the two articles do not propose comprehensive solutions to efficient computation of skyline cubes, skyline groups, and their decisive subspaces.

In this section, we review some related work on skyline query processing and formal concept analysis and its applications in multidimensional data analysis as well as data cubes.

### 6.1 Skyline Query Processing

The problem of finding a skyline is a typical type of multiobjective query processing [Balke and Güntzer 2004]. It is first investigated in Kung et al. [1975] where an $O(n \log^{d-2} n)$ time algorithm for $d \geq 4$ and an $O(n \log n)$ time algorithm for $d = 2, 3$ are proposed. An expected linear running time algorithm is presented in Bentley et al. [1978] if the data distribution on each dimension is independent. Bentley et al. [1978] also estimates that the expected number of skyline objects under the independent distribution assumption is $O(\ln^{d-1} n)$. In Buchta [1989], the estimation is improved to $\Theta((\ln^{d-1} n)/(d-1)!)$.

Since Börzsönyi et al. [2001] introduced skyline operator in the database context, numerous skyline computation algorithms have been developed. Most of these algorithms can be classified into three categories: nested-loop-based, divide-and-conquer-based, and index-based.

*The Block-nested-loop* (BNL) algorithm [Börzsönyi et al. 2001], *Sort-filter-skyline* (SFS) [Chomicki et al. 2003], and (LESS) belong to the first category. Börzsönyi et al. [2001] also presents an algorithm (DC) based on the divide-and-conquer technique. As our two SKYCUBE computation techniques are developed based on these three algorithms, we have described them in detail in Sections 5.3.1 and 5.4.1, respectively.

Index-based algorithms include the *Index method* [Tan et al. 2001], the *Bitmap* method [Tan et al. 2001], *Nearest neighbor search* (NN) [Kossmann et al. 2002], and *Branch-and-bound skyline* (BBS) [Papadias et al. 2003]. Unlike nested-loop-based algorithms which have to visit the entire dataset, index-based algorithms only need to access a portion of the dataset to compute the skylines.

The Index method [Tan et al. 2001] organizes the $d$-dimensional dataset into $d$ $B^+$-Tree indexes, $T_1$, ..., $T_d$. Object $p$ is assigned to the index $T_i$ if $p.D_i$ is the

minimum value among all $d$ dimensions. To progressively compute skylines, in each iteration, the index method processes the object $p$ with the minimal value among all unprocessed objects. If the minimum value of $p$ is greater than the minimum of all the maximum values of each current skyline object, $p$ and all unprocessed objects in the same index are pruned safely. Otherwise, $p$ is evaluated against all the current skyline objects to determine whether it is a new skyline object.

The Bitmap method encodes an object with a bit vector according to its rank on each dimension. The bitmaps enable the algorithm to efficiently determine whether an object is a skyline object by bitwise AND/OR operations. One issue with the method is that the bitmaps can be prohibitively large when the number of distinct values on each dimension is large.

Both NN and BBS are based on nearest neighbor search, assuming the dataset is indexed by an R-Tree. Since BBS is I/O optimal and outperforms NN, we focus on the BBS method here. BBS traverses the R-Tree in an optimal order: it always evaluates and expends the node that is closest to the origin among all unvisited nodes. To do that, a heap is used. The key of this heap is the minimum distance between a node and the origin. Initially, BBS inserts the root of the R-Tree into the heap. In each iteration, the top element $e$ of the heap is removed and examined against the skyline computed so far. If $e$ is not dominated by any current skyline objects, either $e$ is output as a new skyline object (if $e$ is an object); or the child nodes of $e$, which are not dominated by any current skyline objects, are inserted into the heap (if $e$ is an intermediate node). BBS ends when the heap is empty.

Godfrey et al. [2005] surveyed the runtime complexities of existing generic skyline algorithms and introduced a new generic skyline algorithm, *Linear-elimination-sort for skyline* (**LESS**) that has $O(dn)$ average case running time, where $d$ and $n$ are dimensionality and cardinality of the date set, respectively. The basic idea is to improve the SFS algorithm by having a tight integration of external sort merge and skyline computation.

Most recently, several other research work stdy skyline computation in various applications, such as computing a skyline in the distributed database systems [Balke et al. 2004], continuous skyline queries in a data stream environment [Lin et al. 2005], approximate skyline computation [Koltun and Papadimitriou 2005], computing skyline in partially-ordered domains [Chan et al. 2005], computing skyline cubes in dynamic datasets [Xia and Zhang 2006].

### 6.2 Formal Concept Analysis and Its Applications in Multidimensional Data Analysis

The maximal c-group lattice is levered by the formal concept analysis [Ganter and Wille 1996]. However, unlike previous studies on database and data mining studies using formal concept analysis, such as Pasquier et al. [1999] and Lakshmanan et al. [2002], we are interested in a small part of the lattice—only the skyline groups. Moreover, the challenge is how to compute the skyline groups and the objects in the subspace skylines. These issues are far beyond the traditional studies on formal concept analysis.

**BUS** Our Bottom-Up SKYCUBE algorithm.

**BNLS** The algorithm computes each skyline query by BNL algorithm [Börzsönyi et al. 2001].

**SFSS** The algorithm computes each skyline query by SFS algorithm [Chomicki et al. 2003].

**TDS** Our Top-Down SKYCUBE algorithm.

**DCS** The algorithm computes each skyline query by DC algorithm [Börzsönyi et al. 2001].

There have been some recent studies on finding the determinant factors in multidimensional subspaces for some critical data. Two typical examples are Knorr and Ng [1999] and Lakshmanan et al. [2002]. In [Knorr and Ng 1999], the outhors propose an approach to finding the minimal sets of factors that explain the distance-based outliers. In Lakshmanan et al. [2002], the quotient cube lattice is developed to identify groups of aggregates that share the same set of base tuples in a data warehouse and thus the semantics of the aggregates can be explained and summarized concisely. While the philosophy in this research shares some similarity with these studies, the technical problems and the approaches are essentially different.

This study is also related to data cube computation. The concept of data cube was first proposed in Gray et al. [1996]. Efficiently computing data cubes has been an active research topic. A number of techniques have been reported in the literature [Gray et al. 1996; Agarwal et al. 1996; Ross and Srivastava 1997; Zhao et al. 1997; Beyer and Ramakrishnan 1999; Xin et al. 2003; Feng et al. 2004; Han et al. 2001; Lakshmanan et al. 2002; Lakshmanan et al. 2003]. Specifically, several heuristics for computing multiple group-bys (i.e., cuboids) efficiently have been identified, such as *smallest-parent*, *cache-results*, *amortized-scans*, *share-sorts*, and *share-partitions* [Sarawagi et al. 1996].

## 7. EXPERIMENTAL EVALUATION

In this section, we present the comprehensive performance evaluations of our techniques. As mentioned earlier, there is no existing technique specifically designed to support efficient SKYCUBE computation and skyline group computation. In our performance study, we implement some skyline algorithms (e.g., BNL, SFS, and DC) to compute each skyline and the skyline groups independently and use them as benchmark algorithms to evaluate our techniques. Below are the algorithms that have been evaluated.

Similar to BUS, BNLS and SFSS compute the SKYCUBE in the bottom-up manner. We also apply the sharing result strategy to both of them to share the child cuboid for computing the parent cuboid. DCS computes the SKYCUBE in the top-down manner. The sharing parent strategy is applied to DCS as well, which enables DCS to compute each cuboid from one of its parent cuboids instead of the whole dataset. To maintain the skyline group information during SKYCUBE computation, similar methods to **UpdateSkyGroup** are used in BNLS, SFSS, and DCS.

In the following sections, we first evaluate the meaningfulness of skyline groups and their signatures. Then, we study the overall performance of the BUS and the TDS algorithms, which includes the sensitivity and scalability against the data distribution, dimensionality, and cardinality. We also evaluate the efficiency of the filter-based heuristic to BUS and the sharing parent strategy to TDS, respectively. Finally, the effect of the number of duplicate values per dimension on our techniques is studied.

In our experiments, we employ both three real datasets and the three most popular synthetic benchmark datasets, *correlated*, *independent*, and *anti-correlated* [Börzsönyi et al. 2001], with dimensionality $d$ in the range $[4, 10]$ and cardinality $n$ in the range $[100k, 500k]$. For each experiment, we measure the query execution times of different algorithms for SKYCUBE *and* skyline group computation on these datasets, unless explicitly stated otherwise. All the experiments were carried out on a Pentium 4 PC with a 2.8GHz processor and 1GB main memory.

### 7.1 Results on Real Dataset Great NBA Players' Statistics

We downloaded from the NBA official Web site (www.nba.com) the Great NBA Players' technical statistics from 1960 to 2001. There are 17,266 total records. Each record is the statistics of a player in a season. We selected four attributes: the number of games played (GP), total points (PTS), total rebounds (REB), and total assists (AST). In this dataset, the larger the attribute values, the better. That is, player $A$ dominates player $B$ if $A$'s attribute values are not less than $B$'s, and $A$ has at least one attribute better than $B$.

Finding the skyline in these players' statistics dataset makes excellent sense in practice. People are often interested in finding the skyline players—players who have some outstanding merits that are not dominated by some other players. Moreover, finding the semantics of skyline in this application is of great interest; we not only want to know who the great players are, but we also want to know exactly on which combinations of factors a player is dominates the other players.

The knowledge of subspace skylines has immediate applications. For example, if a coach wants to find a player good at total points and total rebounds, he should look at the skyline players in the subspace $(PTS, REB)$, instead of all skyline players.

In this dataset, we found 67 skyline records in the full 4D space. The total number of corresponding decisive subspaces is 146, and the average dimensionality of the decisive subspaces is 2.21. We list some skyline players and their decisive subspaces in Table II.

The first four records are in the skyline since each of them takes the maximum value in one dimension. Interestingly, Wilt Chamberlain's performance in 1961 was outstanding in some combinations of attributes. Michael Jordan's performance in 1988 was not exceptional in terms of any single attribute. However, it is in the skyline once attribute combinations (PTS, REB, AST) or (GP, PTS, AST) are considered. Gary Payton's performance in 2001 is in the skyline only if all the attributes are considered. Clearly the decisive subspaces provide more insightful information than just the list of skyline players in the full space.

Table II. Some Skyline Players and the Corresponding Decisive Subspaces

| Player | GP | PTS | REB | AST | Decisive subspaces |
|---|---|---|---|---|---|
| Wilt Chamberlain (1960) | 79 | 3033 | 2149 | 148 | (REB) |
| Wilt Chamberlain (1961) | 80 | 4029 | 2052 | 192 | (PTS), (GP, REB), (REB, AST) |
| Chuck Williams (1973) | 90 | 1113 | 250 | 557 | (GP) |
| John Stockton (1990) | 82 | 1413 | 237 | 1164 | (AST) |
| Michael Jordan (1988) | 81 | 2633 | 652 | 650 | (PTS, REB, AST), (GP, PTS, AST) |
| Gary Payton (2001) | 82 | 1815 | 396 | 737 | (GP, PTS, REB, AST) |

Table III. Number of Skyline Players in Subspaces with Different Dimensionality

| Dimensionality | # of players |
|---|---|
| 1 | 4 |
| 2 | 41 |
| 3 | 102 |
| 4 | 67 |

We found skyline records in all nonempty subspaces. Some of them may not be skyline records in the full space. The numbers of subspace skyline groups are listed in Table III. These numbers can be explained by the different number of subspaces associated with the given dimensionality and by the fact that the number of skyline records increases with increasing dimensionality.

We also counted the total number of subspaces where a record is in the skyline. This is an interesting measure. Intuitively, if a player is in the skylines of more subspaces, he has a better overall capability in terms of combinations of attributes. We found that, in addition to dominating all others in total points (PTS), Wilt Chamberlain's performance in 1961 has the highest number, 13, of subspaces where it is in the skyline. On the other hand, although John Stockton's performance in 1990 dominates all others in total assists (AST) which is decisive, it is only in the skyline of 5 subspaces.

From the analysis on the real dataset, we obtained some interesting and meaningful observations that cannot be derived from the traditional skyline computation. This demonstrates the meaningfulness of the proposed subspace skyline analysis.

### 7.2 Effect of Dimensionality

In this section, we study the effect of dimensionality on our techniques. Correlated, independent, and anticorrelated datasets with dimensionality $d$ between 4 to 10 and cardinality $n = 500k$ are used in this experiment. Figure 12 shows the times of all the algorithms.

From the results, it is clear that although SFSS and BNLS adopt the sharing result strategy, their performances are worse than that of BUS in all datasets. This is because both insert some non-skyline objects into the candidate list which leads to more unnecessary pairwise comparison. Due to $O(2^d)$ presorting overhead, SFSS has the worst performance among all the algorithms. We do not plot the results of SFSS and BNLS in high-dimensional datasets because their

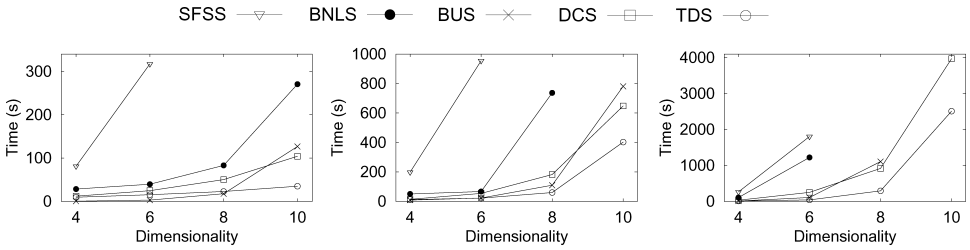



Fig. 12. Effect of dimensionality ($n = 500k$).

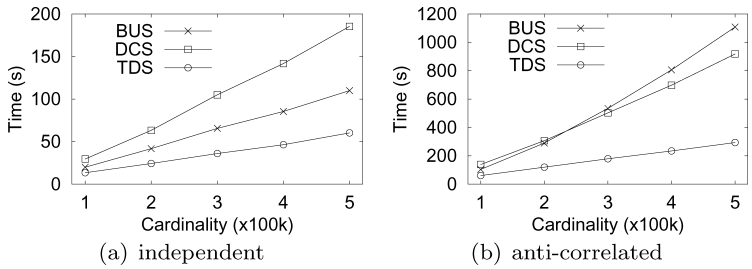


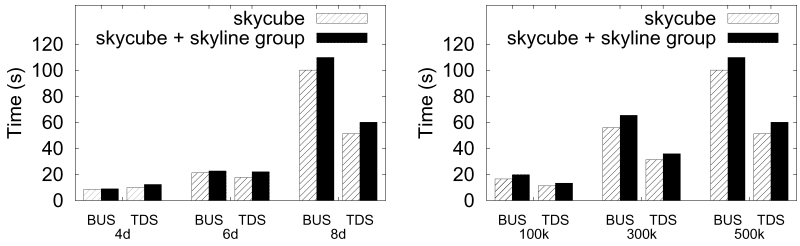

Fig. 13. Effect of cardinality ($d = 8$).

times are too large. For instance, in an anticorrelated dataset with $d = 10$, BNLS takes 265,000 seconds, while TDS requires only 2,500 seconds, which outperforms the former by 2 orders of magnitude. Due to their bad performance, we do not evaluate SFSS and BNLS in the following experiments.

In lower-dimensional datasets, BUS is faster than DCS and TDS. However, with the growth of the skyline size (e.g., in high-dimensional datasets or in anticorrelated datasets), more pairwise comparisons between objects and skyline objects are performed in BUS. Therefore, BUS performs badly in these datasets. As TDS employs the SDC algorithm, which can compute cuboids of a path with the same cost with which DC can compute the last cuboid in the path, TDS outperforms DCS in all datasets. When the dimensionality is low, the performance difference between DCS and TDS is relatively small. This is because the advantage of shared computation of cuboids in TDS over DCS is rather limited as both the number of cuboids and the cost of computing subspace cuboids are small with low dimensionality. The difference between their times increases quickly in high-dimensional datasets, since the number of cuboids increases exponentially as the dimensionality increases linearly. TDS is always at least 1.5 times faster than DCS in the dataset with dimensionality $d = 10$.

### 7.3 Effect of Cardinality

In order to evaluate the effect of cardinality on our techniques, we use datasets with the dimensionality $d = 8$ and vary the cardinality $n$ from $100k$ to $500k$. As the skyline in correlated datasets contains few skyline objects, we evaluate the performance of BUS, DCS, and TDS in independent and anticorrelated datasets only. Their times are shown in Figure 13.




(a) Varying Dim. ($n = 500k$) (b) Varying Card. ($d = 8$)

Fig. 14. Cost of skyline group computation.

Although both DCS and TDS take advantages of the sharing parent strategy, DCS is slower than BUS in independent datasets, while TDS outperforms BUS. It indicates that simply adopting the sharing parent strategy in DCS only cannot improve it significantly as DCS computes each skyline independently.

It is appares that TDS is the winner among all these algorithms. In the low-cardinality datasets ($d \leq 2$), TDS is 2 times faster than the others. With the growth of cardinality, TDS' time increases only slightly, while others increase rapidly. For example, in anticorrelated datasets, the ratio of TDS' time with the dataset $n = 500k$ to that with $n = 100k$ is 4.8, while BUS' and DCS' are 10.7 and 6.6, respectively. Compared with the other algorithms, TDS does not only have better performance but also better scalability.

### 7.4 Cost of Skyline Group Computation

As described in Section 5, both our BUS and TDS algorithms can construct SKYCUBE with or without computing skyline groups. In this section, we investigate the impact of the skyline group computation on the SKYCUBE computation. We conduct this experiment on two sets of datasets. The first set consists of independent datasets with dimensionality $d$ varying between 4 to 8 and fixed cardinality $n = 500k$. The other set contains $8d$ independent datasets with varying cardinality $n$ from $100k$ to $500k$. We measure the processing times of SKYCUBE computation with and without skyline the group computation. The results on the two datasets are shown in Figure 14(a) and Figure 14(b), respectively.

As we can see from the figures, computing the skyline group information in addition to the SKYCUBE computation incurs overheads. The overhead grows with the increase of cardinality or dimensionality, as more skyline groups exist and need to be computed. Nonetheless, the additional cost is relatively small compared to the cost of computing the SKYCUBE. In our experiments, the overhead due to skyline group computation remains under 20% in all the datasets for both algorithms.

### 7.5 Efficiency of Filter-Based Heuristic

Now we study the efficiency of the filter-based heuristic to our BUS algorithm. We implement our BUS in two ways. One (with filter) adopts a filter-based heuristic, the other (without filter) does not, which evaluates objects in the

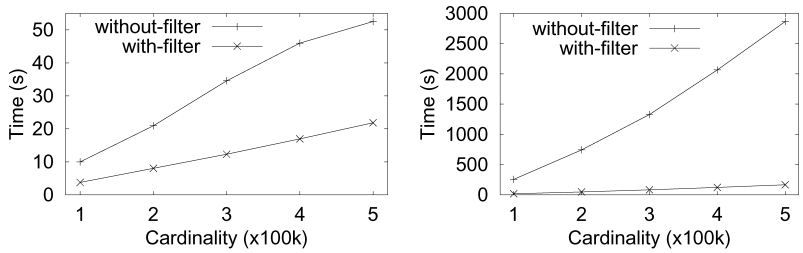



Fig. 15. Efficiency of filter-based heuristic ($d = 6$).

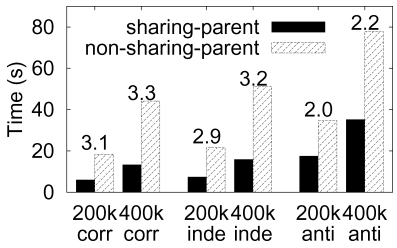



Fig. 16. Efficiency of sharing parent ($d = 6$).

same way as that of BNL. We evaluate them in the datasets with dimensionality $d = 6$ and vary cardinality $n$ from $100k$ to $500k$. Figure 15 shows the time in each dataset.

From the results, it is clear that with filter outperforms nonfilter in all datasets. As analyzed in Section 5.3.3, the filter-based heuristic can reduce the pairwise comparison between objects and the skyline objects, which is the most expensive computation in BUS. Thus, in filter-based heuristic improves BUS. In independent datasets the version with filter is 40% faster than without filter. With the growth of cardinality, the time of with filter remains fairly consistent, while that of nonfilter increases significantly. In the anticorrelated data setwith cardinality $n = 500k$, without filter is 27 times slower than with filter.

## 7.6 Efficiency of Sharing Parent Strategy

In this section, we evaluate the efficiency of the sharing parent strategy on the TDS algorithm. For two implementations (sharing-parent and nonsharing-parent) of TDS, we study their performance in correlated, independent, and anticorrelated datasets with dimensionality $d = 6$ and cardinality $n = 200k, 400k$. Figure 16 reports the query times on these datasets. The number above the bars inside of the figure is the ratio of nonsharing-parent time to that of sharing-parent.

It is clear that the sharing parent strategy improves TDS in all datasets. The sharing parent strategy enables TDS to compute the cuboids of one path from its parent cuboids while the nonsharing-parent does this from the whole dataset. Generally, the parent cuboid is much smaller than the whole dataset.

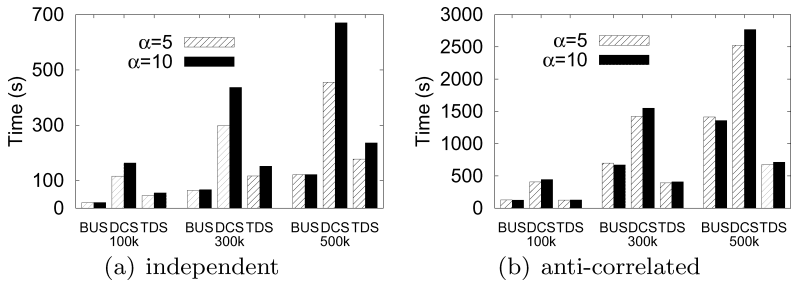



Fig. 17. Effect of duplicate values ($d = 8$).

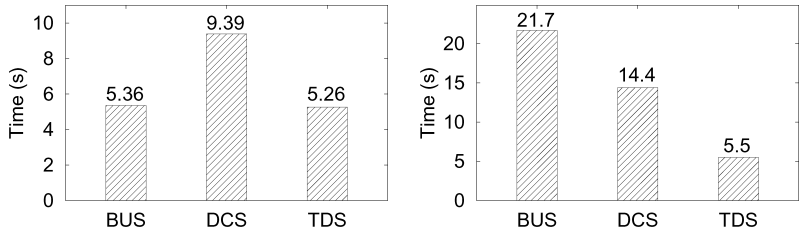
Furthermore, by the time ratios, it is shown that sharing-parent becomes more efficient in the larger dataset.

### 7.7 Effect of Duplicate Values

In this experiment, we study the effect of the number of duplicate values per dimension on our techniques. We define the duplicate ratio $\alpha$ in the following way: on each dimension for every object $p$, there are $\alpha - 1$ number of other objects with the same coordinate as $p$'s. In this experiment, independent and anticorrelated datasets with dimensionality $d = 8$, cardinality $n = 100k, 300k, 500k$, and duplicate ratio $\alpha = 5, 10$ are generated. Note that compared with the datasets used in the previous experiments, these datasets contain more duplicate values on each dimension. Figure 17 shows the times in these datasets.

In the independent dataset, all algorithms take more time in higher, duplicate ratio datasets because the size of each cuboid is larger. Among these algorithms, the effect of duplicate values on BUS is the least. Although the larger size of the cuboid in higher duplicate ratio datasets causes more pairwise comparison in BUS, it also makes the sharing result strategy, which is adopted in BUS, more efficient. As a result, BUS has similar performance in these two datasets with different duplicate ratios. On the contrary, in higher duplicate ratio dataset, the larger size of the cuboid decreases the efficiency of the sharing parent strategy because the difference between the size of the whole dataset and that of the parent cuboid is smaller. Therefore, DCS, which employs the sharing parent strategy only, is affected by duplicate values most.

In anticorrelated datasets, it is interesting that BUS is a little bit faster in the higher duplicate ratio dataset than that in the lower one. It confirms that the sharing result strategy is more efficient when each cuboid's size is larger. However, with the growth of the dataset, the number of pairwise comparisons between objects and skyline objects in BUS increases rapidly which slows BUS down. Similar to the trends in the previous experiments (Figure 12, 13), TDS performs best among these algorithms. In the datasets with cardinality $n = 500k$ and duplicate ratio $\alpha = 10$, TDS takes 710 seconds which is around 4 times faster than DCS. Moreover, it is clear that the difference between TDS' times on two duplicate ratio databases is smaller than that of DCS.




(a) Household $(d = 6, n = 128k)$ (b) Color $(d = 9, n = 68k)$

Fig. 18. Performance on the real datasets.

### 7.8 Performance on Real Datasets

Finally, we evaluate our skycube computation algorithms on real datasets. In this experiment, two real datasets Household (http://www.ipums.org) and Color (http://kdd.ics.uci.edu) are deployed. Specifically, Household is a 6-dimensional dataset with a cardinality 127, 931, and each object represents the percentage of an American family's annual income spent on 6 types of expenditures (e.g., gas, etc.). Color consists of 68, 040 9-dimensional objects, where the dimensions represent different properties of an image. Figure 18 plots the runtimes of BUS, DCS, and TDS on these two datasets.

In accordance with the trends of the experimental results on the synthetic datasets, TDS outperforms the other algorithms in both datasets. This is because TDS can efficiently compute several cuboids simultaneously with little overhead. In the lower-dimensional dataset (i.e., Household), BUS has similar performance as TDS. However, with increasing dimensionality, more pairwise comparisons among objects and skyline objects are performed in BUS. Consequently, the performance of BUS is deteriorated in the Color dataset.

### 7.9 Summary

Our extensive performance study clearly shows that the multidimensional skyline analysis is interesting and useful in real datasets. The three algorithms, BUS, DCS and TDS, have similar performance on small datasets of low dimensionality. TDS outperforms the others substantially when the dimensionality is high, the cardinality is large, and there are many subspace skyline objects (e.g., in the anticorrelated datasets).

## 8. CONCLUSIONS

In this article, we answered the questions about semantics of skyline objects by introducing the novel notions of skyline groups and decisive subspaces. We proposed the problem of subspace skyline analysis and computation. On the subspace skyline analysis side, a novel roll-up and drill-down analysis of skylines in various subspaces was introduced. On the subspace skyline computation side, two efficient algorithms Bottom-Up and Top-Down were developed. A performance study using both real and synthetic datasets was conducted to verify the meaningfulness and the efficiency of our approach. The experimental results strongly suggest that the semantics of skyline objects and subspace

skyline analysis are highly meaningful in practice, and the two algorithms are efficient and scalable.

This research also naturally leads to a few interesting problems for future work. For example, a SKYCUBE and a complete set of skyline groups can be costly in space. Under a constraint on space usage, how to materialize a subset of skyline cuboids to facilitate query answering is an interesting question. Moreover, it is important to develop efficient algorithms for SKYCUBE, and skyline group computation for high dimensional data is a challenging task. Some other issues, such as incremental maintenance of SKYCUBE and skyline groups as well as disk-based and/or parallel algorithms, also deserve further investigation.

## ACKNOWLEDGMENTS

The authors are grateful to the reviewers for their careful and insightful reviews, and to Dr. Donald Kossmann for providing us the synthetic data generator.

## REFERENCES

AGARWAL, S., AGRAWAL, R., DESHPANDE, P., GUPTA, A., NAUGHTON, J. F., RAMAKRISHNAN, R., AND SARAWAGI, S. 1996. On the computation of multidimensional aggregates. In *Proceedings of the 22ndh International Conference on Very Large Data Bases (VLDB'96)*. Mumbai (Bombay), India, 506–521.

AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. N. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'93)*. Washington, DC, 207–216.

ANGIULLI, F., IANNI, G., AND PALOPOLI, L. 2004. On the complexity of inducing categorical and quantitative association rules. *Theor. Comput. Sci. 314*, 1, 217–249.

BALKE, W.-T. AND GÜNTZER, U. 2004. Multi-objective query processing for database systems. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*. Toronto, Canada, 936–947.

BALKE, W.-T., GÜNTZER, U., AND ZHENG, J. X. 2004. Efficient distributed skylining for web information systems. In *The 9th International Conference on Extending Database Technology (EDBT'04)*. Heraklion, Crete, Greece, 256–273.

BENTLEY, J. L., KUNG, H. T., SCHKOLNICK, M., AND THOMPSON, C. D. 1978. On the average number of maxima in a set of vectors and applications. *J. ACM 25*, 4, 536–543.

BEYER, K. AND RAMAKRISHNAN, R. 1999. Bottom-up computation of sparse and iceberg cubes. In *Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*. Philadelphia, 359–370.

BÖRZSÖNYI, S., KOSSMANN, D., AND STOCKER, K. 2001. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*. Heidelberg, Germany, 421–430.

BUCHTA, C. 1989. On the average number of maxima in a set of vectors. *Information Processing Letters 33*, 2, 63–65.

CHAN, C. Y., ENG, P.-K., AND TAN, K.-L. 2005. Stratified computation of skylines with partially-ordered domains. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'05)*. 203–214.

CHOMICKI, J., GODFREY, P., GRYZ, J., AND LIANG, D. 2003. Skyline with presorting. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*. Bangalore, India, 717–816.

FENG, Y., AGRAWAL, D., ABBADI, A. E., AND METWALLY, A. 2004. Range cube: Efficient cube computation by exploiting data correlation. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*. Boston, MA, 658–670.

GANTER, B. AND WILLE, R. 1996. *Formal Concept Analysis—Mathematical Foundations*. Springer, Berline, Germany.

GODFREY, P., SHIPLEY, R., AND GRYZ, J. 2005. Maximal vector computation in large data sets. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*. Trondheim, Norway, 229–240.

GRAY, J., BOSWORTH, A., LAYMAN, A., AND PIRAHESH, H. 1996. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*. New Orleans, LA, 152–159.

HAN, J., PEI, J., DONG, G., AND WANG, K. 2001. Efficient computation of iceberg cubes with complex measures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'01)*. Santa Barbara, CA, 1–12.

KNORR, E. M. AND NG, R. T. 1999. Finding intensional knowledge of distance-based outliers. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*. Morgan Kaufmann Publishers, 211–222.

KOLTUN, V. AND PAPADIMITRIOU, C. H. 2005. Approximately dominating representatives. In *The 10th International Conference on Database Theory (ICDT'05)*. Edinburgh, Scotland, 204–214.

KOSSMANN, D., RAMSAK, F., AND ROST, S. 2002. Shooting starts in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*. Hong Kong, China, 301–310.

KUNG, H. T., LUCCIO, F., AND PREPARATA, F. P. 1975. On finding the maxima of a set of vectors. *J. ACM 22*, 4, 469–476.

LAKSHMANAN, L. V. S., PEI, J., AND HAN, J. 2002. Quotient cube: How to summarize the semantics of a data cube. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*. Hong Kong, China, 778–789.

LAKSHMANAN, L. V. S., PEI, J., AND ZHAO, Y. 2003. Qc-trees: An efficient summary structure for semantic OLAP. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*. San Diego, CA, 64–75.

LIN, X., YUAN, Y., WANG, W., AND LU, H. 2005. Stabbing the sky: Efficient skyline computation over sliding windows. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*. Tokyo, Japan, 502–513.

PAPADIAS, D., TAO, Y., FU, G., AND SEEGER, B. 2003. An optimal and progressive algorithm for skyline queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*. San Diego, CA, 467–478.

PASQUIER, N., BASTIDE, Y., TAOUIL, R., AND LAKHALS, L. 1999. Discovering frequent closed itemsets for association rules. In *The 7th International Conference on Database Theory (ICDT'99)*. Jerusalem, Israel, 398–416.

PEI, J., JIN, W., ESTER, M., AND TAO, Y. 2005. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*. Trondheim, Norway, 253–264.

ROSS, K. A. AND SRIVASTAVA, D. 1997. Fast computation of sparse datacubes. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*. Athens, Greece, 116–125.

SARAWAGI, S., AGRAWAL, R., AND GUPTA, A. 1996. On computing the data cube. Tech. rep., IBM Almaden Research Center.

TAN, K.-L., ENG, P.-K., AND OOI, B. C. 2001. Efficient progressive skyline computation. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*. Roma, Italy, 301–310.

XIA, T. AND ZHANG, D. 2006. Refreshing the sky: The compressed skycube with efficient support for frequent updates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*.

XIN, D., HAN, J., LI, X., AND WAH, B. W. 2003. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*. Berlin, Germany, 476–487.

YANG, G. 2004. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*. ACM Press, 344–353.

YUAN, Y., LIN, X., LIU, Q., WANG, W., YU, J. X., AND ZHANG, Q. 2005. Efficient computation of the skyline cube. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*. Trondheim, Norway, 241–252.

ZHAO, Y., DESHPANDE, P. M., AND NAUGHTON, J. F. 1997. An array-based algorithm for simultaneous multidimensional aggregates. In *Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD'97)*. Tucson, Az, 159–170.

Received January 2006; revised June 2006; accepted August 2006