Measuring the Sky: On Computing Data Cubes via Skylining the Measures

Man Lung Yiu, Eric Lo, and Duncan Yung

Abstract—Data cube is a key element in supporting fast OLAP. Traditionally, an aggregate function is used to compute the values in data cubes. In this paper, we extend the notion of data cubes with a new perspective. Instead of using an aggregate function, we propose to build data cubes using the skyline operation as the "aggregate function". Data cubes built in this way are called "group-by skyline cubes" and can support a variety of analytical tasks. Nevertheless, there are several challenges in implementing group-by skyline cubes in data warehouses: (i) the skyline operation is computational intensive, (ii) the skyline operation is holistic, and (iii) a group-by skyline cube contains both grouping and skyline dimensions, rendering it infeasible to pre-compute all cuboids in advance. This paper gives details on how to store, materialize, and query such cubes.

Index Terms—H.2.4.h Query processing; H.2.7.b Data warehouse and repository

1 Introduction

In the data warehousing environment, OLAP tools have been extensively used for a wide range of decision support applications such as sales analysis, customer analysis, marketing, and services planning. These OLAP tools are built upon a *multidimensional data model*, in which data tuples are partitioned into different *cells* based on the values of their *dimension* attributes. For each cell of tuples, an *aggregate function* (e.g., SUM()) is applied to their *measure* attributes (e.g., sales) and the resulting aggregated value contributes to the value of that cell. A data *cuboid* is then formed by constituting cells that are created from the same set of dimension attributes. Each data cuboid represents a unique view of the underlying data. The collection of data cuboids, which are based on different combinations of dimension attributes, forms a *data cube*.

In traditional data cubes, an aggregate function takes as input a set of measure values and returns a *single* numeric value. In this paper, we extend the notion of data cube with a new perspective. Specifically, we study the issues of building data cubes that exploit the skyline operator [4] as the post-operation instead of the traditional aggregate functions. We name this type of data cubes as *group-by skyline cubes*. The skyline operator has been well recognized as a very important decision-support operator in recent literature and has started to be implemented in commercial query engines [7]. Given a set S of skyline attributes, a tuple t is said to *dominate* another tuple t', denoted by $t \succ_S t'$, if

$$(\exists\ A_i \in S,\ t[A_i] < t'[A_i]) \wedge (\forall\ A_i \in S,\ t[A_i] \leq t'[A_i]) \quad (1)$$

assuming that smaller values are preferable over larger ones. Here, we use $t[A_i]$ to represent the value of the attribute A_i of the tuple t. Given a set $\mathcal{D}$ of tuples, the skyline operation Ψ on $\mathcal{D}$ is defined as:

$$\Psi(\mathcal{D}, S) = \{t \in \mathcal{D} \mid \nexists t' \in \mathcal{D}, t' \succ_S t\} \quad (2)$$

In other words, a tuple t belongs to the skyline result set if no other tuple dominates it.

We observe that a group-by skyline cube has two distinguished features when compared to a traditional data cube. First, a cell of a group-by skyline cube stores a set of skyline tuples rather than a single numeric aggregate value. Second, since the skyline operation can be applied to multiple measure attributes, a group-by skyline cube includes not only a set of usual *grouping dimensions*, but also a set of *skyline dimensions*, in which the latter is defined on the measure attributes.

Name	Region	Position	Mentality	Attitude	Flexibility
a	Asia	Admin	5	5	6
b	Asia	Admin	4	8	7
c	Asia	Technical	3	3	4
d	Asia	Technical	6	6	6
e	Europe	Technical	2	4	8
f	Europe	Technical	4	4	3
g	Europe	Technical	7	7	4

Fig. 1. Employee Database $\mathcal{D}$

We illustrate the concept of group-by skyline cube by the example in Figure 1, which is a table $\mathcal{D}$ of employee records. Each tuple represents an employee, with the employee's 'Region' and 'Position', as well as his/her evaluated scores 'Mentality', 'Attitude', and 'Flexibility'. Suppose that lower scores are preferable over higher ones. If we want to find the outstanding employees of each position, the table in Figure 2a can be used to answer the query directly. That table is called a *group-by skyline cuboid* with grouping dimension set $G = \{$'Position'$\}$ and skyline dimension set $S = \{$'Mentality', 'Attitude', 'Flexibility'$\}$. That essentially means that the employee in $\mathcal{D}$ are first grouped based on their positions and then the skyline operation is applied to each group to find those who are not dominated by the others on all three scores.

The research was supported by grant PolyU 525009E from Hong Kong RGC.

- *M. L. Yiu, E. Lo and D. Yung are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong.*
  E-mail: {csmlyiu, ericlo, cskwyung}@comp.polyu.edu.hk

$\mathcal{G}$={Position}, $\mathcal{S}$={Mentality, Attitude, Flexibility}

Position	Name	Mentality	Attitude	Flexibility
Admin	a	5	5	6
	b	4	8	7
Technical	c	3	3	4
	e	2	4	8
	f	4	4	3

(a) Group-by skyline cuboid $\mathcal{C}_1$

$\mathcal{G}$={Region, Position}, $\mathcal{S}$={Mentality, Attitude}

Region	Position	Name	Mentality	Attitude
Asia	Admin	a	5	5
		b	4	8
Asia	Technical	c	3	3
Europe	Technical	e	2	4

(b) Group-by skyline cuboid $\mathcal{C}_2$

$\mathcal{G}$={Region, Position}, $\mathcal{S}$={Mentality}

Region	Position	Name	Attitude
Asia	Admin	a	5
Asia	Technical	c	3
Europe	Technical	e	4
		f	4

(c) Group-by skyline cuboid $\mathcal{C}_3$

Fig. 2. Group-by skyline cuboids

Figures 2b and 2c show two more examples of group-by skyline cuboids for the employee dataset. Cuboid $\mathcal{C}_2$ groups the employees based on their regions and positions and the skyline operation is applied to their 'Mentality' and 'Attitude' attributes. Cuboid $\mathcal{C}_3$ essentially partitions the employees in the same way as $\mathcal{C}_2$ but the skyline operation is applied to only the attribute 'Attitude'. Hence, for the dataset $\mathcal{D}$ in Figure 1, its corresponding group-by skyline cube is a collection of all cuboids, whereas each such cuboid $\mathcal{C}_i(G_i, S_i)$ has its grouping dimension set G_i as a non-empty subset of {'Region', 'Position'} and its skyline dimension set S_i as a non-empty subset of {'Mentality', 'Attitude', 'Flexibility'}.

We find group-by skyline cube useful for other data analysis applications as well. One can extend a supermarket's transactional data warehouse with group-by skyline cube so that attributes like 'Location', 'Time' and 'Product' still serve as the grouping dimensions and the set of measure attributes like 'Sales' and 'Number of visitors' serve as the skyline dimensions. For hotel analysis, we can build a group-by skyline cube with respect to all dimensions such as 'Star' (e.g., 5-star, 4-star) and 'City' and all measure attributes such as 'Room price', 'Room quality', 'Transportation convenience' and 'Entertainment quality'.

Using the skyline operation as an "aggregate function" in data cubes and the notion of group-by skyline cubes is a completely new concept. First, while the skyline operator and its variants have been extensively studied (e.g., [23], [10]), none of them has ever discussed the incorporation of the skyline operation into traditional data warehouses as a post-operation. Second, although the concept of *skycubes* [36], [26] exists, that is fundamentally different from our concept of group-by skyline cube. Skycubes are designed for the efficient evaluation of skyline queries in any non-empty subspaces. In other words, that does not consider any grouping at all. In practice, however, skyline analysis is often to be more meaningful if the data tuples are first grouped based on their dimensions before finding their skylines. Let's take the employee dataset in Figure 1 as an example. Without grouping, finding the skyline of *all* employee, no matter whether it is in the full space (i.e., all three measure attributes) or in any sub-space (e.g., 'Mentality' and 'Attitude'), may not be that meaningful because it is unfair to compare an Admin staff with a Technical staff. In contrast, building a group-by skyline cube for that dataset allows the management to find out the outstanding employee of each region and/or position. Consider the NBA player statistics as another example. Obviously, finding the skyline, despite that is full-space skyline or sub-space skyline, of *all* NBA players is not that meaningful because it is unfair to compare Michael Jordon with Wilt Chamberlain, which were active in NBA in different periods and played different positions. In contrast, we can analyze the NBA data using a group-by skyline cube, treating all dimensions such as 'Year', 'Team' and 'Position' as the set of all possible grouping dimensions and treating all measure attributes such as 'Points', 'Assists' and 'Rebounds' as the set of all possible skyline dimensions.

Implementing the concept of group-by skyline cube in today's data warehouses poses several technical challenges. First, the skyline operation is much more computational intensive [4], [7] when compared to simple aggregations such as COUNT(), SUM() and AVG(). Consequently, building or querying group-by skyline cubes needs to consider not only the I/O cost, but also the computation (CPU) cost. Second, the skyline operation is *holistic* in nature as the skyline of a cuboid is not necessarily derivable from another cuboid. For example, consider cuboids $\mathcal{C}_2$ and $\mathcal{C}_3$ in Figure 2, although they share the same set of grouping dimensions and the skyline dimension set of $\mathcal{C}_3$ is a subset of that of $\mathcal{C}_2$, their skyline results are not derivable from each other. Specifically, we can see that $\mathcal{C}_2$ is not derivable from $\mathcal{C}_3$ because the skyline employee b in $\mathcal{C}_2$ does not exist in $\mathcal{C}_3$. Moreover, $\mathcal{C}_3$ is not derivable from $\mathcal{C}_2$ because the skyline employee f in $\mathcal{C}_3$ does not exist in $\mathcal{C}_2$ as well. Third, the dimensions that define a group-by skyline cube include not only the usual grouping dimensions but also the measure attributes. For a group-by skyline cube that is built on a dataset with $|G|$ grouping dimensions and $|S|$ measure attributes, there would be $(2^{|G|}-1)(2^{|S|}-1)$ possible cuboids. That explosive number of possible cuboids makes materialization of group-by skyline cubes for efficient query evaluation becomes especially challenging.

To the best of our knowledge, the building of group-by skyline cube, or the implementation of the skyline operation as a post-operation in data warehouses, has not been addressed previously in the research literature or in commercial products. This paper studies this issue in detail. Our contributions can be summarized as follows:

1) The concept of group-by skyline cube is presented. That includes the discussion of what a 'group-by skyline cuboid' is, the relationships between different group-by skyline cuboids, and how these cuboids constitute a group-by skyline cube.
2) The technical details of supporting group-by skyline cube are presented. Specifically, we propose to materialize a group-by skyline cube as an *extended group-by skyline cube* (ES-cube). In an ES-cube, skyline results across cuboids are derivable from each other. We further

develop construction and query processing algorithms for ES-cube. We also develop a budget-based partial materialization algorithm that is able to select and materialize a good subset of cuboids in ES-cube that yields the highest query improvement in terms of both CPU and I/O costs.

3) An extensive set of experiments has been carried out on both real and synthetic data. Experimental results show that queries can be answered efficiently

The rest of the paper is organized as follows. Section 2 gives the background and the related work of this paper. Section 3 elaborates the concept of group-by skyline cube in more detail. Section 4 presents the issues related to implementation of group-by skyline cube. Section 5 reports experimental results. Section 6 concludes the paper with future research directions. The symbols to be used in subsequent discussion are summarized in Table 1.

TABLE 1
Summary of Symbols

Symbol	Meaning
G	a set of grouping attributes
S	a set of skyline attributes
$\mathcal{D}$	a set of tuples (with attributes on G and S)
$\mathcal{D}(g)$	a set of tuples in $\mathcal{D}$ belonging to the group g
$t \succ_S t'$	a tuple t dominates a tuple t' with respect to S
$t \succ_S^+ t'$	a tuple t strictly dominates a tuple t' with respect to S
$\Psi(\mathcal{D}(g), S)$	a skyline set on $\mathcal{D}(g)$ with respect to S
$\Psi^+(\mathcal{D}(g), S)$	an extended skyline set on $\mathcal{D}(g)$ with respect to S
$\widehat{\Psi}(\mathcal{D}(g), S)$	$\Psi^+(\mathcal{D}(g), S) - \Psi(\mathcal{D}(g), S)$
$Q(G_Q, S_Q)$	a group-by skyline query
$\mathcal{C}^+(G_C, S_C)$	an ES-cuboid
$\mathcal{P}_S(t)$	a parent skyline set of t with respect to S
$t.\gamma$	the bounding region of a OSP tree node
$t.\gamma^-$	the lower corner of $t.\gamma$
$t.\gamma^+$	the upper corner of $t.\gamma$
R_t	the region of points that causes visiting a tree node t
$H_Q(R)$	the prob. density of a region R in the histogram H_Q
β	# distinct values per dimension of G
θ	# distinct values per dimension of S (if discrete)
N	the number of data points
B_{item}	the number of items/attributes that fit in a disk page
ω	the number of distinct groups
$\mathcal{PA}_Q(\mathcal{C}^+)$	# disk page access of query Q on cuboid $\mathcal{C}^+$
$\mathcal{CP}_Q(\mathcal{C}^+)$	# dominance comparisons of query Q on cuboid $\mathcal{C}^+$

2 Background and Related Work

2.1 Data cubes

A data cube [13] can be viewed as a collection of *cuboids*, in which each cuboid stores the group-by aggregate result with respect to a particular set of attributes called dimensions. For a data cube with non-holistic aggregate functions, cuboids can be organized as a lattice. Figure 3 shows a lattice of 2^3 cuboids of a dataset with three dimension attributes A_1, A_2, and A_3. There exists a path from a cuboid $\mathcal{C}_i$ to a cuboid $\mathcal{C}_j$ if the attributes of $\mathcal{C}_i$ contain those of $\mathcal{C}_j$, meaning that $\mathcal{C}_i$ can be used to derive $\mathcal{C}_j$. For example, any cuboid in Figure 3 can be derived from the top cuboid $\mathcal{C}_1$.

To meet the performance demands imposed by OLAP operations, the cube materialization approach [14] which precomputes some cuboids in advance, is extensively applied to speed up the evaluation of various OLAP queries. Full

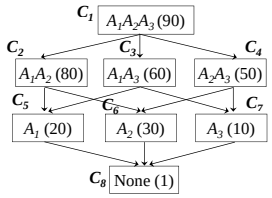


Fig. 3. A lattice of cuboids with attributes A_1, A_2 and A_3. The number of tuples in each cuboid is in the bracket.

materialization refers to the precomputation of all cuboids (i.e., the full cube) and is impractical because the number of cuboids is exponential to the number of dimensions. On the other hand, partial materialization refers to the precomputation of a subset of cuboids (i.e., the subcube) and is generally more frequently used. The cost of processing a query using a cuboid $\mathcal{C}_i$ is described by a *linear cost model* [15], i.e., the query time is assumed to solely depend on the I/O time of scanning $\mathcal{C}_i$, which is linear to the size of $\mathcal{C}_i$. During query evaluation, if a cuboid $\mathcal{C}_i$ requested by an OLAP query has already been materialized, it can serve as the query result right away. Even if $\mathcal{C}_i$ has not been materialized, it can still be efficiently derived from the smallest materialized cuboid $\mathcal{C}_j$ in which the dimensions of $\mathcal{C}_i$ are a subset of those of $\mathcal{C}_j$.

Given a space budget, the materialization algorithm in [15] selects cuboids to be materialized based on the greedy heuristics, aiming at improving the overall query cost. Initially, it materializes the top cuboid, i.e., the cuboid with all dimensions, and puts it into a materialized set $\mathcal{MT}$ because other cuboids can always be derived from the top cuboid directly. Afterwards, it iteratively materializes a cuboid $\mathcal{C}_i \notin \mathcal{MT}$ that is expected to bring the maximum *benefit* to the overall query cost and adds it to $\mathcal{MT}$. Based on the linear cost model, the benefit of materializing a cuboid $\mathcal{C}_i$ is defined as the improvement in I/O, with respect to the set of already materialized cuboids. For example, Figure 3 shows the number of tuples of each cuboid in a bracket. The top cuboid $\mathcal{C}_1$, with 90 tuples, is the first to be materialized. Next, the benefit of materializing cuboid $\mathcal{C}_4$ is calculated as $(90-50)\times 4$ because answering queries related to $\mathcal{C}_4$, $\mathcal{C}_6$, $\mathcal{C}_7$, or $\mathcal{C}_8$ can now exploit $\mathcal{C}_4$ (which requires scanning only 50 tuples) instead of the top cuboid $\mathcal{C}_1$ (which requires scanning 90 tuples). Since the benefit of $\mathcal{C}_4$ is the highest among all the cuboids that have not been materialized, it is next selected to be materialized and added to $\mathcal{MT}$. The iteration continues until the selected cuboids exceed the space budget.

Owning to its importance to data-warehousing technology, research related to data cube has continuously attracted the attentions of many researchers. Some focus on the efficient computation of cubes (e.g., [3], [27]), some focus on the compression and summarization of cubes (e.g., [29]), and some focus on the variations of cubes (e.g., [35]).

2.2 Skyline Problems

Skyline computation algorithms.
The skyline operator, introduced in [4], retrieves each tuple

t from the dataset $\mathcal{D}$ such that it is not dominated by any other tuple t' of $\mathcal{D}$. Existing skyline computation methods can be divided into two categories: (i) index-based solutions [30], [17], [23], and (ii) non-indexed solutions [4], [8], [12], [2], [37].

Early index-based solutions [30] operate on bitmap and B^+-tree indices on the skyline attributes, whereas the others [17], [23] operate on an R-tree that indexes the dataset by skyline attributes. The *Branch-and Bound Skyline* (BBS) algorithm [23] is the state-of-the-art indexed approach for skyline computation, and its I/O cost is proven to be optimal with respect to the R-tree instance.

The state-of-the-art skyline computation algorithm for non-indexed data is OSP [37]. It will be adopted as the skyline computation method in our proposed solution (see Section 4.3). Thus, we describe this method in detail below. Like the SFS algorithm [8], the OSP algorithm first sorts the dataset by a monotone score function $\mathcal{F}$ (on skyline attributes). $\mathcal{F}$ is simply the `SUM` function in [37]. The main difference is that OSP employs a (main-memory) OSP tree for indexing the skyline points found-so-far, rendering it efficient to process the remaining data points. Figure 4 illustrates how OSP works on a set of points (in the space defined by two skyline attributes S_1 and S_2). Each contour (in dotted line) depicts all possible locations with the same `SUM` score. The sorted points follow the order: $t_2, t_4, t_7, t_5, t_1, t_6, t_3$.

When a point t is retrieved (by the sorted order), we check whether the OSP tree contains any point t' that dominates t. This is implemented by a depth-first traversal of the OSP tree branches that potentially contain some point dominating t. If there is no such point t', then we report t as a skyline point and insert it into the OSP tree. Each node of the OSP tree stores exactly one skyline point, and its maximum number of children is $2^{|S|} - 2$, where S is the set of skyline attributes. In the example, when the first point t_2 is retrieved, it is inserted as the tree root, which decomposes the domain space into $2^2 = 4$ regions. Since t_2 is a skyline point, its bottom-left region is guaranteed to contain no points. Also, its top-right region is dominated by t_2 so the region cannot contain any skyline points. Those two regions do not need to be maintained by the OSP tree. The next retrieved point t_4 is a skyline as it is not dominated by any point in the OSP tree. Since t_4 falls into the top-left region of t_2, it is inserted into the left subtree of t_2. Then, the next point t_7 is again a skyline point; it falls into the bottom-right region of t_2 so it becomes the right child of t_2. The next point t_5 is also a skyline point, so it is inserted into the left subtree of t_7. The algorithm proceeds in this fashion until all the remaining points have been examined.

Variants of skyline queries.
Since the introduction of the skyline query in [4], numerous skyline variants have been proposed, for example, skyband [23], spatial skyline [28], reverse skyline [10], subspace skyline [32], [31], [16], dominance-based analysis [20], relaxation of the skyline conditions [6], [5], k most representative skyline points [21], as well as skyline processing using parallel execution [9] and skylining from distributed data sources [1].

In the OLTP environment, [23], [22] have studied the evaluation of skyline queries with group-by attributes. Papadias et al. [23] proposed an efficient R-tree based algorithm for processing skyline queries with group-by attributes. Unfortunately, their solution relies on the availability of an R-tree (with both group-by attributes and skyline attributes) so it is not suitable for handling ad-hoc queries. Luk et al. [22] focused on the relational engine and investigated how to find an efficient query plan for skyline queries with group-by attributes. Their solution, however, does not study the materialization of skyline cuboids in the OLAP environment.

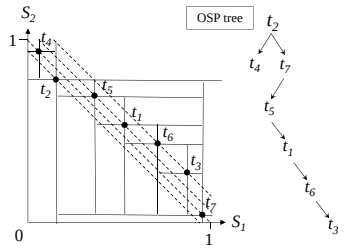


Fig. 4. Example of an OSP tree

Subspace skyline analysis.
Building data cubes using the skyline operation as the "aggregate function", to the best of our knowledge, have not been addressed before. Work in [24], [36], [25], [26], [31], [34] is developed for subspace skyline analysis. In the example of Figure 2, the cuboids in our group-by skyline cube contain both group-by attributes and skyline attributes. In contrast, the *skycube* [24], [36], [25] of a dataset $\mathcal{D}$ is defined as the collection of skyline result set $\Psi(\mathcal{D}, S)$ for each non-empty subset S of S^* but without any grouping attributes. A compressed skycube (CSC) [34], [26] is a compressed structure for storing a skycube; it still does not involve any grouping attributes. Those works do not consider any grouping operation and partial materialization, and thus they are different from our work here. Another related work is P-cube [35]. P-cube is designed to facilitate the processing of *selection skyline* queries, i.e., finding the skyline from tuples that satisfy the `WHERE` clause in SQL. P-cube is orthogonal to us because it does not consider partial materialization and it is designed for selection queries that access a small portion of data. In contrast, group-by skyline cube is designed for OLAP-style queries that access a significant portion of data for report generation.

Skyline cardinality and cost estimation.
In this paper, we adopt a budget-based partial materialization approach for selecting group-by skyline cuboids to be materialized such that they yield the highest improvement in query cost. Recall that each cell of a cuboid corresponds to a set of skyline tuples (with respect to particular group-by attributes). Thus, it is important to study existing work on estimating skyline cardinality and its computation cost [11], [7], [12], [38].

Godfrey et al. [11], [12] propose a mathematical model of the skyline cardinality for a uniformly-distributed dataset, and the asymptotic cost of several skyline algorithms in the average-case and the worst-case. Chaudhuri et al. [7] apply

the log sampling technique for estimating the skyline cardinality and the skyline computation cost of the BNL and SFS algorithms, with respect to arbitrary data distribution. Zhang et al. [38] propose a kernel-based model for a more accurate estimation of the skyline cardinality of arbitrary datasets; however, it does not study the computation cost of any skyline algorithm.

3 Concept of Group-by Skyline Cube

In this section, we first introduce the concept of group-by skyline cube whose measure is defined by a skyline operation, and then show that some nice properties of traditional datacubes are not longer valid in this context. We will discuss how those invalidated properties can be re-enabled again by using a skyline variant definition in Section 4.

Definition of group-by skyline cube.
Group-by skyline cube is intended for the efficient evaluation of various *group-by skyline queries*. Let $\mathcal{D}$ be a relational table instance, with the schema $A = (A_1, A_2, \ldots, A_k)$. For simplicity, suppose that every attribute A_i is numeric. Given a set $G \subset A$ of grouping attributes and a group instance g of G, we define the set $\mathcal{D}(g)$ as the set of tuples of $\mathcal{D}$ belonging to the group instance g. Accordingly, a group-by skyline query is defined as follows:

Definition 1: **Group-by skyline query.**
Given the sets G and S of attributes, a *group-by skyline* query $Q = (G, S)$ computes a skyline result set $\Psi(\mathcal{D}(g), S)$ for each group instance g defined on G.[1]

Considering the dataset $\mathcal{D}$ in Figure 1, the result of a group-by skyline query Q with its grouping dimension set G = {'Region', 'Position'} and its skyline dimension set S = {'Mentality', 'Attitude'} is shown in Figure 2b. An example of a group instance g is ('Asia', 'Admin').

In order to evaluate various group-by skyline queries efficiently, we introduce the concepts of group-by skyline cuboid and group-by skyline cube.

Definition 2: **Group-by skyline cube**
Let G^* be the set of all possible grouping dimensions and S^* be the set of all possible skyline dimensions. Given a subset $G \subseteq G^*$ and a subset $S \subseteq S^*$, a *group-by skyline cuboid* $\mathcal{C}(G, S)$ is defined as a collection of *cells*, each cell g (corresponding to a group of G) stores the set of skyline result set $\Psi(\mathcal{D}(g), S)$. The *group-by skyline cube* is defined as the collection of $\mathcal{C}(G, S)$, for any subset $G \subseteq G^*$ and non-empty $S \subseteq S^*$.

Consider the relational table $\mathcal{D}$ in the left part of Figure 5. In this example, $G^* = \{G_1, G_2, G_3\}$ and $S^* = \{S_1, S_2\}$. The group-by skyline cube for $\mathcal{D}$ thus contains $(2^{|G^*|} - 1)(2^{|S^*|} - 1)$, i.e., $7 \times 3 = 21$ cuboids, in total. The right part of Figure 5 depicts the contents of six cuboids. For instance, cuboid A contains 3 cells, indicating the distinct combinations (c_1, d_1), (c_1, d_2), (c_2, d_2) on the grouping attributes (G_2, G_3), for tuples in the table $\mathcal{D}$. For the cell (c_1, d_1) of cuboid A, the skyline of this group are tuples t_1 and t_2.

Conditions for sharing computations of cubes.
Most data cube algorithms, regardless of whether they are used to compute data cubes or to answer queries, also rely heavily on the sharing of computations to achieve higher efficiency. For example, in a sales application, if a cuboid (*year*, *department*: COUNT(*sales*)) that stores the sales of each department in every year has been computed, it can be used to compute the cuboid (*year* : COUNT(*sales*)) that stores the sales of all departments in every year, without accessing the base data. However, it is unclear whether such sharing is still valid in the context of group-by skyline cube.

In the following, we show that group-by skyline cubes can share some but not all computational effort. Specifically, for group-by skyline cuboids that share the same set of skyline dimensions S, the computational effort can be shared:

Lemma 1: **Union of skyline.**
Given any subset $\mathcal{D}' \subseteq \mathcal{D}$, $\Psi(\mathcal{D}, S) \subseteq \Psi(\mathcal{D}', S) \cup \Psi(\mathcal{D} - \mathcal{D}', S)$.

Through Lemma 1 [19], we arrive at the following lemma that shows that one group-by skyline cuboid can be derived from another if they share the same set of skyline dimensions:

Lemma 2: **Group-by skyline cuboid hierarchy.**
Given two group-by skyline cuboids $\mathcal{C}(G, S)$ and $\mathcal{C}(G', S)$ such that $G' \subseteq G$, the group-by skyline cuboid $\mathcal{C}(G', S)$ can be derived from $\mathcal{C}(G, S)$.

Proof: By the definition of group-by skyline cuboid, each cell g' in $\mathcal{C}(G', S)$ corresponds to a subset V of cells in $\mathcal{C}(G)$, i.e., $\mathcal{D}(g') = \bigcup_{g \in V} \mathcal{D}(g)$ and different $\mathcal{D}(g)$ are disjoint. By applying Lemma 1, the result of g' can be derived from the results of the set V of cells in $\mathcal{C}(G, S)$. Thus, this lemma is proved. □

Figure 6 shows the lattice of group-by skyline cubiods (i.e., group-by skyline cube) for the dataset in Figure 5. We organize the group-by skyline cuboids in a way such that each edge represents a parent-child relationship between two group-by skyline cuboids, meaning that the result of a child cuboid can be derived from its parent cuboid through Lemma 2. For instance, there is an edge from cuboid A to cuboid D, showing that cuboid D can be derived from cuboid A. To illustrate, cell c_1 of cuboid D in Figure 5 can be obtained by finding the skylines among the cells (c_1, d_1) and (c_1, d_2) of cuboid A.

Unfortunately, we remark that computational efforts between cuboids that share different sets of skyline dimensions cannot be shared unless the *distinct value condition* (no two tuples share the same value on any subset of skyline attributes) [36] holds. For instance, in Figure 6, although cuboid C shares the same set of grouping dimensions as cuboid A and the skyline dimension of cuboid C is a subset of cuboid A, C cannot be derived from A, unless the tuples have distinct values on the skyline attributes. To illustrate, in Figure 5, the cell (c_2, d_2) of cuboid C includes a tuple t_6 that does not exist in cuboid A. As we have mentioned, the efficiency of the state-of-the-art data cube construction/query algorithms rely heavily on the fact that many computational efforts can be shared. However, in a

1. Note that when S contains only one attribute, the skyline operation becomes a simple aggregate function MAX or MIN. Nonetheless, our definition includes such case for generality.

Example $\mathcal{D}$

TID	Possible Grouping Attributes			Possible Skyline Attributes	
	G_1	G_2	G_3	S_1	S_2
t_1	b_1	c_1	d_1	0.5	0.5
t_2	b_1	c_1	d_1	0.1	0.8
t_3	b_1	c_1	d_2	0.3	0.3
t_4	b_2	c_1	d_2	0.6	0.6
t_5	b_2	c_2	d_2	0.2	0.4
t_6	b_2	c_2	d_2	0.4	0.4
t_7	b_2	c_2	d_2	0.7	0.7

A: Cuboid $\mathcal{C}(G_2G_3, S_1S_2)$

G_2	G_3	TID	S_1	S_2
c_1	d_1	t_1	0.5	0.5
		t_2	0.1	0.8
c_1	d_2	t_3	0.3	0.3
c_2	d_2	t_5	0.2	0.4

B: Cuboid $\mathcal{C}(G_2G_3, S_1)$

G_2	G_3	TID	S_1
c_1	d_1	t_2	0.1
c_1	d_2	t_3	0.3
c_2	d_2	t_5	0.2

C: Cuboid $\mathcal{C}(G_2G_3, S_2)$

G_2	G_3	TID	S_2
c_1	d_1	t_1	0.5
c_1	d_2	t_3	0.3
c_2	d_2	t_5	0.4
		t_6	0.4

D: Cuboid $\mathcal{C}(G_2, S_1S_2)$

G_2	TID	S_1	S_2
c_1	t_2	0.1	0.8
	t_3	0.3	0.3
c_2	t_5	0.2	0.4

E: Cuboid $\mathcal{C}(G_2, S_1)$

G_2	TID	S_1
c_1	t_2	0.1
c_2	t_5	0.2

F: Cuboid $\mathcal{C}(G_2, S_2)$

G_2	TID	S_2
c_1	t_3	0.3
c_2	t_5	0.4
	t_6	0.4

Fig. 5. An example dataset (on the left) and some of its group-by skyline cuboids (on the right)

group-by skyline cube, the sharing of computation is limited by the distinct value condition, which almost never holds in real datasets. Therefore the group-by skyline cuboids in Figure 6 are separated into 3 *disjoint* clusters in which computational efforts are not sharable across the clusters. Fortunately, there are ways to remove such barriers and we will discuss them in the next section.

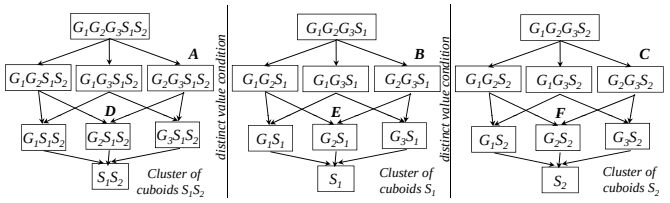


Fig. 6. Group-by skyline cube

4 IMPLEMENTATION

In this section, we address the issues related to the implementation of group-by skyline cube. Specifically, we attempt to answer the following research questions:

- ($\mathbb{RQ}1$) *Given the poor connections between group-by skyline cuboids, how can we enable the sharing of computational efforts between cuboids?* (Section 4.1)
- ($\mathbb{RQ}2$) *Given a materialized cuboid, how can a query Q be answered efficiently?* (Section 4.2)
- ($\mathbb{RQ}3$) *Given a limited storage space budget, how shall we select a subset of cuboids that offers the best improvement in query performance?* (Section 4.3)
- ($\mathbb{RQ}4$) *After selecting the cuboids, how can we construct them in an efficient manner?* (Section 4.4)

4.1 Materialization: extended group-by skyline cube

In order to maximize the sharing of computation of group-by skyline cuboids, especially the sharing of computation across the set of cuboids separated by the distinct value condition, we materialize a group-by skyline cuboid using an extended definition of skyline. We show that group-by skyline cubes materialized in this way can enable sharing across various cuboids.

Definition 3: **Extended group-by skyline.**
The *extended group-by skyline* of the tuple set $\mathcal{D}(g)$ with respect to the skyline attribute set S is defined as:

$$\Psi^+(\mathcal{D}(g), S) = \{t \in \mathcal{D}(g) \mid \nexists t' \in \mathcal{D}(g), t' \succ_S^+ t\}$$

where a tuple t is said to *strictly dominate* another tuple t' with respect to the attribute set S, denoted by $t \succ_S^+ t'$, if:

$$(\forall\ A_i \in S,\ t[A_i] < t'[A_i])$$

The notion of extended skyline [33] was originally proposed for skyline evaluation in P2P network. We borrow that concept and propose to materialize a group-by skyline cube as an *extended group-by skyline cube*.

Definition 4: **Extended group-by skyline cube (ES-cube).** Given a set G of grouping attributes and a set S of skyline attributes, we define an *extended group-by skyline cuboid* $\mathcal{C}^+(G, S)$ as a collection of cells, where each cell g (corresponding to a group of G) stores the set of extended skyline $\Psi^+(\mathcal{D}(g), S)$. The *extended group-by skyline cube* is defined as the collection of $\mathcal{C}^+(G, S)$, for any subset $G \subseteq G^*$ and $S \subseteq S^*$.

Lemma 3: **Extended group-by skyline cuboid hierarchy.** Given two extended group-by skyline cuboids $\mathcal{C}^+(G, S)$ and $\mathcal{C}^+(G', S')$ such that $G' \subseteq G$ and $S' \subseteq S$, the cuboid $\mathcal{C}^+(G', S')$ can be derived from $\mathcal{C}^+(G, S)$.

Proof: Each cell g' in the cuboid $\mathcal{C}^+(G', S')$ corresponds to a subset V of cells in the cuboid $\mathcal{C}^+(G, S)$, i.e., $\mathcal{D}(g') = \bigcup_{g \in V} \mathcal{D}(g)$ and different $\mathcal{D}(g)$ are disjoint. Thus, the result of g' can be derived from the results of the set V of cells in $\mathcal{C}^+(G, S)$. □

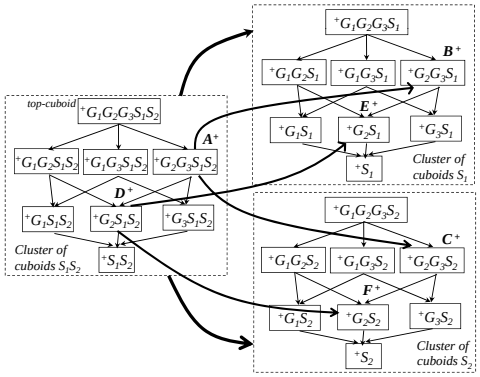


Fig. 7. Extended group-by skyline cube

Lemma 3 is powerful. It allows us to maximize the possibility of sharing computations among the ES-cuboids. Figure 7 shows the lattice of ES-cuboids (i.e., ES-cube) for the dataset in Figure 5. Note that, through Lemma 3, the 3 disjoint clusters of cuboids in Figure 6 are now connected.

For instance, cuboids B and C cannot be derived from any cuboid in the cluster of cuboids S_1S_2 (the left cluster in Figure 6) but now they can be derived from extended cuboid A^+. To illustrate, Figure 8 depicts the extended group-by skyline version of the six cuboids we illustrated in Figure 5. Note that, now cuboid A^+ contains tuple t_6 because t_5 cannot strictly dominate t_6 on every skyline attribute. Consequently, cell (c_2, d_2) of group-by skyline cuboid C can now be obtained by computing the skyline of cell (c_2, d_2) of cuboid A^+ on dimension S_2. Similarly, group-by skyline cuboids E and F cannot be derived from any group-by skyline cuboid in the cluster of cuboids S_1S_2 but now they can be derived from extended group-by skyline cuboid D^+. For brevity, we do not show all parent-child relationships in Figure 7, but every ES-cuboid on the right hand side of Figure 7 (and every group-by skyline cuboid) indeed can be derived from a cuboid on the left hand side. Furthermore, every ES cuboid (and every group-by skyline cuboid) can be derived from the top cuboid $\mathcal{C}^+(G_1G_2G_3, S_1S_2)$.

Fig. 8. Example of extended group-by skyline cuboids

A^+: Cuboid $\mathcal{C}^+(G_2G_3, S_1S_2)$

G_2	G_3	TID	S_1	S_2
c_1	d_1	t_1	0.5	0.5
		t_2	0.1	0.8
c_1	d_2	t_3	0.3	0.3
c_2	d_2	t_5	0.2	0.4
		t_6	0.4	0.4

D^+: Cuboid $\mathcal{C}^+(G_2, S_1S_2)$

G_2	TID	S_1	S_2
c_1	t_2	0.1	0.8
	t_3	0.3	0.3
c_2	t_5	0.2	0.4
	t_6	0.4	0.4

B^+: Cuboid $\mathcal{C}^+(G_2G_3, S_1)$

G_2	G_3	TID	S_1
c_1	d_1	t_2	0.1
c_1	d_2	t_3	0.3
c_2	d_2	t_5	0.2

E^+: Cuboid $\mathcal{C}^+(G_2, S_1)$

G_2	TID	S_1
c_1	t_2	0.1
c_2	t_5	0.2

C^+: Cuboid $\mathcal{C}^+(G_2G_3, S_2)$

G_2	G_3	TID	S_2
c_1	d_1	t_1	0.5
c_1	d_2	t_3	0.3
c_2	d_2	t_5	0.4
		t_6	0.4

F^+: Cuboid $\mathcal{C}^+(G_2, S_2)$

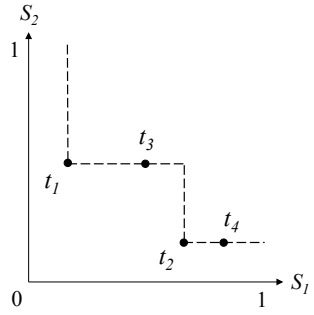
G_2	TID	S_2
c_1	t_3	0.3
c_2	t_5	0.4
	t_6	0.4

4.2 Storage and Query Processing

Now, we discuss how to answer a group-by skyline query $Q = (G_Q, S_Q)$ using a materialized ES-cuboid $\mathcal{C}^+(G, S)$, where $G_Q \subseteq G$ and $S_Q \subseteq S$. We first concentrate on the case that $G_Q = G$ and $S_Q \subseteq S$. A simple method in this case is to project the points in each cell $g \in \mathcal{C}^+$ to the query skyline subspace S_Q, and then find the skyline. Figure 9 shows a set of extended skyline points in a cell. If $S_Q = \{S_1\}$, this method projects all four points to the S_1 and then finds the skyline among those points. Note that if the skyline dimensions are low-cardinalities, the performance gain brought by ES-cubes may significantly decrease because the number of extended skylines increases. However, we are going to show that, by carefully storing the extended skyline points, ES-cubes can still provide certain improvement when the skyline dimensions are low-cardinalities. First, from the example above, we can see that t_3 is at most as good as t_1 in sub-space S_2, but t_3 gets dominated by t_1 in the full space and sub-space S_1. In other words, if t_1 is not a skyline in the query subspace, then it is not necessary to examine t_3 at all. Therefore, we decompose a set of extended skyline points Ψ^+ into two disjoint subsets: (i) the skyline set Ψ and (ii) the *child set* $\widehat{\Psi}$, where $\widehat{\Psi} = \Psi^+ - \Psi$. Given a point t in the child set $\widehat{\Psi}$, we define the *parent skyline set* of t as follows:

$$\mathcal{P}_S(t) = \{t' \in \Psi \mid (t' \succ_S t) \wedge (t' \not\succ_S^+ t)\}$$

Figure 9b illustrates the decomposition of Ψ^+ in terms of Ψ and $\widehat{\Psi}$, with respect to the example in Figure 9a. The parent skyline set of each tuple in $\widehat{\Psi}$ is also shown in the figure. For instance, the parent set of t_3 contains t_1 because t_1 dominates t_3 but t_1 does not strictly dominates t_3. Note that we only store the parent skyline set as a set of IDs. To avoid random page accesses (during query processing), each tuple t of $\widehat{\Psi}$ is stored with its parent skyline set sequentially in the disk. By decomposing and storing the extended skyline this way, we apply Lemma 4 to exploit parent skyline for pruning unqualified tuples in the child set effectively, thereby reducing the query CPU cost (i.e., number of dominance comparisons).



Ψ tuples
$t_1\langle 0.2, 0.5\rangle$
$t_2\langle 0.6, 0.1\rangle$

$\widehat{\Psi}$ tuples	Parent $\mathcal{P}$
$t_3\langle 0.5, 0.5\rangle$	t_1
$t_4\langle 0.7, 0.1\rangle$	t_2

(a) A set of extended skyline tuples (b) Storing extended skyline

Fig. 9. Extended skyline of a group

Lemma 4: **Parent-based pruning.**
Given a sub-dataset $\mathcal{D}' \subseteq \mathcal{D}$ and a skyline attribute set S. Let S_Q be any skyline attribute set that satisfies $S_Q \subseteq S$. Given a tuple $t \in \widehat{\Psi}$, if $\exists\ t' \in \mathcal{P}_S(t),\ t' \notin \Psi(\mathcal{D}', S_Q)$, then $t \notin \Psi(\mathcal{D}', S_Q)$.

Proof: It is given that $t' \in \mathcal{P}_S(t)$, so we have: $t' \succ_S t$. By imposing the condition $S_Q \subseteq S$ on the above, we obtain: $\forall\ A_i \in S_Q,\ t'[A_i] \leq t[A_i]$.

It is also given that $t' \notin \Psi(\mathcal{D}', S_Q)$. Thus, there exists a tuple $t'' \in \Psi(\mathcal{D}', S_Q)$ such that $t'' \succ_{S_Q} t'$. By combining both $t'' \succ_{S_Q} t'$ and $\forall\ A_i \in S_Q,\ t'[A_i] \leq t[A_i]$, we derive: $t'' \succ_{S_Q} t$. This implies that t does not belong to the result set $\Psi(\mathcal{D}', S_Q)$. □

Therefore, the idea of the query algorithm is as follows. For a cell g, we first compute the projected skyline $\Psi^+_{S_Q}$ on the skyline set Ψ^+ along the query sub-space S_Q. These tuples are part of the final skyline for that cell. Then, we examine the child set $\widehat{\Psi}$ and a tuple $t \in \widehat{\Psi}$ is pruned when $\mathcal{P}(t) \notin \Psi^+_{S_Q}$. Finally, we compare the remaining tuples with the projected skyline $\Psi^+_{S_Q}$ on the query sub-space S_Q and those still remain as skyline are also added as the final skyline for that cell.

Query processing algorithm.
In the above discussion, we require the cuboid $\mathcal{C}^+(G, S)$ and

the query $Q(G_Q, S_Q)$ to have the same group-by attributes, i.e., $G_Q = G$.

We now present the query processing method in Algorithm 1, which handles the general case where $G_Q \subseteq G$. It assumes that a particular cuboid $\mathcal{C}^+$ has been chosen for processing the query. The issues on choosing a cuboid and quantifying its 'goodness' will be studied in Section 4.3.

For the general case $G_Q \subseteq G$, a group instance g_Q of G_Q corresponds to a subset V_c of cells of $\mathcal{C}^+$ (see Line 3). At Lines 4–7, we extract the skyline set $\Psi(g_c)$ of each cell $g_c \in V_c$ as the set T_{first}, and then compute its skyline (along S_Q) as the candidate set R_{first}. At Lines 9–12, we examine the child set $\widehat{\Psi}(g_c)$ of each cell $g_c \in V_c$. A tuple t from a child set is inserted into the candidate set T_{second} if all parents of t exist in the set R_{first}. Finally, we merge the sets R_{first} and T_{second} together, and compute the skyline of the merged tuple set. The specific skyline computation algorithm used at Line 13 will be clarified in Section 4.3.

Algorithm 1 Query Processing with an ES-cuboid

Input: Cuboid $\mathcal{C}^+(G, S)$, Query $Q(G_Q, S_Q)$
Precondition: $G_Q \subseteq G$ and $S_Q \subseteq S$
1: apply a grouping algorithm on the cells of $\mathcal{C}^+$ according to G_Q;
2: **for each** non-empty group instance g_Q of G_Q **do**
3: let V_c be the set of cells of $\mathcal{C}^+$ that are grouped as g_Q;
4: T_{first}:=$\emptyset$;
5: **for each** cell g_c in V_C **do**
6: T_{first}:=$T_{first} \cup \Psi(g_c)$;
7: R_{first}:=compute the skyline of T_{first};
8: T_{second}:=$\emptyset$;
9: **for each** cell g_c in V_c **do**
10: **for each** tuple t in $\widehat{\Psi}(g_c)$ **do**
11: **if** $\forall\, t' \in \mathcal{P}_S(t),\ t' \in T_{first}$ **then**
12: T_{second}:=$T_{second} \cup \{t\}$;
13: R_{final}:=compute the skyline of $R_{first} \cup T_{second}$;

4.3 Partial Materialization

In order to ensure fast on-line group-by skyline query processing, it is often desirable to pre-compute/materialize the extended skyline cuboids in the ES-cube. Since it is not a practical option to materialize all ES-cuboids, we focus on *partial materialization* [15], which is one of the most popular adopted methods in commercial products. Given a specific space budget, we greedily choose to materialize a subset of ES-cuboids that can bring the maximum query processing improvement.

In traditional data cube, the selection of a cuboid for materialization is based on a linear cost model, i.e., that the cost of evaluating a query using a cuboid $\mathcal{C}$ is linear to the size of $\mathcal{C}$, i.e., the I/O cost of scanning $\mathcal{C}$ once (or at most twice). Recent work in skyline [7], [38] has mentioned that the CPU cost contributes a significant portion of the overall skyline query processing cost. However, does that argument also hold in group-by skyline processing? If yes, we shall really consider the CPU cost in selecting which ES-cuboid to be materialized. To answer the question, we have carried out an investigation experiment. Here, we use the default parameter setting as in the experimental section and we measure the wall clock time of answering 100 random valid group-by skyline queries from a set of materialized ES-cuboids. In this paper and in the experiment, we adopt OSP [37], the best skyline algorithm to-date, for computing skyline. Our findings are: "*only 26.9% of the execution time is spent on the I/O cost (of reading an ES-cuboid from disk), whereas skyline computation shares 55.7% of the time. The remaining 17.4% of time contributes to the grouping operation and the other overhead.*"

Therefore, it is important to capture the CPU cost in selecting a cuboid to be materialized. Now, we define the benefit $\mathcal{B}$ of including an ES-cuboid $\mathcal{C}^+(G, S)$ into $\mathcal{MT}$ (which represents the set of ES-cuboids that should be materialized) as follows. Let $\mathcal{PA}_Q(\mathcal{C}^+)$ and $\mathcal{CP}_Q(\mathcal{C}^+)$ be the I/O cost (page accesses) and CPU cost (number of comparisons) of evaluating any group-by skyline query $Q(G_Q, S_Q)$ using an ES-cuboid $\mathcal{C}^+(G, S)$ (where $G_Q \subseteq G, S_Q \subseteq S$). The benefit $\mathcal{B}$ of including ES-cuboid $\mathcal{C}^+(G, S)$ into $\mathcal{MT}$ is based on how much gain in terms of I/O accesses and tuple comparisons if $\mathcal{C}^+(G, S)$ is materialized:

$$\sum_{G_Q \subseteq G, S_Q \subseteq S} \max\{0, \min_{\mathcal{C}_i^+(G',S') \in \mathcal{MT}, G_Q \subseteq G', S_Q \subseteq S'} \mathcal{T}_{IO}(\mathcal{PA}_Q(\mathcal{C}_i^+) - \mathcal{PA}_Q(\mathcal{C}^+)) + \mathcal{T}_{CP}(\mathcal{CP}_Q(\mathcal{C}_i^+) - \mathcal{CP}_Q(\mathcal{C}^+))\} \quad (3)$$

where $\mathcal{T}_{IO}$ denotes the time of accessing a disk page, $\mathcal{T}_{CP}$ denotes the time of performing a dominance comparison, and $\mathcal{C}_i^+(G', S')$ is any ES-cuboid in $\mathcal{MT}$ that could be used to answer Q. $\mathcal{T}_{IO}$ and $\mathcal{T}_{CP}$ are machine-specific system parameters. They are usually stored in the system catalog or can be derived, for example, by measuring the number of page accesses and the number of dominance comparisons that can be done in one second.

Regarding the cuboid selection problem, we apply the greedy approach (as described in Section 2.1) for picking cuboids to be included into the set $\mathcal{MT}$. The only difference is that, Equation 3 is used to compute the benefit of each cuboid. Next, we elaborate the I/O and CPU component costs of Equation 3.

4.3.1 Derivation of $\mathcal{PA}_Q(\mathcal{C}^+)$

We proceed to discuss the first component of benefit, i.e., the I/O cost $\mathcal{PA}_Q(\mathcal{C}^+)$ of evaluating a query $Q(G_Q, S_Q)$ by using an ES-cuboid $\mathcal{C}^+(G, S)$ (where $G_Q \subseteq G, S_Q \subseteq S$).

We assume that the main memory is large enough to hold the ES-cuboid $\mathcal{C}^+$. It suffices to read $\mathcal{C}^+$ from the disk once. Both the grouping and skyline operations are performed solely in main memory, so they do not incur additional I/O cost. Thus, $\mathcal{PA}_Q(\mathcal{C}^+)$ can be derived as the I/O cost of scanning $\mathcal{C}^+$ once. Hence, the I/O cost $\mathcal{PA}_Q(\mathcal{C}^+)$ depends on the *size* (i.e., the number of tuples and the number of attributes) of $\mathcal{C}^+$ but it is independent of Q.

If the domain of the attributes are real-values (i.e., very few tuples share duplicate values), the size of an ES-cuboid is close to the size of a group-by skyline cuboid. In that case, the size of an ES-cuboid can be estimated by summing the estimated skyline size of each cell of tuples in the cuboid through the skyline cardinality estimation functions developed in [7], [38]. However, when the domain of an attribute is finite (e.g., an

integer domain of (say) [0,99k] for number of rebounds in the NBA dataset), then the size of an ES-cuboid would be larger than the size of a group-by skyline cuboid. Consequently, in order to compute $\mathcal{PA}_Q(\mathcal{C}^+)$, we first devise techniques to estimate the size of an ES-cuboid, or essentially, the size of an extended skyline, on a finite domain. After that, we will come back to the derivation of $\mathcal{PA}_Q(\mathcal{C}^+)$.

Estimating extended skyline size.
Figure 9a illustrates a set of tuples with two skyline dimensions S_1 and S_2. The skyline set contains t_1, t_2, which collectively form a (dotted) "contour" in the figure and the extended skyline set contains all tuples on the contour: t_1, t_2, t_3, t_4. Suppose that each skyline attribute has an integer domain $[0, \theta]$, i.e., the number of possible values is $\theta + 1$. A tuple t belongs to the extended skyline if there exists no tuple $t' \in \mathcal{D}$ that strictly dominates it.

We now generalize the above observation to a dataset with a set $\mathcal{S}$ of skyline dimensions and N tuples. Let $\Xi(S)$ be the space of all possible points t, satisfying $t[A_i] \in [0, \theta],\ \forall A_i \in S$. Clearly, there are $(\theta + 1)^{|S|}$ possible distinct points in the space domain. Given a point $t \in \Xi(S)$, we define its *probability density* as:

$$p(t, S) = \frac{|\{t' \in \mathcal{D} \mid t'[A_i] = t[A_i],\ \forall A_i \in S\}|}{N} \tag{4}$$

and its *cumulative strict dominated density* as:

$$P(t, S) = \sum_{t' \in \Xi(S),\ \forall A_i \in S, t'[A_i] < t[A_i]} p(t', S)$$

Each possible point has a probability of $p(t, S)$ being an actual tuple in the dataset $\mathcal{D}$, which contains N tuples. The probability of t being a extended skyline tuple is: $(1 - P(t, S))^N$. Thus, we obtain the size of an extended skyline of N data tuples as:

$$\Phi_{ES}(N) = \sum_{t \in \Xi(S)} p(t, S)\,(1 - P(t, S))^N \tag{5}$$

The above equations (containing $(\theta + 1)^{|S|}$ terms) can be efficiently computed by using a numerical technique called midpoint approximation [18].

Derivation of $\mathcal{PA}_Q(\mathcal{C}^+)$, re-visit.
Now, we can derive $\mathcal{PA}_Q(\mathcal{C}^+)$ by using Equation 5. Recall that $\mathcal{PA}_Q(\mathcal{C}^+)$ can be simply derived as the I/O cost of scanning $\mathcal{C}^+$ once, let ω be the number of distinct groups in $\mathcal{C}^+$ and each group g_i has $|\mathcal{D}(g_i)|$ tuples, we have:

$$\mathcal{PA}_Q(\mathcal{C}^+) = \frac{1 + |G| + |S|}{B_{item}} \cdot \sum_{i=1}^{\omega} \Phi_{ES}(|\mathcal{D}(g_i)|) \tag{6}$$

whereas the factor $(1 + |G| + |S|)$ accounts for the size of a tuple in $\mathcal{C}$ (a tuple ID takes 1 unit of storage), and B_{item} represents the number of items/attributes that fit in a disk page.

Given a dataset $\mathcal{D}$ and a group by skyline query $Q = (G_Q, S_Q)$, the number of distinct groups ω and the number of tuples of each group $|\mathcal{D}(g_i)|$ can be easily derived from the data distribution using some basic text-book formulae. Please refer to [22] for details.

4.3.2 Derivation of $\mathcal{CP}_Q(\mathcal{C}^+)$

We now discuss the second component of benefit (Equation 3), i.e., the CPU cost $\mathcal{CP}_Q(\mathcal{C}^+)$ (i.e., number of dominance comparisons) of evaluating the query $Q(G_Q, S_Q)$ by using an ES-cuboid $\mathcal{C}^+(G, S)$ (where $G_Q \subseteq G, S_Q \subseteq S$) for arbitrary data distribution. The model is derived based on OSP [37], the algorithm that we adopted in query processing, which is the most efficient index-free skyline algorithm to-date.

Suppose that we have a multi-dimensional histogram H_C that models the distribution of tuples of $\mathcal{C}^+$ in the space of S with values in histogram buckets are normalized to have a sum of 1. The CPU cost of OSP can be estimated by (i) construct a histogram H_Q (for the data distribution of $\mathcal{C}^+$ in the subspace S_Q); (ii) apply H_Q to estimate the skyline locations in the space S_Q; and (iii) build an OSP tree with the estimated skyline and estimate the query CPU cost. Step (i) can be efficiently done by projecting the dimensions of S_Q from the original histogram H_C. Step (ii) can be implemented by the approximate skyline location estimation technique in [23]. Thus, our main focus is on Step (iii).

Histogram-based CPU cost estimation of OSP.
It remains to discuss how we utilize the histogram H_C for estimating the CPU cost of OSP. We suppose that the readers are already familiar with how OSP works, as described in Section 2.2. The subsequent example follows the setting of the OSP example described in Figure 4.

We shall use the terms 'point' and 'OSP tree node' interchangeably in the discussion below. Assume, after estimating the skyline locations (i.e., after Steps (i) and (ii) we mentioned above), the points of a cell on a sub-space $S_Q = \{S_1, S_2\}$ are shown in the left hand side of Figure 10 and those points form an OSP tree on the right hand side of the figure.

Let t be a tree node (with the point t). We use t^p to represent the parent of t; it is null if t is the root. The *bounding region* $t.\gamma$ of t is a hyper-rectangle formed by its lower corner $t.\gamma^-$ and upper corner $t.\gamma^+$, which are in turn defined as:

$$t.\gamma^-[S_i], t.\gamma^+[S_i] \quad = \quad \begin{cases} 0, 1 & \text{if } (t^p = \text{null}) \\ t^p.\gamma^-[S_i], t^p[S_i] & \text{if } (t[S_i] < t^p[S_i]) \\ t^p[S_i], t^p.\gamma^+[S_i] & \text{if } (t[S_i] \geq t^p[S_i]) \end{cases}$$

For instance, in Figure 10, node t_1 has no parent, so we have $t_1.\gamma = [0, 1) \times [0, 1)$. Node t_2 is the 'top-left' child of t_1 so we have $t_2.\gamma = [0, 0.3) \times [0.4, 1)$.

Let v be the next (arbitrary) point in the space to be visited by OSP, t be a node in the OSP tree, and $\Upsilon(t)$ be the set of ancestor tree nodes of t. In OSP, we need to perform a dominance comparison between v and t iff t gets visited. This happens when (i) the lower corner of t dominates v, i.e., $t.\gamma^- \succ v$, and (ii) none of the ancestor nodes of t dominates v, i.e., $\bigwedge_{t' \in \Upsilon(t)} t' \not\succ v$. To capture all the possible points that cause a node t to be visited, we define a region R_t for t so that points in R_t satisfy conditions (i) and (ii). Consequently, R_t can be written as $R_t = R^t_{dom} - R^t_{anc}$, where R^t_{dom} is the region dominated by $t.\gamma^-$ (condition (i)), and R^t_{anc} is the (union of) region dominated by $t'.\gamma^-$ for any ancestor $t' \in \Upsilon(t)$ (condition (ii)). For example,

for node t_2, its lower corner $t_2.\gamma^-$ dominates the region $R^{t_2}_{dom} = [0,1) \times [0.4,1)$ and its parent t_1 dominates the region $R^{t_2}_{anc} = [0.3,1) \times [0.4,1)$. Thus, any points that fall into the region $R^{t_2}_{dom} - R^{t_2}_{anc} = [0,0.3) \times [0.4,1)$ cause node t_2 to be visited.

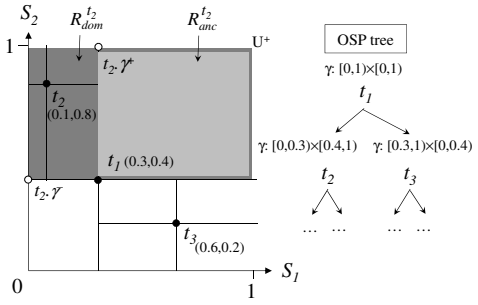


Fig. 10. Estimation of the comparisons on the OSP tree

Given the histogram H_Q, we use $H_Q(R)$ and $H_Q(v_1, v_2)$ to denote the probability density of a region R and the region formed by the corners v_1 and v_2, respectively. Therefore, the number of dominance comparisons of OSP is estimated as:

$$\begin{aligned} \mathcal{CP}_Q(\mathcal{C}^+) &= \omega \cdot \sum_{t \in \Gamma} \frac{N}{\omega} \cdot H_Q(R_t) \\ &= N \cdot \sum_{t \in \Gamma} \left(H_Q(t.\gamma^-, \mathbb{U}^+_Q) - \sum_{t' \in \Upsilon(t)} H_Q(t', t'.\gamma^+) \right) \end{aligned}$$

where Γ denotes the set of OSP tree nodes and $\mathbb{U}^+_Q$ denotes the upper corner of the (skyline) space domain in H_Q.

Note that the above equation is a pessimistic estimate of the CPU cost because it does not capture the pruning effect of Lemma 4. Nevertheless, this cost model is adequate to provide good estimates — our experiments show that this cost model yields query improvement close to that offered by an optimal cost model.

4.4 Construction

In this section, we develop an efficient method for constructing the cuboids of $\mathcal{MT}$, which denotes the set of chosen ES-cuboids according to Section 4.3. For the running example here, suppose that we use the dataset shown in Figure 5 and the set $\mathcal{MT}$ contains the following ES-cuboids (see their tuples in Figure 8):

- $A^+ : \mathcal{C}^+(G_2G_3, S_1S_2)$
- $C^+ : \mathcal{C}^+(G_2G_3, S_2)$
- $D^+ : \mathcal{C}^+(G_2, S_1S_2)$
- $F^+ : \mathcal{C}^+(G_2, S_2)$

A simple method for constructing a cuboid $C^+(G,S) \in \mathcal{MT}$ is to take the dataset $\mathcal{D}$ as input, apply the group-by operator on $\mathcal{D}$ with the group-by attribute set G, and then apply the skyline operator (e.g., OSP) as the "aggregation function" on each group with the skyline attribute set S. However, this method is slow as it does not exploit the sharing of computations among cuboids.

We then discuss how to construct these ES-cuboids in an efficient manner. Let $\mathcal{AMT}$ be the set of constructed cuboids, which is empty in the beginning. According to Lemma 3, a cuboid $\mathcal{C}^+(G',S')$ can be derived from an ancestor cuboid $\mathcal{C}^+(G,S)$, i.e., $G' \subseteq G$ and $S' \subseteq S$ (provided that such a cuboid $\mathcal{C}^+(G,S)$ has been constructed, i.e., it exists in $\mathcal{AMT}$).

In order to utilize this sharing effect, we propose to sort the (chosen) ES-cuboids of $\mathcal{MT}$ in the *topological order* of their attributes, such that a cuboid appears after all its (possible) ancestors. For instance, the ES-cuboids are sorted in the order: $\mathcal{C}^+(G_2G_3, S_1S_2), \mathcal{C}^+(G_2G_3, S_2), \mathcal{C}^+(G_2, S_1S_2)$, and $\mathcal{C}^+(G_2, S_2)$.

Continuing with the example, we first build $\mathcal{C}^+(G_2G_3, S_1S_2)$ and insert it into $\mathcal{AMT}$. For the next two cuboids $\mathcal{C}^+(G_2G_3, S_2)$ and $\mathcal{C}^+(G_2, S_1S_2)$, we build them from their ancestors $\mathcal{C}^+(G_2G_3, S_1S_2)$ (according to Lemma 3). These two cuboids are then inserted into $\mathcal{AMT}$. Finally, we attempt to construct the cuboid $\mathcal{C}^+(G_2, S_2)$ and find that it has multiple ancestors: $\mathcal{C}^+(G_2G_3, S_1S_2)$, $\mathcal{C}^+(G_2G_3, S_2)$, $\mathcal{C}^+(G_2, S_1S_2)$. These three cuboids contain 5, 4, and 4 tuples, respectively, as shown in Figure 8.

Regarding the cuboid used for constructing $\mathcal{C}^+(G_2, S_2)$, a sensible choice would be to use its smallest ancestor cuboid $\mathcal{C}^+(G_2G_3, S_2)$ (having 4 tuples), which incurs low construction cost.

Algorithm 2 is the pseudo-code for constructing ES-cuboids efficiently. It takes the set $\mathcal{MT}$ of chosen cuboids as input. Let $\mathcal{AMT}$ be the set of constructed cuboids, initialized to the empty set. First, we sort the cuboids of $\mathcal{MT}$ by the topological order. For each cuboid $\mathcal{C}^+(G',S')$ in the sorted $\mathcal{MT}$, we find its set Ω of ancestor cuboids that have been constructed (i.e., found in $\mathcal{AMT}$). Then, we pick the smallest ancestor cuboid $\mathcal{C}^*$ from Ω, for constructing $\mathcal{C}^+(G',S')$. After that, we insert $\mathcal{C}^+(G',S')$ into $\mathcal{AMT}$ for subsequent use.

Algorithm 2 Cuboid Construction Algorithm

```
Input: Set of chosen ES-cuboids MT
AMT:=∅;                                   ▷ the set of materialized cuboids
sort the ES-cuboids of MT by the topological order;
for each cuboid C+(G',S') ∈ MT do
    Ω:={C+(G,S) ∈ AMT | G' ⊆ G, S' ⊆ S};
    C*:=the smallest cuboid in Ω;
    use C* to construct C+(G',S');
    insert C+(G',S') into AMT;
```

5 Experiments

In this section, we experimentally evaluate the efficiency and effectiveness of the proposed technique (we name it as ES). The objective is to evaluate the query I/O cost (i.e., disk page accesses), CPU cost (i.e., dominance comparisons), and cube construction time (if applicable) of using the proposed technique with respect to a workload of 100 randomly generated group-by skyline queries, each of which has a random number of 1–5 group-by attributes and 1–5 skyline attributes. All the experiments were conducted on an Intel Core 2 Quad 2.83 GHz PC with 4GB of memory. All the algorithms were

implemented using C++. The page size is set to 4K Bytes. We use the term 'budget-to-data ratio' to represent the ratio of the datacube budget space (in disk pages) to the dataset storage size (in disk pages). The default budget-to-data ratio (for datacube construction) is set to 10:1.

5.1 Description of Datasets and Methods

We have carried out experiments on synthetic datasets and three real datasets (NBA, Baseball, and Forest Cover).

- **NBA.**
  It stores the NBA players' statistics[2] from 1946 to now. It contains 20,788 tuples, where each tuple stores the statistics of a player in a season. We identify attributes 'Year' and 'Team' as grouping dimensions and attributes 'Games played', 'Points', 'Rebounds', 'Assists', 'Steals', 'Blocks', 'Free throws', and '3-point shots' as the measures (i.e., skyline dimensions).
- **Baseball (BB).**
  It stores the baseball batters' statistics[3] from 1871 to now. It contains 88,686 tuples, where each tuple stores the statistics of a batter in a season. We identify attributes 'Year', 'Team', and 'League' as grouping dimensions and attributes 'Games', 'At bats', 'Runs', 'Hits', 'Doubles', and 'Runs batted in' as the measures (i.e., skyline dimensions).
- **Forest Cover (FC).**
  It is obtained from UCI Machine Learning Repository[4]. It stores 581,012 tuples, each representing a 30×30 forest land cell. We identify attributes 'Soil type A', 'Soil type B', 'Slope', and 'Cover type' as grouping dimensions and attributes 'Roadway distance', 'Vertical hydrology', 'Horizontal hydrology' 'Hill-shade at am', 'Hill-shade at noon', and 'Hill-shade at pm' as skyline dimensions.
- **Synthetic datasets.**
  Our synthetic datasets are generated similar to the benchmark setting described in [7]. Unless stated otherwise, each dataset contains $N = 1,000,000$ tuples, with a total of 14 attributes (a_1, a_2, ..., a_{14}): 5 attributes $a_1 \ldots a_5$ are for grouping, and 9 attributes $a_6 \ldots a_{14}$ are for skyline. Specifically, $a_6 \ldots a_8$ are generated independently, $a_9 \ldots a_{11}$ are correlated, and $a_{12} \ldots a_{14}$ are anti-correlated. By default, the domain size β of each group-by attribute is 5 and the domain size θ of each skyline attribute is 1000.

In our study, we mainly compare our approach (ES-cube) with two baseline approaches.

- **Relational (REL).**
  This method evaluates a query using a relational engine that supports group-by skyline queries [22]. It does not exploit any materialized cuboids.
- **Projection data cube (PROJ).**
  This method does not distinguish between grouping dimensions and skyline dimensions and follows the method in [15] to select the cuboids. The cuboid selection algorithm strictly follows the linear cost model. As a result, it considers only the I/O costs by setting $\mathcal{T}_{CP}$ of Equation 3 to zero. Also, in order to find skyline, each cell materializes the set of tuples belong to that cell (only the projected values of the selected dimensions are stored), instead of an aggregated value.

5.2 Experimental Results on Real Data

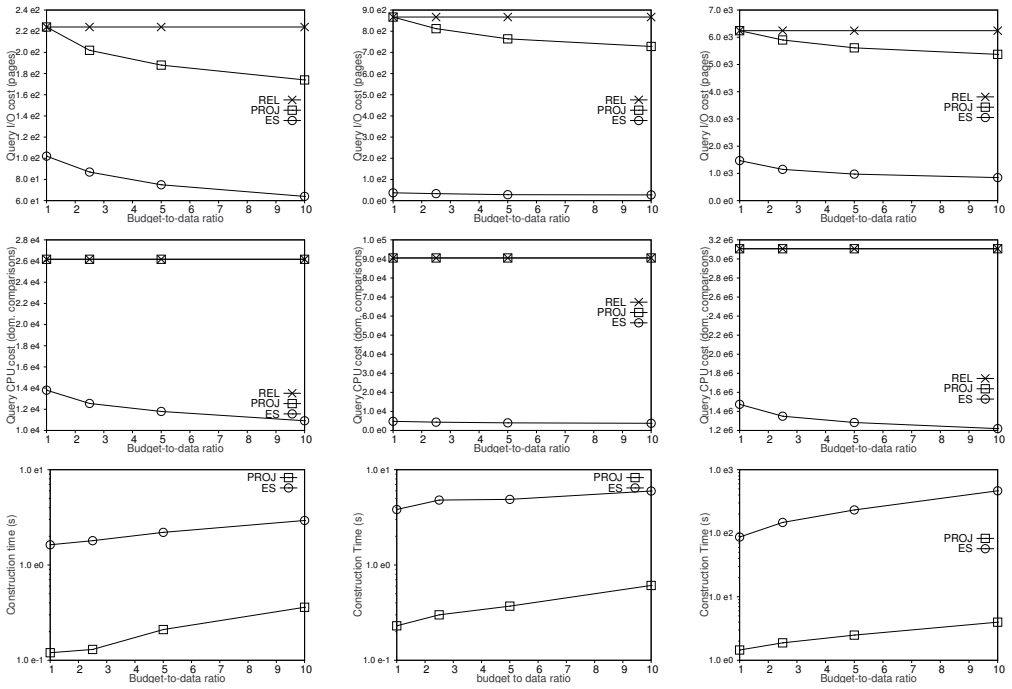
Comparison with baseline approaches.
We first compare the performance of ES with the baseline methods REL and PROJ on real datasets. Figure 11 shows the query I/O cost, CPU cost, and construction time of the methods on the real datasets under different budget-to-data ratio. Observe that the query cost of REL remains constant as it does not materialize any cuboids. The query I/O costs offered by PROJ and ES gradually reduce when the budget-to-data ratio increases. This is consistent with the behavior of the greedy materialization on traditional datacubes [15] — the early selected cuboids provide more significant query cost savings. Given the same budget, ES is able to materialize more cuboids than PROJ, thus ES offers better query I/O cost saving than PROJ. In addition to the reduction of I/O, ES can dramatically reduce the CPU cost when compared with PROJ and REL, even for cases with tight space budget. Note that PROJ does not reduce any comparisons because the cuboid materialized by PROJ contains all the tuples (but with fewer attributes). The construction time rises slowly when the budget-to-data ratio increases.

Comparison with other cost models.
In order to evaluate the proposed cost model, we compare the performance of ES with two alternate cost models:

- **ES-cube with optimal cost model (OPT).**
  This method is a variant of ES. It differs from ES in the following: (1) instead of estimating the cuboid size, it really materializes every cuboid ahead in order to know their actual sizes, and (2) it utilizes the actual values obtained from the materialized cuboids to construct an optimal model for cost estimation. Note that this method is impractical due to its very high construction time and tremendous storage space for storing all cuboids. It merely serves as an indicator of the lowest query cost that can be achieved with a perfect cost model.
- **ES-cube with random strategy (RAN).**
  This method is a variant of ES, except that each time a random cuboid is chosen to be materialized.

Figure 12 shows the query I/O cost, CPU cost, and construction time of these methods on real datasets, as a function of the budget-to-data ratio. We can see that the query performance offered by ES is much better than RAN and it stays very close to OPT. Furthermore, the construction time of ES is

2. http://basketballreference.com/stats_download.htm
3. http://baseball1.com/statistics/
4. http://archive.ics.uci.edu/ml/



(a) NBA (b) BB (c) FC

Fig. 11. Performance vs. budget-to-data ratio, on real datasets

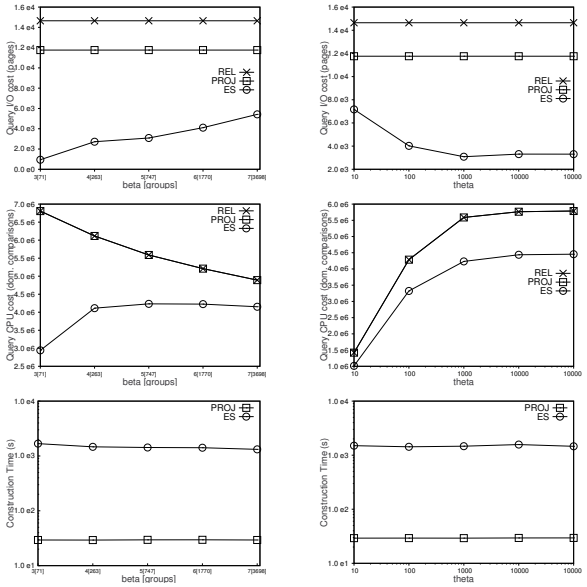
significantly lower than that of OPT. Because of that, we will simply ignore RAN and OPT in the remaining experiments.

5.3 Experimental Results on Synthetic Data

In the following, we study the effects of various parameters on the performance of the methods.

Varying the group-by attribute domain size β.
Figure 13a investigates the impact of the group-by attribute domain size β on the construction time and query costs of the three methods. Each square-bracketed number (along the x-axis) represents the average number of measured groups per query in the workload. When β is small, the size of an ES-cuboid becomes small. Thus, more ES-cuboids could be constructed from the fixed space budget. During query processing, small ES-cuboids are accessed for answering queries, so ES has small query I/O cost. Since the size of a PROJ-cuboid is independent of β, it has constant construction time and query I/O cost. At a small β value, the number of tuples per group in a PROJ-cuboid is high, causing PROJ to have high query CPU time. However, the CPU cost trend of ES is slightly different from that of PROJ. Even at a small β value, the number of tuples per group in an ES-cuboid remains small, and thus the CPU cost of ES is not high.

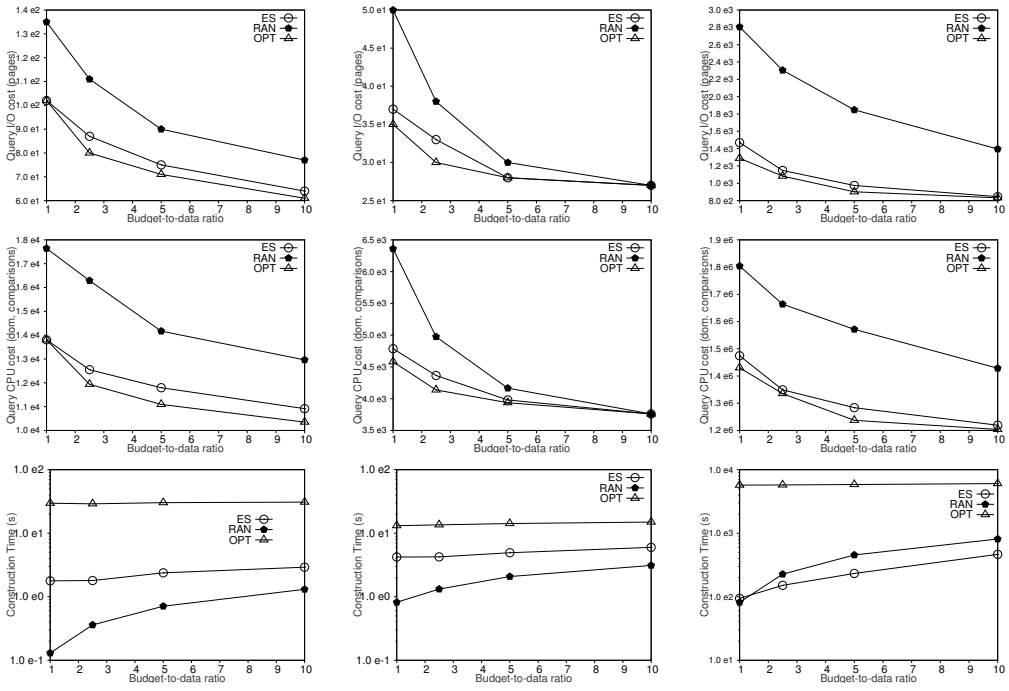
Varying the skyline attribute domain size θ.
We then study the effect of the skyline attribute domain size θ on the construction time and query cost of all methods (see Figure 13b). When θ is small, a tuple t has high probability of belonging to the extended skyline of its group. As a result, the size of each materialized cuboid is large, so ES materializes few cuboids within the given budget. Therefore, ES incurs more query I/O cost at a low θ value. On the other hand, all methods have lower CPU cost at a low θ value, as it becomes easier to obtain a tuple with low values in skyline attributes, which helps pruning other unqualified tuples effectively.



(a) vs. group-by domain β (b) vs. skyline domain size θ

Fig. 13. Effect of attribute domain size

Varying the data size N.
Figure 14 shows the construction time and query costs of REL, PROJ and ES with respect to the data size. When N rises, the extended skyline size of a group increases sub-linearly with N. Thus, the query cost of ES grows sub-linearly. The performance gap between ES and the baseline methods widens



(a) NBA (b) BB (c) FC

Fig. 12. Comparison of cost models, on real datasets

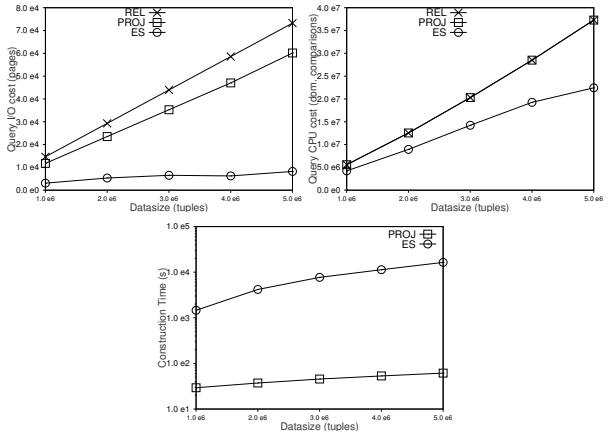
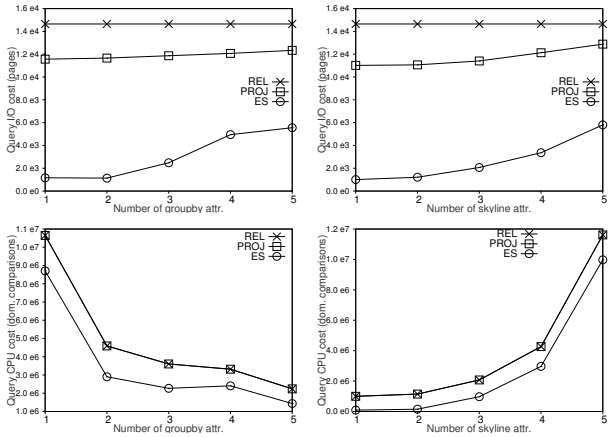


Fig. 14. Effect of the data size

when N increases.

Varying the number of query attributes.
Our default workload contains queries with random 1–5 group-by attributes and 1–5 skyline attributes. We now study the effect of group-by and skyline attributes on the query cost. ES outperforms REL and PROJ in all cases. Figure 15a shows the query cost when we control the number of group-by attributes in queries. Observe that ES saves significant I/O cost and CPU cost over REL and PROJ in all cases. When a query has a higher number of group-by attributes, ES answers it with a larger cuboid, thus incurring higher I/O cost. However, when there are more group-by attributes, the number of groups increases and thus there are fewer tuples (and also skyline tuples) per group, so the CPU cost decreases.

Figure 15b shows the query cost when we control the number of skyline attributes in queries. ES also outperforms REL and PROJ in all cases. When a query contains more skyline attributes, ES answers it with a larger cuboid, thus incurring higher I/O cost and CPU cost. Since the size of skyline is $O((\ln n)^{d-1}/(d-1)!)$ (on a uniform dataset with n tuples and d attributes) [11], it explains the increasing trend of the query costs of all methods.



(a) vs. group-by attributes (b) vs. skyline attributes

Fig. 15. Effect of the number of query attributes

6 Conclusions and Future Work

This paper studies the support of group-by skyline queries in data warehouses by proposing the concept of group-by skyline cube. To implement this concept, we have developed techniques for addressing the following important issues: (i) the concept of ES-cuboids for re-enabling the sharing of

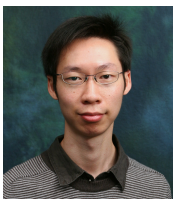
computational efforts between cuboids, (ii) an algorithm for answering a query from a materialized cuboid efficiently, (iii) cost models for facilitating the selection of cuboids to be materialized for the the best query performance improvement, and (iv) an efficient algorithm for constructing cuboids. Experimental results show that the proposal techniques significantly reduce the query costs in terms of both CPU time and I/O time.

Our future work includes incremental maintenance of skyline cube. We also plan to study the integration of other useful decision-making operators such as top-k into data-warehouses as "aggregate functions". Furthermore, throughout this paper, we assume that smaller values are preferable over larger ones, for skyline attributes. We plan to remove this limitation by defining the concept of generic ES-cuboid, which consists of the union of ES-cuboids obtained from different combinations of preferences on skyline attributes. Such a generic ES-cuboid can then be used to answer a query irrespective of its preferences on attributes. An important issue would be to study an efficient method for estimating the size of a generic ES-cuboid, without having to enumerate all combinations of preferences.

REFERENCES

[1] W. T. Balke, U. Güntzer, and J. X. Zheng. Efficient Distributed Skylining for Web Information Systems. In *EDBT*, 2004.
[2] I. Bartolini, P. Ciaccia, and M. Patella. Efficient Sort-Based Skyline Evaluation. *ACM TODS*, 33(4), 2008.
[3] K. S. Beyer et al. Bottom-Up Computation of Sparse and Iceberg CUBEs. In *SIGMOD*, 1999.
[4] S. Börzsönyi et al. The Skyline Operator. In *ICDE*, 2001.
[5] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. Tung, and Z. Zhang. Finding k-Dominant Skylines in High Dimensional Space. In *SIGMOD*, 2006.
[6] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. Tung, and Z. Zhang. On High Dimensional Skylines. In *EDBT*, 2006.
[7] S. Chaudhuri et al. Robust Cardinality and Cost Estimation for Skyline Operator. In *ICDE*, 2006.
[8] J. Chomicki et al. Skyline with Presorting. In *ICDE*, 2003.
[9] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou. Parallel Distributed Processing of Constrained Skyline Queries by Filtering. In *ICDE*, 2008.
[10] E. Dellis et al. Efficient Computation of Reverse Skyline Queries. In *VLDB*, 2007.
[11] P. Godfrey. Skyline Cardinality for Relational Processing. In *FoIKS*, 2004.
[12] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and Analysis for Maximal Vector Computation. *VLDB J.*, 16(1):5–28, 2007.
[13] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In *ICDE*, 1996.
[14] A. Gupta and I. S. Mumick. *Materialized views: techniques, implementations, and applications*. MIT Press, 1999.
[15] V. Harinarayan et al. Implementing Data Cubes Efficiently. In *SIGMOD*, 1996.
[16] W. Jin, A. K. H. Tung, M. Ester, and J. Han. On Efficient Processing of Subspace Skyline Queries on High Dimensional Data. In *SSDBM*, 2007.
[17] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB*, 2002.
[18] A. R. Krommer and C. W. Ueberhuber. *Numerical Integration on Advanced Computer Systems*, volume 848 of *Lecture Notes in Computer Science*. Springer, 1994.
[19] H. T. Kung et al. On Finding the Maxima of a Set of Vectors. *J. ACM*, 22(4):469–476, 1975.
[20] C. Li, B. C. Ooi, A. Tung, and S. Wang. DADA: A Data Cube for Dominant Relationship Analysis. In *SIGMOD*, 2006.
[21] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting Stars: The k Most Representative Skyline Operator. In *ICDE*, 2007.
[22] M.-H. Luk et al. Group-by skyline query processing in relational engines. In *CIKM*, 2009.
[23] D. Papadias et al. Progressive Skyline Computation in Database Systems. *TODS*, 30(1):41–82, 2005.
[24] J. Pei et al. Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces. In *VLDB*, 2005.
[25] J. Pei et al. Towards Multidimensional Subspace Skyline Analysis. *TODS*, 31(4):1335–1381, 2006.
[26] J. Pei et al. Computing Compressed Multidimensional Skyline Cubes Efficiently. In *ICDE*, 2007.
[27] K. A. Ross et al. Fast Computation of Sparse Datacubes. In *VLDB*, 1997.
[28] M. Sharifzadeh and C. Shahabi. The Spatial Skyline Queries. In *VLDB*, 2006.
[29] Y. Sismanis et al. Dwarf: Shrinking the PetaCube. In *SIGMOD*, 2002.
[30] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient Progressive Skyline Computation. In *VLDB*, 2001.
[31] Y. Tao et al. Efficient Skyline and Top-k Retrieval in Subspaces. *IEEE Trans. Knowl. Data Eng.*, 19(8):1072–1088, 2007.
[32] Y. Tao, X. Xiao, and J. Pei. SUBSKY: Efficient Computation of Skylines in Subspaces. In *ICDE*, 2006.
[33] A. Vlachou et al. SKYPEER: Efficient Subspace Skyline Computation over Distributed Data. In *ICDE*, pages 416–425, 2007.
[34] T. Xia et al. Refreshing the Sky: The Compressed Skycube with Efficient Support for Frequent Updates. In *SIGMOD*, pages 491–502, 2006.
[35] D. Xin et al. P-Cube: Answering Preference Queries in Multi-Dimensional Space. In *ICDE*, 2008.
[36] Y. Yuan et al. Efficient Computation of the Skyline Cube. In *VLDB*, 2005.
[37] S. Zhang et al. Scalable Skyline Computation Using Object-based Space Partitioning. In *SIGMOD*, 2009.
[38] Z. Zhang et al. Kernel-based skyline cardinality estimation. In *SIGMOD*, 2009.

Man Lung Yiu received the bachelors degree in computer engineering and the PhD degree in computer science from the University of Hong Kong in 2002 and 2006, respectively. Prior to his current post, he worked at Aalborg University for three years starting in the Fall of 2006. He is now an assistant professor in the Department of Computing, Hong Kong Polytechnic University. His research focuses on the management of complex data, in particular query processing topics on spatiotemporal data and multidimensional data.



Eric Lo received a PhD degree in 2007 from ETH Zurich. He is currently an assistant professor in the Department of Computing, Hong Kong Polytechnic University. He research interests include databases and software engineering.

Duncan Yung received Bachelor Degree in Computer Science in 2009 from the University of Hong Kong, Hong Kong. He is currently an MPhil student at the Department of Computing, Hong Kong Polytechnic University, Hong Kong, under the supervision of Dr Eric Lo. His research focuses on different kinds of spatial data management and authentication.