QSkycube: Efficient Skycube Computation Using Point-Based Space Partitioning

Jongwuk Lee, Seung-won Hwang
Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, Republic of Korea, 790-784
{julee, swhwang}@postech.ac.kr

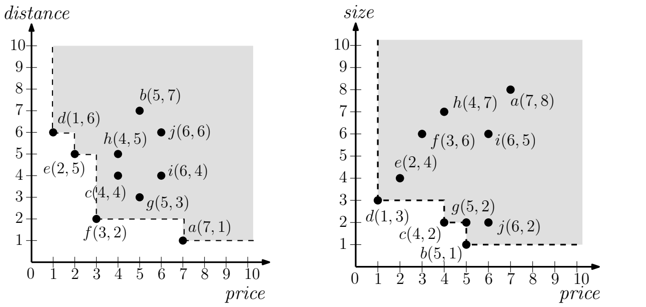
ABSTRACT

Skyline queries have gained considerable attention for multi-criteria analysis of large-scale datasets. However, the skyline queries are known to return too many results for high-dimensional data. To address this problem, a *skycube* is introduced to efficiently provide users with multiple skylines with different strengths. For efficient skycube construction, state-of-the-art algorithms amortized redundant computation among *subspace skylines*, or *cuboids*, either (1) in a bottom-up fashion with the principle of *sharing result* or (2) in a top-down fashion with the principle of *sharing structure*. However, we observed further room for optimization in both principles. This paper thus aims to design a more efficient skycube algorithm that shares multiple cuboids using more effective structures. Specifically, we first develop each principle by leveraging *multiple parents* and a *skytree*, representing *recursive point-based space partitioning*. We then design an efficient algorithm exploiting these principles. Experimental results demonstrate that our proposed algorithm is significantly faster than state-of-the-art skycube algorithms in extensive datasets.

1. INTRODUCTION

Skyline queries have gained considerable attention as an alternative operator to assist multi-criteria decision making in large-scale datasets. Given a dataset $\mathcal{S}$, a skyline query returns a set of "interesting" points, or a *skyline*, that are not "dominated" by any other points – a point p *dominates* another point q if p is no worse than q on all dimensions and is better than q on at least one dimension. However, the well-known downside of the skyline query retrieves too many results for high-dimensional data, called *the curse of dimensionality*.

This problem naturally leads to selecting a subset of dimensions to reduce the skyline to a manageable size. To illustrate this, we provide the following real-life scenario:



(a) Q1

(b) Q2

Figure 1: Two skyline queries using a toy dataset in two-dimensional space

Example 1 (Skyline query scenario) Consider a hotel retrieval system using a database Hotel(*name*, *price*, *distance*, *size*, *age*, *city*). The skyline query on all four numerical dimensions renders too many results. (Smaller values on all dimensions are preferred.) Users would thus narrow down dimensions of interest: (Q1) a user could find hotels $\{a, d, e, f\}$ that are cheap and close to the conference venue in Seattle using *price* and *distance* (Figure 1a); (Q2) another user could focus on different hotels $\{b, c, d\}$ using *price* and *size* (Figure 1b). Users could iteratively change the dimensions until the results converge to meaningful candidates.

This scenario has motivated us to study *skycube* computation [11, 16] which pre-materializes all possible $2^d - 1$ *subspace skylines*, or *cuboids*, for d-dimensional data. Once the skycube is built up, users can retrieve multiple relevant skylines with a fast response time, and identify meaningful skyline results. However, a naive skycube construction that computes skylines on every subspace incurs prohibitive cost.

For efficient construction, state-of-the-art skycube algorithms [11, 16] have focused on sharing computation across cuboids by extending well-known skyline algorithms SFS [2] and DC [1]. From these algorithms, we could observe the main principles of sharing computation:

- **Principle of sharing result**: Bottom-Up Skycube algorithm (BUS) computes a cuboid on subspace $\mathcal{U}$ by exploiting pre-computed cuboids on all subspaces of $\mathcal{U}$. That is, because skylines on subspaces of $\mathcal{U}$ can also be a skyline on $\mathcal{U}$, we can bypass the computation for the skylines in computing the skyline on $\mathcal{U}$.
- **Principle of sharing structure**: Top-Down Skycube algorithm (TDS) keeps a "working set" of multiple sub-

Articles from this volume were invited to present their results at The 37th International Conference on Very Large Data Bases, August 29th - September 3rd 2011, Seattle, Washington.
Proceedings of the VLDB Endowment, Vol. 4, No. 3

spaces, and amortizes the cost of computing skylines on the subspaces by sharing one "structure" for all the subspaces. Sharing structure can thus provide a new opportunity for sharing computation by materializing *point-wise dominance relationships* from preceding cuboid computation.

In addition, TDS leverages the sharing result principle. Specifically, when computing a skyline on $\mathcal{U}$, TDS exploits a pre-computed skyline on $\mathcal{V}$ such that $\mathcal{U} \subset \mathcal{V}$, instead of an entire dataset. TDS, by leveraging both principles of sharing result and structure, is reported to be faster than BUS, especially in high-dimensional data [11, 16].

Meanwhile, we observed further room for optimization of both principles.

First, regarding the sharing result principle, broadening the working set invites more optimization opportunities. For example, given a subspace set $\{D_1D_2D_3,\ D_1D_2D_4,\ D_1D_2\}$, TDS only leverages a "single parent relationship" in computing a skyline on D_1D_2. However, we can exploit skylines on both $D_1D_2D_3$ and $D_1D_2D_4$, which can be more efficient than using one or the other.

Second, regarding the sharing structure principle, TDS employs two-dimensional space partitioning derived from DC [1]. This structure not only is incapable of optimizing both *dominance* and *incomparability* in high-dimensional data but also is limited to sharing cuboids with multiple relationships. In clear contrast, we exploit *point-based space partitioning* [6, 17] with a fine-granularity structure.

This paper thus aims to design a more efficient skycube algorithm that shares multiple cuboids using more effective structures. Specifically, we first develop each principle by leveraging *multiple parents* and a *skytree*, derived from recursive point-based space partitioning. We then design an efficient skycube algorithm exploiting these principles.

To sum up, we believe that this paper makes the following contributions:

- We address the problem of optimizing skycube computation using point-based space partitioning (Section 3).
- We extend the principles of sharing result and sharing structure by leveraging multiple parents and a skytree respectively. (Section 4).
- We propose an efficient skycube algorithm that takes advantage of two computation sharing principles (Section 5).
- We evaluate the efficiency of our proposed algorithm by comparing state-of-the-art skycube algorithms in extensive synthetic datasets (Section 6).

The remainder of this paper is organized as follows. Section 2 presents the basics on skycube computation. Section 3 observes the limitations of existing work and explains point-based space partitioning. Section 4 extends two computation sharing principles, and Section 5 proposes an efficient skycube algorithm. Section 6 reports our extensive evaluation results. Section 7 surveys related work. Finally, Section 8 concludes this paper.

2. PRELIMINARIES

We first introduce basic notations to address the skycube problem (Appendix A). Let $\mathcal{D}$ be a finite d-dimensional space, *i.e.*, $\mathcal{D}=\{D_1,\ \ldots,\ D_d\}$, where each dimension has a domain of non-negative rational number $\mathbb{Q}^+$, denoted as $dom(D_i) \rightarrow \mathbb{Q}^+$. Let $\mathcal{S}$ be a set of finite n points as a subset of $dom(\mathcal{D})$, *i.e.*, $\mathcal{S} \subset dom(\mathcal{D})$ such that $dom(\mathcal{D}) = dom(D_1) \times \ldots \times dom(D_d)$. A point p in $\mathcal{S}$ is represented by $p = (p_1, \ldots, p_d)$ such that $\forall i \in [1, d] : p_i \in dom(D_i)$.

Table 1: A skycube using a toy dataset

	D_1	D_2	D_3	D_4
a	7	1	8	4
b	5	7	1	2
c	4	4	2	1
d	1	6	3	3
e	2	5	4	6
f	3	2	6	8
g	5	3	2	6
h	4	5	7	2
i	6	4	5	7
j	6	6	2	6

Subspace	Skyline
D_1	d
D_2	a
D_3	b
D_4	c
D_1D_2	a, d, e, f
D_1D_3	b, c, d
D_1D_4	c, d
D_2D_3	a, b, f, g
D_2D_4	a, c
D_3D_4	b, c
$D_1D_2D_3$	a, b, c, d, e, f, g
$D_1D_2D_4$	a, c, d, e, f, g
$D_1D_3D_4$	b, c, d, g
$D_2D_3D_4$	a, b, c, f, g
$D_1D_2D_3D_4$	a, b, c, d, e, f, g

We then present common notions used in the skyline literature. Throughout this paper, we consistently use *min* operator for skyline queries. Given two points p and q, p *dominates* q on $\mathcal{D}$ if $\forall\ i \in [1, d] : p_i \leq q_i$ and $\exists\ j \in [1, d] : p_j < q_j$, denoted as $p \prec_{\mathcal{D}} q$. Also, p and q are *incomparable* on $\mathcal{D}$ if they do not dominate each other, denoted as $p \sim_{\mathcal{D}} q$. Given $\mathcal{S}$, a skyline query on $\mathcal{D}$ returns a set of points, or a *skyline*, that are not dominated by any other points in $\mathcal{S}$, *i.e.*, $\textsc{Sky}_{\mathcal{D}}(\mathcal{S}) = \{p \in \mathcal{S} | \nexists q \in \mathcal{S} : q \prec_{\mathcal{D}} p\}$. A point in the skyline is called a *skyline point*.

The notions on $\mathcal{D}$ can be intuitively extended to its subspace. Given a subspace $\mathcal{U} \subseteq \mathcal{D}$ ($\mathcal{U} \neq \emptyset$), p dominates q on projected $|\mathcal{U}|$-dimensional space if and only if $\forall\ i \in [1, |\mathcal{U}|] : p_{k_i} \leq q_{k_i}$ and $\exists\ j \in [1, |\mathcal{U}|] : p_{k_j} < q_{k_j}$. Using the dominance notion on $\mathcal{U}$, we define a *subspace skyline* [11, 14, 16] – given $\mathcal{U}$, a point p is a skyline point on projected $|\mathcal{U}|$-dimensional space if no other point q dominates p on $\mathcal{U}$.

Based on this notion, we introduce a *skycube* [11, 16], which consists of $2^d - 1$ subspace skylines on all possible non-empty subspaces of $\mathcal{D}$. We depict a skycube using a toy dataset (Table 1), where $\mathcal{D}$ is four-dimensional space, denoted as $D_1D_2D_3D_4$. For brevity, we represent $D_1D_2D_3D_4$ as D_{1234} from now on. Formally:

Definition 1 (Skycube) Given $\mathcal{S}$ on $\mathcal{D}$, a *skycube* is a set of all skylines in $2^d - 1$ nonempty subspaces, denoted as $\textsc{Skycube}(\mathcal{S}, \mathcal{D}) = \{(\mathcal{U}, \textsc{Sky}_{\mathcal{U}}(\mathcal{S})) | \mathcal{U} \subseteq \mathcal{D}, \mathcal{U} \neq \emptyset\}$, where $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$ is called a *cuboid* on subspace $\mathcal{U}$.

The skycube can be visualized as a lattice structure like a data cube. According to the size of subspace, we number the level of each cuboid from the bottom to the top. Given two subspaces $\mathcal{U}$ and $\mathcal{V}$, if $\mathcal{U} \subset \mathcal{V}$, $\textsc{Sky}_{\mathcal{V}}(\mathcal{S})$ is an *ancestor* of $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$, and $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$ is a *descendant* of $\textsc{Sky}_{\mathcal{V}}(\mathcal{S})$. In particular, if $\mathcal{U} \subset \mathcal{V}$ and $|\mathcal{V} - \mathcal{U}| = 1$, $\textsc{Sky}_{\mathcal{V}}(\mathcal{S})$ is a *parent* of $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$, and $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$ is a *child* of $\textsc{Sky}_{\mathcal{V}}(\mathcal{S})$.

Once the skycube is built up, users can retrieve multiple relevant skylines with a fast response time. Specifically, using analytical operations such as "roll-up" or "drill-down" over the skycube, users can shrink or extend subspace skylines to find out meaningful skyline results.

In spite of the usefulness of the skycube, the complexity of skycube computation is known to be NP-hard [11]. In particular, skyline or skycube computation depends heavily on CPU-intensive point-wise comparisons to check the dominance relationships between points, called *dominance tests*. To address the problem, we will discuss the opportunities to share computation in detail.

3. OBSERVATIONS

In this section, we first observe the limitations of existing skycube algorithms (Section 3.1). To overcome these limitations, we then exploit point-based space partitioning to share structure in finer granularity (Section 3.2).

From this point on, we "tentatively" assume *distinct value condition* as similarly assumed in the prior skyline literature [11, 16] for simpler presentation. Specifically, two points p and q in $\mathcal{S}$ are *distinct* on $\mathcal{D}$ if $\forall i \in [1,d] : p_i \neq q_i$. Based on this assumption, the containment relationship between cuboids holds, called *skyline monotonicity*, *i.e.*, if $\mathcal{U} \subset \mathcal{V}$, then $\textsc{Sky}_{\mathcal{U}}(\mathcal{S}) \subseteq \textsc{Sky}_{\mathcal{V}}(\mathcal{S})$ holds. We will later generalize our discussion beyond this assumption.

3.1 Limitations of Existing Approaches

We first observe the two state-of-the-art skycube algorithms and their limitations.

Bottom-**U**p **S**kycube algorithm (BUS): This algorithm serially computes cuboids by using the principle of *sharing result* in a bottom-up manner. That is, BUS exploits skyline monotonicity to bypass dominance tests for skylines on $\mathcal{U}$, when computing a skyline on $\mathcal{V}$ using SFS [2].

The limitation of this approach is, by sharing only the results, the dominance tests of non-skyline points cannot be reused. Specifically, if a point p is a non-skyline point on $\mathcal{U}$, BUS needs to perform dominance tests for p on $\mathcal{V}$ without reusing dominance tests on $\mathcal{U}$. In addition, BUS requires repeated access to an entire dataset in computing each cuboid, incurring additional overhead.

Top-**D**own **S**kycube algorithm (TDS): This algorithm is based on DC [1], and leverages both the principles of *sharing result* and *sharing structure* in a top-down manner.

Specifically, when computing $\textsc{Sky}_{\mathcal{V}}(\mathcal{S})$, TDS first employs a single parent $\textsc{Sky}_{\mathcal{V}'}(\mathcal{S})$ such that $\mathcal{V} \subset \mathcal{V}'$, instead of $\mathcal{S}$. TDS then amortizes the computation of monotonic subspaces *e.g.*, $\{D_{123}, D_{12}, D_1\}$, where an adjacent pair has a parent-child relationship as in $\mathcal{U}$ and $\mathcal{V}$. In this process, a two-dimensional partitioning structure originated from DC is employed – TDS evenly divides $\mathcal{U}$ into four subregions, and then merges local skylines on each subregion into a global skyline. Because partitioning-and-merging on $\mathcal{U}$ totally overlap with that on $\mathcal{V}$, some of local skylines on $\mathcal{U}$ can be skylines on either $\mathcal{U}$ or $\mathcal{V}$. TDS can thus reuse dominance tests on $\mathcal{U}$ when computing a skyline on $\mathcal{V}$.

While TDS computes multiple cuboids simultaneously, it still has more room for optimizing both principles. First, TDS solely deals with a single parent by neglecting the opportunity of sharing *multiple parents*. Second, TDS simply shares a two-dimensional partitioning structure, obtained as a byproduct of DC. This structure, by being restricted to partition a dataset into four subsets, cannot optimize dominance and incomparability relationships in high-dimensional data, which in turn restricts optimization margin of TDS.

To overcome these limitations, we deal with the principle of sharing result and sharing structure by leveraging *multiple parents* and a *skytree*, representing recursive *point-based space partitioning* respectively (Section 4).

3.2 Point-based Space Partitioning

This section investigates point-based space partitioning [6, 17] to further speed up skycube computation. Specifically, we consider a fine-granularity structure to share dominance tests on non-skyline points and incomparable points.

Given $\mathcal{S}$, a pivot point p^V on $\mathcal{D}$ partitions $\mathcal{S}$ into 2^d disjoint subregions in which all dominance tests between $\mathcal{S}$ and p^V can be summarized. In particular, the points with the same relationships can be subsumed in a subregion. Formally, let $\mathcal{R}$ denote a set of subregions such that $\mathcal{R} = \{R^0, \ldots R^{2^d-1}\}$.

To clarify region-wise relationships, we then introduce a d-dimensional binary vector. Let $\mathcal{B}$ denote a set of binary vectors, which are mapped into subregions such that $\forall B^i \in \mathcal{B} : B^i \rightarrow R^i$. Given p^V, we formally present the i^{th} dimension value of B with respect to a point q, denoted as B_i. Note that B_i is the i^{th} most significant bit in B.

$$B_i \leftarrow \begin{cases} 0, & \text{if } q_i < p_i^V; \\ 1, & \text{otherwise.} \end{cases}$$

To illustrate this, we describe three-dimensional partitioning. Suppose that $p^V = (p_1^V, \ldots, p_d^V)$ divides an entire space into eight subregions, which is associated with binary vectors, *i.e.*, $\mathcal{B} = \{000,\ 001,\ \ldots,\ 111\}$. If B_i is 0, the range of possible point values is $[0, p_i^V)$. Otherwise, the range is $[p_i^V, 1]$. Accordingly, 001 and 100 are associated with subregions $R^1 = [0, p_1^V) \times [0, p_2^V) \times [p_3^V, 1]$ and $R^4 = [p_1^V, 1] \times [0, p_2^V) \times [0, p_3^V)$, and the points mapped to 001 and 100 are located in R^1 and R^4.

Based on the binary vectors, we can derive region-wise relationships [6] in $\mathcal{B}$ as follows:

Definition 2 (Dominance in $\mathcal{B}$) Given two vectors B and B' , B *dominates* B' on $\mathcal{D}$, denoted as $B \prec_{\mathcal{D}} B'$, if and only if $\forall i \in [1,d] : B_i < B'_i$, *i.e.*, $\forall i \in [1,d] : B_i = 0$ and $B'_i = 1$.

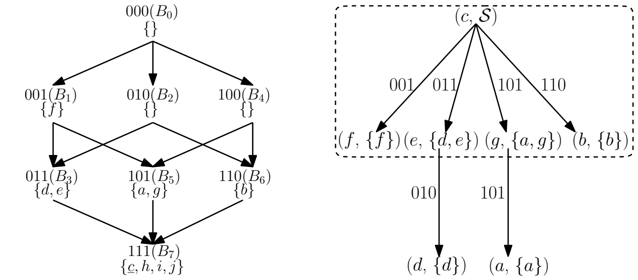
Definition 3 (Partial dominance in $\mathcal{B}$) Given two vectors B and B', B *partially dominates* B' on $\mathcal{D}$, denoted as $B \prec_{Par} B'$, if and only if $\forall i \in [1,d] : B_i \leq B'_i$ and $\exists j \in [1,d] : B_j = B'_j$.

Definition 4 (Incomparability in $\mathcal{B}$) Given two vectors B and B', B is *incomparable* with B' on $\mathcal{D}$, denoted as $B \sim_{\mathcal{D}} B'$, if and only if $\exists i \in [1,d] : B_i < B'_i$ and $\exists j \in [1,d] : B'_j < B_j$.

The relationships between binary vectors can be visualized as a *lattice* (Figure 2a). In this lattice, adjacent node pairs connected by an arrow have partial dominance relationships. By *transitivity*, node pairs reachable by one of multiple paths also have partial dominance relationships. As an exception, the dominance holds between the top and the bottom, *e.g.*, 000 dominates 111. Lastly, the incomparability holds for all non-reachable node pairs, *e.g.*, 001 and 010 are incomparable.

4. SHARING COMPUTATION

This section explores the opportunities for sharing computation across subspaces. As a starting point, we first introduce a *skytree*, representing recursive point-based space



(a) A lattice on D_{123} ($p^V = c$) (b) A skytree on D_{123}

Figure 2: A lattice and a skytree on D_{123}

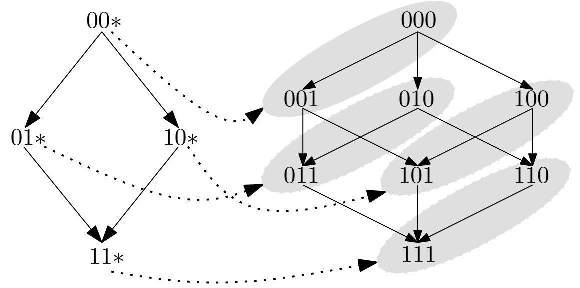


Figure 3: Sharing lattices between D_{12} and D_{123}

partitioning (Section 4.1). We then develop the principles of sharing structure and sharing result by leveraging a skytree (Section 4.2) and multiple parents (Section 4.3). Recall that we tentatively assume distinct value condition. We will later drop this assumption (Section 5.2).

4.1 SkyTree as an Input of Skycube Computation

The starting point for sharing structure is to run a skyline algorithm [6] using point-based space partitioning on full space $\mathcal{D}$. To illustrate, the algorithm may pick $p^V = c$ (underline) as the first pivot in a toy dataset $\mathcal{S}$ (Table 1) on D_{123}, based on which we generate a lattice that consists of eight binary vectors associated with subregions partitioned (Figure 2a). After pruning non-skyline points $\{h, i, j\}$ mapped to 111, we can again construct lattices by recursively partitioning $\{d, e\}$ and $\{a, g\}$ mapped to 011 and 101 in the same way, where these lattices are represented by a recursive *hierarchical* structure.

The recursive lattices can be visualized as a tree, called a *skytree*. Formally, a skytree on $\mathcal{D}$, denoted as $\mathcal{T}_{\mathcal{D}} = (N, I)$, is represented by a set N of nodes and a set I of links. Each node (p, S) in N consists of a pivot point p that partitions a subset S of points into child nodes on $\mathcal{D}$, and a link is associated with a binary vector given by a mapping function $m : I \to B$. Observe that the lattice in Figure 2(a) is equivalent to the first-level subtree marked within a dashed box in Figure 2(b). The second-level tree shows how $\{d, e\}$ and $\{a, g\}$ mapped to 011 and 101 are further partitioned with respect to e and g as pivot points respectively.

This skytree, obtained as a by-product of the point-based space partitioning skyline algorithm [6], is an input of our skycube computation. We then move on to fully exploit this skytree for the principle of sharing structure.

4.2 Principle of Sharing Structure by Leveraging the Skytree

We first present how to share point-wise dominance and incomparability. Given two *distinct* points p and q on $\mathcal{V}$, we formally state dominance and incomparability shared across multiple subspaces. The detailed proofs are described in Appendix B.

LEMMA 1 (SHARING DOMINANCE) Given two points p and q on $\mathcal{V}$, if $p \prec_{\mathcal{V}} q$ such that $\mathcal{U} \subset \mathcal{V}$, then $p \prec_{\mathcal{U}} q$ holds.

LEMMA 2 (SHARING INCOMPARABILITY) Given two points p and q on $\mathcal{U}$, if $p \sim_{\mathcal{U}} q$ such that $\mathcal{U} \subset \mathcal{V}$, then $p \sim_{\mathcal{V}} q$ holds.

These properties are essential for reusing point-wise relationships. Specifically, when computing $\text{SKY}_{\mathcal{V}}(\mathcal{S})$, we can safely prune out dominated points on $\mathcal{V}$, because they are non-skyline points on both $\mathcal{U}$ and $\mathcal{V}$. Also, for incomparable points on $\mathcal{U}$, we can bypass their dominance tests on $\mathcal{V}$.

We extend these properties for binary vectors in the lattice. First, when p^V partitions $\mathcal{S}$ on $\mathcal{V}$, any point mapped to a binary vector B such that $\forall d_i \in \mathcal{V} : B_i = 0$ dominates all points mapped to a binary vector B' such that $\forall d_i \in \mathcal{V} : B'_i = 1$ on $\mathcal{U}$ and $\mathcal{V}$. Formally, we state sharing dominance in $\mathcal{B}$ as follows:

THEOREM 1 (SHARING DOMINANCE IN $\mathcal{B}$) Given two vectors B and B' on $\mathcal{V}$, if $B \prec_{\mathcal{V}} B'$ such that $\mathcal{U} \subset \mathcal{V}$, then $B \prec_{\mathcal{U}} B'$ holds.

Second, we extend sharing point-wise incomparability for the lattice. Given two incomparable vectors B and B' on $\mathcal{U}$, their extended vectors are still incomparable regardless the values on $\mathcal{V} - \mathcal{U}$. We can thus bypass dominance tests between incomparable vectors on both $\mathcal{U}$ and $\mathcal{V}$. Formally, sharing incomparability in $\mathcal{B}$ is stated as follows:

THEOREM 2 (SHARING INCOMPARABILITY IN $\mathcal{B}$) Given two vectors B and B' on $\mathcal{U}$, if $B \sim_{\mathcal{U}} B'$ such that $\mathcal{U} \subset \mathcal{V}$, then $B \sim_{\mathcal{V}} B'$ holds.

Based on these properties, we explore the possibility of *sharing lattices* between $\mathcal{U}$ and $\mathcal{V}$. When a pivot point p^V divides two subspaces $\mathcal{U}$ and $\mathcal{V}$, a subregion on $\mathcal{U}$ covers finer $2^{|\mathcal{V}-\mathcal{U}|}$ subregions on $\mathcal{V}$. Accordingly, a vector B on $\mathcal{U}$ can be represented by $2^{|\mathcal{V}-\mathcal{U}|}$ vectors on $\mathcal{V}$. For brevity, we use a symbol $*$ to denote all possible binary values on $\mathcal{V}$. We formally state the principle of sharing lattices:

THEOREM 3 (SHARING LATTICES) Each binary vector B on $\mathcal{V}$ is shared with a vector B' on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$, $\forall d_i \in \mathcal{U} : B'_i = B_i$ and $\forall d_j \in \mathcal{V} - \mathcal{U} : B'_j = *$.

Figure 3 illustrates sharing lattices between $\mathcal{U} = D_{12}$ and $\mathcal{V} = D_{123}$. Each vector on D_{12} can be extended to two vectors on D_{123}, *e.g.*, $00* = \{000, 001\}$. Based on sharing lattices, we can prune out the points mapped to 111 because p^V dominates the points on both $\mathcal{U}$ and $\mathcal{V}$ (Theorem 1). Also, incomparable vectors $01*$ and $10*$ on D_{12} are also incomparable on D_{123} (Theorem 2).

We next compute possible *minimum subspaces* [15] of each binary vector based on sharing lattices (shown as a set of subspaces in Figure 4). Specifically, when binary vectors are projected to $0 * *$, the possible minimum subspaces are covered with all subspaces. Similarly, when projected binary vectors are $10*$, their minimum subspaces exclude the smallest subspace D_1 because the points mapped to $10*$ are

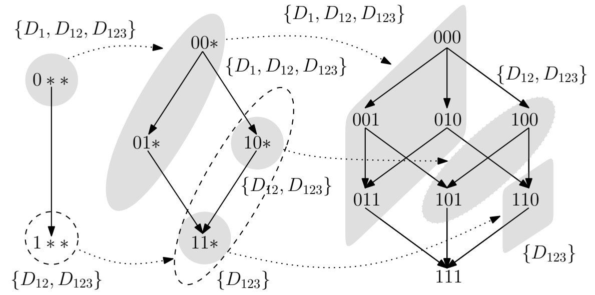
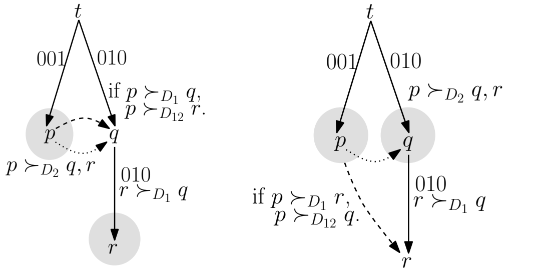


Figure 4: Possible minimum subspaces on D_{123}



(a) Inferring $p \succ_{D_{12}} r$ (b) Inferring $p \succ_{D_{12}} q$

Figure 5: Combining two relationships on $\mathcal{T}_{D_{123}}$

dominated by a pivot point on D_1. That is, based on the position of binary values with 0's, we can narrow down possible minimum subspaces.

We develop Algorithm SSkyTree, which takes a skytree $\mathcal{T}_{\mathcal{V}}$ as an input and reuses it to share partial dominance and incomparability between $\mathcal{U}$ and $\mathcal{V}$. While our baseline skyline algorithm [6] is only optimized to single skyline computation, SSkyTree focuses mainly on amortizing computation across multiple subspaces.

Specifically, SSkyTree makes use of $\mathcal{T}_{\mathcal{V}}$ with two main pillars. First, by leveraging partial dominance relationships between nodes in $\mathcal{T}_{\mathcal{V}}$, we eliminate non-skyline points on $\mathcal{U}$. Second, by leveraging spatial proximity inferred from binary vectors, we select a pivot point on $\mathcal{U}$, enabling us to leverage point-wise incomparability on $\mathcal{V}$ as much as possible. The pseudo-code of SSkyTree is described in Appendix C.

For the sake of representation, we only use a point p for a node (p, S) in the skytree from now on.

4.2.1 On Reusing Partial Dominance

We present a way of pruning non-skyline points on $\mathcal{U}$ from a skytree $\mathcal{T}_{\mathcal{V}}$. First, we can leverage a *vertical relationship*, representing two reachable nodes at different levels. For example, suppose there are two nodes c and b with a "parent-child" relationship in the skytree on D_{123} (Figure 2b). Considering the skyline on a subspace D_{12}, c can dominate b by projecting a binary vector $11*$ associated with two nodes on D_{12}. We formally generalize the parent-child relationship for all reachable nodes with vertical relationships:

Lemma 3 (Vertical relationship I) Given two nodes p and q in $\mathcal{T}_{\mathcal{V}}$, p dominates q on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$, if any link connected between p and q is associated with a binary vector B such that $\forall d_i \in \mathcal{U} : B_i = 1$.

In addition, we identify a parent node that is a non-skyline point on $\mathcal{U}$. Observe a binary vector between two nodes implies a "mutual" relationship. For example, a child node f dominates a parent node c on D_{12} by "flipping" $00*$ into $11*$ (Figure 2b). Formally:

Lemma 4 (Vertical relationship II) Given two nodes p and q in $\mathcal{T}_{\mathcal{V}}$, q dominates p on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$, if every link connected between p and q is associated with a binary vector B such that $\forall d_i \in \mathcal{U} : B_i = 0$.

Second, we consider a *horizontal* relationship on $\mathcal{T}_{\mathcal{V}}$, representing two nodes at the same level. Suppose there are two nodes f and b with a "sibling" relationship on the skytree $\mathcal{T}_{D_{123}}$ (Figure 2b). In this case, using vertical relationships for a common parent node c, we can infer f dominates c on D_{12} (Lemma 3), and c dominates b on D_{12} (Lemma 4). By transitivity, we can derive that f dominates b on D_{12}. For simplicity, we use a *pseudo-link*, which presents the dominance relationship between sibling nodes on D_{12}.

Lemma 5 (Horizontal relationship) Given two sibling nodes p and q with a common parent node r in $\mathcal{T}_{\mathcal{V}}$, p dominates q on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$, if a binary vector B between p and r and a binary vector B' between q and r are $\forall d_i \in \mathcal{U} : B_i = 0, B'_i = 1$.

We further investigate the skytree $\mathcal{T}_{\mathcal{V}}$ to reduce non-skyline points on $\mathcal{U}$ by combining the vertical relationship with the horizontal relationship. Suppose that three nodes p, q, and r, where p and q are sibling nodes, and q and r have a parent-child relationship on D_{123} (Figure 5a). Based on the horizontal relationship (dotted arrow), p dominates q and r on D_2 (Lemma 5). Also, q dominates r on D_1 by the vertical relationship (Lemma 3). In this case, if p dominates q on D_1 (dashed arrow), then p can also dominate r on D_{12} by transitivity (gray circle). That is, by performing dominance tests between sibling nodes on D_1, we can eliminate child nodes of a sibling node on D_{12}.

Lemma 6 (Combining relationships I) Given three nodes p, q and r in $\mathcal{T}_{\mathcal{V}}$, suppose p has a pseudo-link with q on $\mathcal{W}$ such that $\mathcal{W} \subset \mathcal{U}$, and q dominates r on $\mathcal{U} - \mathcal{W}$ by vertical relationship I. If p dominates q on $\mathcal{U} - \mathcal{W}$, then p dominates r on $\mathcal{U}$.

Also, we can prune a sibling node by combining two relationships. Given three nodes p, q and r on D_{123} (Figure 5b), we can infer that p dominates q and r on D_2 (Lemma 5) by a horizontal relationship (dotted arrow), and r dominates q on D_1 by a vertical relationship (Lemma 4). In this case, if p dominates r on D_1 (dashed arrow), then p can also dominate q on D_{12} (gray circle).

Lemma 7 (Combining relationships II) Given three nodes p, q and r in $\mathcal{T}_{\mathcal{V}}$, suppose p has a pseudo-link with q on $\mathcal{W}$ such that $\mathcal{W} \subset \mathcal{U}$, and r dominates q on $\mathcal{U} - \mathcal{W}$ by vertical relationship II. If p dominates r on $\mathcal{U} - \mathcal{W}$, then p dominates q on $\mathcal{U}$.

4.2.2 On Reusing Incomparability

We present a pivot point selection on $\mathcal{U}$ to share incomparability using a skytree $\mathcal{T}_{\mathcal{V}}$. When selecting a pivot point, it is critical to maximize both dominance and incomparability [6]. Based on the vertical relationship on $\mathcal{T}_{\mathcal{V}}$, observe that a child node p associated with B such that $\forall d_i \in \mathcal{U} : B.d_i = 0$ can be the most promising pivot point on $\mathcal{U}$ in that p cannot be dominated by points associated with the other binary vectors, and is close to an optimal pivot point by spatial proximity. More importantly, the point p can mostly share incomparability between $\mathcal{V}$ and $\mathcal{U}$. For example, when two skyline points on $\mathcal{U}$ are mapped to binary vectors $01*$ and $10*$ on D_{123}, a point p mapped to $00*$ can preserve the binary vectors mapped to points on D_{123}, which can be reused for sharing incomparability between points on D_{12} by simply projecting binary vectors.

We discuss how to find a pivot point on $\mathcal{V}$ using $\mathcal{T}_{\mathcal{V}}$. Specifically, we recursively traverse $\mathcal{T}_{\mathcal{V}}$ in depth-first order until finding a node that does not have a child node associated with a binary vector with 0's on $\mathcal{U}$. Using this pivot point, we can construct a lattice sharing incomparability on $\mathcal{U}$ by reusing binary vectors associated with points on $\mathcal{V}$.

4.3 Principle of Sharing Result by Leveraging Multiple Parents

We first present how to share a single parent. Specifically, when computing a skyline $\text{Sky}_{\mathcal{U}}(\mathcal{S})$ on $\mathcal{U}$ under distinct value condition, we can safely employ a pre-computed parent $\text{Sky}_{\mathcal{V}}(\mathcal{S})$ instead of $\mathcal{S}$. We formally state this property:

Lemma 8 Given a dataset $\mathcal{S}$ under distinct value condition, $\text{Sky}_{\mathcal{U}}(\mathcal{S}) = \text{Sky}_{\mathcal{U}}(\text{Sky}_{\mathcal{V}}(\mathcal{S}))$ holds on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$.

We then discuss how to extend this property for sharing multiple parents. Given multiple parents of $\text{Sky}_{\mathcal{U}}(\mathcal{S})$, each parent can be a candidate set to compute $\text{Sky}_{\mathcal{U}}(\mathcal{S})$ by skyline monotonicity. In this case, observe that a skyline point p in $\text{Sky}_{\mathcal{U}}(\mathcal{S})$ also has to exist in all its parents. This implies that p in $\text{Sky}_{\mathcal{U}}(\mathcal{S})$ has to be a point intersected with its parents. Formally:

Lemma 9 A point p cannot be subsumed to $\text{Sky}_{\mathcal{U}}(\mathcal{S})$ under distinct value condition, if $\exists \text{Sky}_{\mathcal{V}}(\mathcal{S}) : p \notin \text{Sky}_{\mathcal{V}}(\mathcal{S})$ such that $\mathcal{U} \subset \mathcal{V}$.

Theorem 4 (Sharing multiple parents) Given multiple parents of $\text{Sky}_{\mathcal{U}}(\mathcal{S})$ under distinct value condition, the intersection of multiple parents includes skyline on $\mathcal{U}$, *i.e.*, $\text{Sky}_{\mathcal{U}}(\mathcal{S}) \subseteq \bigcap_{\mathcal{U} \subset \mathcal{V}} \text{Sky}_{\mathcal{V}}(\mathcal{S})$.

The principle of sharing multiple parents can be combined with that of sharing structure using a skytree. Suppose that the skycube is computed in a top-down manner. In this case, when computing $\text{Sky}_{\mathcal{U}}(\mathcal{S})$, we can remove non-skyline points on $\mathcal{U}$ by reusing a skytree $\mathcal{T}_{\mathcal{V}}$. After that, we can additionally narrow a candidate set by intersecting remaining points for each parent. Using two principles, we can significantly reduce a candidate set for $\text{Sky}_{\mathcal{U}}(\mathcal{S})$.

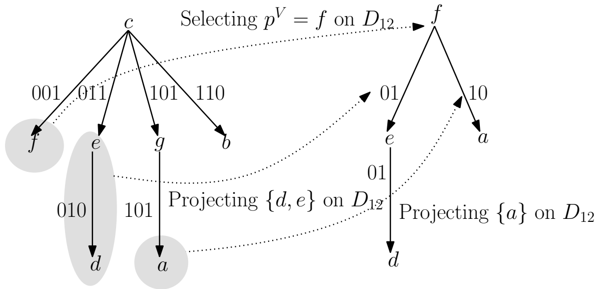
5. SKYCUBE COMPUTATION

This section first proposes an efficient skycube algorithm QSkycube which leverages two sharing computation principles (Section 5.1), and then discusses the modifications of QSkycube in general data case scenarios (Section 5.2).

5.1 Algorithm QSkycube

We develop Algorithm QSkycube, which takes advantage of two sharing computation principles. QSkycube constructs the skycube in a top-down fashion, where SSkyTree (Section 4) is employed to compute each cuboid. The detailed pseudo-code of QSkycube can be found in Appendix C.

Specifically, to compute a cuboid $\text{Sky}_{\mathcal{U}}(\mathcal{S})$, QSkycube first prunes non-skyline points by the sharing structure principle (Algorithm SSkyTree), and identifies the points intersected with multiple parents. QSkycube then computes a skyline on $\mathcal{U}$ using a baseline skyline algorithm [6].



c
Selecting $p^V = f$ on D_{12}
f
001
011
101
110
01
10
f
e
g
b
e
a
01
010
101
Projecting $\{d, e\}$ on D_{12}
Projecting $\{a\}$ on D_{12}
d
a
d

Figure 6: Illustration of QSkycube on $\{D_{123}, D_{12}, D_1\}$

We describe the overall procedure of QSkycube (Figure 6). For brevity, we provide the computation for $\{D_{123}, D_{12}, D_1\}$ with a single parent in a toy dataset (Table 1), which can be easily extended for multiple parents.

We first construct a skytree $\mathcal{T}_{D_{123}}$ on D_{123} (in the left). We can safely prune out non-skyline points $\{h, i, j\}$ on D_{123}, which do not appear in $\mathcal{T}_{D_{123}}$ (Theorem 1). After recursive partitioning in $\{d, e\}$ and $\{a, g\}$, we can identify skyline $\{a, b, c, d, e, f, g\}$ on D_{123} as node points on $\mathcal{T}_{D_{123}}$.

Using $\mathcal{T}_{D_{123}}$, we then select a pivot point and eliminate non-skyline points on D_{12}. Specifically, we select a point f associated with $00*$ as a pivot point on D_{12}, and prune points b and c by vertical relationships (Lemmas 3 and 4). Also, we can prune a point g that is dominated by a pivot point f on D_2 by performing a dominance test (Lemma 6). We lastly share the incomparability on $\mathcal{V}$ (Theorem 2) by projecting binary vectors on D_{12} (gray colors).

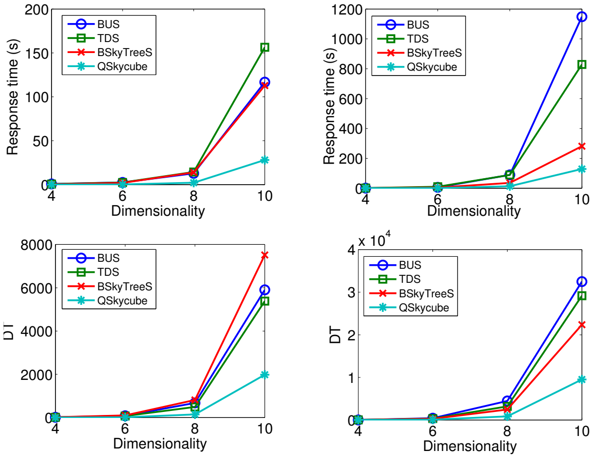
After computing our baseline algorithm [6] using the first-level lattice and a pivot point f on D_{12}, we can obtain a skytree $\mathcal{T}_{D_{12}}$ on D_{12} (in the right), and the skyline $\{a, d, e, f\}$ on D_{12}. By following the same process, we can select a point d as a pivot point on D_1, and the other points are pruned out. Finally, we can identify $\{d\}$ as a skyline on D_1.

5.2 Extending QSkycube for General Cases

This section discusses the modifications in QSkycube to handle a dataset without distinct value condition, referred to as a "general case" in the prior literature [11, 16]. Specifically, a point p such that $\forall i \in [1, |\mathcal{U}|] : p_{k_i} = q_{k_i}$ for a point $q \in \text{Sky}_{\mathcal{U}}(\mathcal{S})$ may exist. In this case, p can be a skyline point on $\mathcal{U}$, but p may be dominated by q on $\mathcal{V}$, implying p can be a skyline point only on $\mathcal{U}$, *i.e.*, skyline monotonicity no longer holds. As a result, q can be a "missing" skyline point in our proposed algorithm.

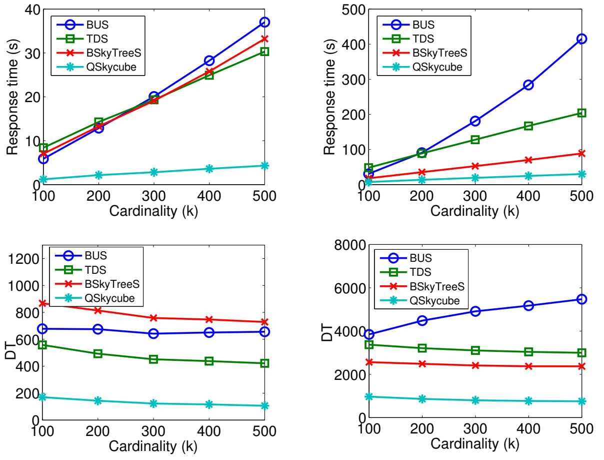
Though the existing idea [11, 16] using "pre-sorted lists" on every dimension can be adopted, it may incur high cost. We briefly explain this naive approach – when a pivot point p^V partitions $\mathcal{S}$ into $2^{|\mathcal{V}|}$ subsets on $\mathcal{V}$, we check if there are missing points for p^V by performing a binary search on the lists, and add missing points into a skyline on $\mathcal{V}$ if they exist. Based on this modification, we can guarantee the correctness of QSkycube in general cases. Because the cost of checking sorted lists increases linearly by the size of the skycube, however, it incurs prohibitive cost for high-dimensional data.

We thus propose to use an additional lattice structure, keeping points sharing the same values on each subspace, *i.e.*, points with *equivalence relationships*. Specifically, when



(a) IND (b) ANT

Figure 7: Varying d (n = 200k, best viewed in color)



(a) IND (b) ANT

Figure 8: Varying n ($d = 8$, best viewed in color)

constructing a skytree, we also keep an additional lattice storing missing points for each skyline point. Let $\mathcal{E}$ denote a set of binary vectors for equivalence relationships, *i.e.*, each vector E is mapped into the subspace with the same values. Formally, given p^V, we present the i^{th} dimension value of E with respect to a point q, denoted as E_i.

$$E_i \leftarrow \begin{cases} 0, & \text{if } \ q_i = p_i^V; \\ 1, & \text{otherwise.} \end{cases}$$

To illustrate this, we provide three-dimensional partitioning. When p^V partitions $\mathcal{V} = D_{123}$, a point q with the same values on D_{12} is mapped to 001 in the lattice. Also, other points with same values on D_{123} are mapped to 000. Recall that the lattice only stores points with equivalence relationships on any subspace, *i.e.*, no point is mapped to 111. To compute $\text{SKY}_{\mathcal{U}}(\mathcal{S})$, we then find missing points by retrieving the lattice for each skyline point on $\text{SKY}_{\mathcal{V}}(\mathcal{S})$.

We lastly discuss how to efficiently employ the lattice for equivalence relationships. First, because all skyline candidates are subsumed to $\text{SKY}_{\mathcal{D}}(\mathcal{S})$ and missing points for $\text{SKY}_{\mathcal{D}}(\mathcal{S})$, we construct a lattice for $\text{SKY}_{\mathcal{D}}(\mathcal{S})$ once, and reuse the lattice. Second, we can reduce missing points by exploiting the smallest minimum subspaces of skyline points. When the minimum subspace of a point p is $\mathcal{V}$, p and missing points for p can only be a skyline on $\mathcal{V}$, and cannot be a skyline on any subspace $\mathcal{U}$ of $\mathcal{V}$. Thus, we can safely ignore the points in computing skylines on $\mathcal{U}$.

6. EXPERIMENTS

This section presents empirical evaluation results for our proposed algorithm. We evaluate the scalability of our proposed algorithm QSkycube, by comparing it with the state-of-the-art algorithms BUS and TDS (Sections 6.1 and 6.2), and validate the efficiency of sharing multiple parents (Section 6.3) using extensive synthetic datasets. Detailed experimental settings are presented in Appendix D. We also evaluate QSkycube using real-life datasets (Appendix E).

6.1 Effect of Dimensionality

This section evaluates the effect of dimensionality by comparing four algorithms (Figure 7). We use independent and anti-correlated datasets with d from 4 to 10 and n = 200k.

In all settings, our proposed algorithm QSkycube is significantly faster than all the other algorithms. In independent datasets, as dimensionality increases, the performance gap between QSkycube and other algorithms increases exponentially. For instance, when $d = 10$, QSkycube is about four times faster than other algorithms. In anti-correlated datasets, as many points are skylines, sharing computation techniques used in QSkycube become less effective. As a result, the performance gap between QSkycube and BSkyTreeS is rather small.

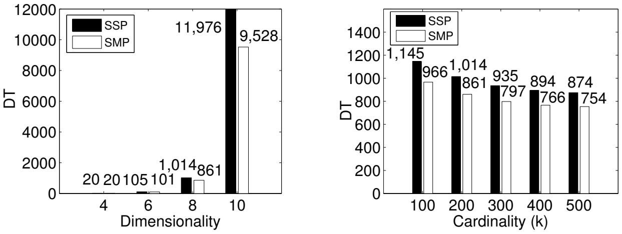
Another interesting observation is that BSkyTreeS, in spite of not sharing any computation between cuboids, outperforms existing skycube algorithms in anti-correlated datasets. This observation suggests the optimization margin coming from point-based space partitioning alone outweighs that from sharing computation across subspaces. This observation supports our decision to adopt point-based partitioning on top of sharing structure and result principles.

Meanwhile, depending on the performance measures used (dominance tests vs. response time), evaluation results can be different. Specifically, TDS shows the worst performance in $d = 10$ (Figure 7a), because TDS, though effective in reducing *dominance tests per point* (*DT*), incurs overhead for too many recursive calls. In addition, in terms of DT, BUS is more efficient than BSkyTreeS, while in terms of response time, BUS and BSkyTreeS show similar performances (Figure 7a). This can be explained by the fact that BUS requires a sequential scan for an entire dataset in computing each cuboid, which makes it relatively less efficient in terms of time. We make consistent observations from another experimental setting (Figure 8a).

6.2 Effect of Cardinality

This section evaluates the effect of cardinality (Figure 8), where independent and anti-correlated datasets with n from 100k to 500k and $d = 8$ are used. As cardinality increases, the performance gap between QSkycube and other algorithms increases quickly. Specifically, while the performance of QSkycube stays more or less constant, that of other algorithms increases linearly – when n = 500k, QSkycube is eight times as fast as other algorithms in independent datasets.

We also observe the difference between BUS and other algorithm increases, as the cardinality increases (Figure 8b). This can be explained by the additional data scan overhead of BUS, incurring cost proportional to cardinality.



(a) Varying d ($n = 200$k) (b) Varying n ($d = 8$)

Figure 9: Effect of sharing multiple parents in anti-correlated datasets

6.3 Effect of Sharing Multiple Parents

This section validates the effect of sharing multiple parents in anti-correlated datasets (Figure 9). We varied dimensionality and cardinality. For brevity, we use abbreviated notations SSP and SMP for "sharing a single parent" and "sharing multiple parents" respectively.

It is clear that SMP outperforms SSP in all datasets. In particular, as dimensionality increases, the difference of SMP and SSP increases rapidly. For instance, when $d = 10$, SMP saves about 20 percent more DT than SSP. This gap stays constant over all cardinality values, which empirically suggests SMP is a more effective strategy than SSP.

7. RELATED WORK

7.1 Skyline Computation

We first review existing efforts on skyline computation. In general, existing skyline algorithms can be classified as:

Sorting-based algorithms: This approach employs a sorted list to prune out non-skyline points as early as possible. An early block-nested-loop algorithm (BNL) [1] could be viewed as a sorting-based algorithm accessing points in stored order. For efficient non-skyline filtering, Tan el al. [13] proposed Index using multiple sorted lists, and Chomicki et al. [2] proposed SFS, where points are sorted in decreasing order of the size of *dominance regions*. Later, Godfrey et al. [3] developed LESS, which combines the advantages of both BNL and SFS.

Partitioning-based algorithms: This approach partitions a region into multiple subregions to exploit spatial proximity. An early algorithm DC [1] addressed skyline computation in a divide-and-conquer manner. Later, NN [5] and BBS [8] leveraged an *R-tree* to prune out non-skyline points efficiently. Recently, Lee et al. [7] proposed ZSearch using a *ZB-tree* as a new variant of a *B-tree*. More recently, *point-based space partitioning* [6, 17] was employed as an effective means of optimizing both dominance and incomparability in skyline computation. In particular, OSPS [17] and BSkyTree [6] are known to be the most efficient skyline algorithms without using pre-computed indices.

7.2 Skycube Computation

To address the curse of dimensionality, the skycube was recently introduced to construct an effective skyline structure. Specifically, to amortize the cost of computing multiple subspace skylines, state-of-the-art skycube algorithms BUS and TDS [11, 16] were proposed by optimizing SFS [2] and DC [1] respectively. To optimize the storage overhead of the skycube, a *compressed skycube* [15] was proposed as a "concise" alternative, removing duplicated skylines from the skycube and storing skylines only on *minimum subspaces*. Recently, Kailsam et al. [4] proposed a skycube algorithm using bitmap structures optimized for low-cardinality datasets.

As another skyline structure, Pei et al. [9, 10, 11] studied how to annotate each skyline with its *decisive subspace*, which is a minimal subspace ensuring *skyline monotonicity* for its superspaces. Similar to the skycube, they also proposed a skyline structure, called a *skyline group lattice*, using decisive subspaces. The structure can be shown as a way of identifying the semantics of skyline points. Recently, Raïssi et al. [12] proposed a skyline cube algorithm to optimize the skyline group lattice computation using formal concept analysis. Because the skycube computation could be extended to construct the skyline group lattice, this paper focused only on efficient skycube computation.

8. CONCLUSIONS

This paper studied efficient skycube computation using point-based space partitioning. To achieve this goal, we first identified more optimization opportunities from the principles of sharing structure and sharing result. We then leveraged a skytree and multiple parents for each principle, based on which we designed an efficient skycube algorithm. In experimental results, our proposed algorithm was significantly faster than existing algorithms.

9. REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[2] J. Chomicki, P. Godfery, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[3] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analysis for maximal vector computation. *VLDB Journal*, 16(1):5–28, 2007.

[4] G. T. Kailasam, J. Lee, J.-W. Rhee, and J. Kang. Efficient skycube computation using point and domain-based filtering. *Infomation Sciences*, 180(7):1090–1103, 2010.

[5] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[6] J. Lee and S. Hwang. BSkyTree: Scalable skyline computating using a balanced pivot selection. In *EDBT*, pages 195–206, 2010.

[7] K. C. K. Lee, W.-C. Lee, B. Zheng, H. Li, and Y. Tian. Z-SKY: an efficient skyline query processing framework based on Z-order. *VLDB Journal*, 19(3):333–362, 2010.

[8] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.

[9] J. Pei, A. W. Fu, X. Lin, and H. Wang. Computing compressed multidimensional skyline cubes efficiently. In *ICDE*, pages 96–105, 2007.

[10] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB*, pages 253–264, 2005.

[11] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu, and Q. Zhang. Towards multidimensional subspace skyline analysis. *ACM TODS*, 31(4):1335–1381, 2006.

[12] C. Raïssi, J. Pei, and T. Kister. Computing closed skycubes. *PVLDB*, 3(1):838–847, 2010.

[13] K. Tan, P. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.

[14] Y. Tao, X. Xiao, and J. Pei. SUBSKY: Efficient computation of skylines in subspaces. In *ICDE*, pages 65–74, 2006.

[15] T. Xia and D. Zhang. Refreshing the sky: The compressing skycube with efficient support for frequent updates. In *SIGMOD*, pages 491–502, 2006.

[16] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *VLDB*, pages 241–252, 2005.

[17] S. Zhang, N. Mamoulis, and D. W. Cheung. Scalable skyline computation using object-based space partitioning. In *SIGMOD*, pages 483–494, 2009.

APPENDIX

A. NOTATIONS

Table 2 summarizes the notations used in this paper.

Table 2: List of notations

Notation	Definition
$\mathcal{S}$	A dataset
n	Cardinality
$\mathcal{D}$	A full-space dimension set
$\mathcal{U}, \mathcal{V}$	A subspace of $\mathcal{D}$
d	Full-space dimensionality
d_i	A dimension ($1 \leq i \leq d$)
p, q	A point in $\mathcal{S}$, $p = (p_1, \ldots, p_d)$
$\prec_{\mathcal{D}}$	Dominance on $\mathcal{D}$
$\sim_{\mathcal{D}}$	Incomparability on $\mathcal{D}$
$\textsc{Sky}_{\mathcal{D}}(\mathcal{S})$	A skyline on $\mathcal{D}$ in $\mathcal{S}$
$\textsc{Skycube}(\mathcal{S}, \mathcal{D})$	A skycube on $\mathcal{D}$ in $\mathcal{S}$
p^V	A pivot point
$\mathcal{R}$	A set of subregions partitioned by p^V
$\mathcal{B}$	A set of binary vectors
B_i	The i^{th} dimension value of B
$\mathcal{T}_{\mathcal{V}}$	A skytree on a subspace $\mathcal{V}$
S	Partitioned point sets mapped to B
$\mathcal{E}$	A set of binary vectors for equivalence
E	A binary vector in $\mathcal{E}$

B. MATHEMATICAL PROOFS

This section proves the lemmas and theorems in this paper.

Lemma 1 (Sharing dominance) Given two points p and q on $\mathcal{V}$, if $p \prec_{\mathcal{V}} q$ such that $\mathcal{U} \subset \mathcal{V}$, then $p \prec_{\mathcal{U}} q$ holds.

Proof. Because $\mathcal{U} \subset \mathcal{V}$, if p dominates q on $\mathcal{V}$, then p also dominates q on $\mathcal{U}$. □

Lemma 2 (Sharing incomparability) Given two points p and q on $\mathcal{U}$, if $p \sim_{\mathcal{U}} q$ such that $\mathcal{U} \subset \mathcal{V}$, then $p \sim_{\mathcal{V}} q$ holds.

Proof. Because $\mathcal{U} \subset \mathcal{V}$, if p and q are incomparable on $\mathcal{U}$, they are also incomparable on $\mathcal{V}$ regardless of $\mathcal{V} - \mathcal{U}$. □

Theorem 1 (Sharing dominance in $\mathcal{B}$) Given two vectors B and B' on $\mathcal{V}$, if $B \prec_{\mathcal{V}} B'$ such that $\mathcal{U} \subset \mathcal{V}$, then $B \prec_{\mathcal{U}} B'$ holds.

Proof. Because the points mapped to $B^{2^{|\mathcal{V}|}-1}$ on $\mathcal{V}$ are dominated by p^V, the points are dominated by p^V on both $\mathcal{V}$ and $\mathcal{U}$ by Lemma 1. □

Theorem 2 (Sharing incomparability in $\mathcal{B}$) Given two vectors B and B' on $\mathcal{U}$, if $B \sim_{\mathcal{U}} B'$ such that $\mathcal{U} \subset \mathcal{V}$, then $B \sim_{\mathcal{V}} B'$ holds.

Proof. Because each point corresponding to incomparable vectors on $\mathcal{U}$ is incomparable, they are also incomparable on $\mathcal{V}$ by Lemma 2. □

Theorem 3 (Sharing lattices) Each binary vector B on $\mathcal{V}$ is shared with a vector B' on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$, $\forall d_i \in \mathcal{U} : B'_i = B_i$ and $\forall d_j \in \mathcal{V} - \mathcal{U} : B'_j = *$.

Proof. The proof is straightforward because a vector B on $\mathcal{V}$ shares the same values with a vector B' on $\mathcal{U}$. □

Lemma 3 (Vertical relationship I) Given two nodes p and q in $\mathcal{T}_{\mathcal{V}}$, p dominates q on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$, if any link connected between p and q is associated with a binary vector B such that $\forall d_i \in \mathcal{U} : B_i = 1$.

Proof. By Theorem 3, p^V dominates the points mapped to a vector B such that $\forall d_i \in \mathcal{U} : B.d_i = 1$ and $\forall d_j \in \mathcal{V} - \mathcal{U} : B.d_j = *$ on $\mathcal{U}$. As a result, the points are non-skyline points on $\mathcal{U}$. □

Lemma 4 (Vertical relationship II) Given two nodes p and q in $\mathcal{T}_{\mathcal{V}}$, q dominates p on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$, if every link connected between p and q is associated with a binary vector B such that $\forall d_i \in \mathcal{U} : B_i = 0$.

Proof. By Theorem 3, a point p mapped to a vector B such that $\forall d_i \in \mathcal{U} : B.d_i = 0$ and $\forall d_j \in \mathcal{V} - \mathcal{U} : B.d_j = *$ dominates a pivot point p^V on $\mathcal{U}$. As a result, p^V is a non-skyline point on $\mathcal{U}$. □

Lemma 5 (Horizontal relationship) Given two sibling nodes p and q with a common parent node r in $\mathcal{T}_{\mathcal{V}}$, p dominates q on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$, if a binary vector B between p and r and a binary vector B' between q and r are $\forall d_i \in \mathcal{U} : B_i = 0, B'_i = 1$.

Proof. By transitivity, we can prove that p dominates q on $\mathcal{U}$ using Lemmas 3 and 4. □

Lemma 6 (Combining relationships I) Given three nodes p, q and r in $\mathcal{T}_{\mathcal{V}}$, suppose p has a pseudo-link with q on $\mathcal{W}$ such that $\mathcal{W} \subset \mathcal{U}$, and q dominates r on $\mathcal{U} - \mathcal{W}$ by vertical relationship I. If p dominates q on $\mathcal{U} - \mathcal{W}$, then p dominates r on $\mathcal{U}$.

Proof. By transitivity, we can prove that p dominates r on $\mathcal{U}$ using Lemmas 3 and 5. □

Lemma 7 (Combining relationships II) Given three nodes p, q and r in $\mathcal{T}_{\mathcal{V}}$, suppose p has a pseudo-link with q on $\mathcal{W}$ such that $\mathcal{W} \subset \mathcal{U}$, and r dominates q on $\mathcal{U} - \mathcal{W}$ by vertical relationship II. If p dominates r on $\mathcal{U} - \mathcal{W}$, then p dominates q on $\mathcal{U}$.

Proof. By transitivity, we can prove that p dominates r on $\mathcal{U}$ using Lemmas 4 and 5. □

Lemma 8 Given a dataset $\mathcal{S}$ under distinct value condition, $\textsc{Sky}_{\mathcal{U}}(\mathcal{S}) = \textsc{Sky}_{\mathcal{U}}(\textsc{Sky}_{\mathcal{V}}(\mathcal{S}))$ holds on $\mathcal{U}$ such that $\mathcal{U} \subset \mathcal{V}$.

Proof. The proof is obvious because a non-skyline point on $\mathcal{V}$ cannot be skyline on $\mathcal{U}$ by Lemma 1. □

Lemma 9 A point p cannot be subsumed to $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$ under distinct value condition, if $\exists \textsc{Sky}_{\mathcal{V}}(\mathcal{S}) : p \notin \textsc{Sky}_{\mathcal{V}}(\mathcal{S})$ such that $\mathcal{U} \subset \mathcal{V}$.

Proof. Assume that a point p is subsumed to $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$. By skyline monotonicity, p has to be a skyline point on all superspaces of $\mathcal{U}$, which contradicts the fact that $\textsc{Sky}_{\mathcal{V}}(\mathcal{S})$ exists such that $p \notin \textsc{Sky}_{\mathcal{V}}(\mathcal{S})$. □

Theorem 4 (Sharing multiple parents) Given multiple parents of $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$ under distinct value condition, the intersection of multiple parents includes skyline on $\mathcal{U}$, *i.e.*, $\textsc{Sky}_{\mathcal{U}}(\mathcal{S}) \subseteq \bigcap_{\mathcal{U} \subset \mathcal{V}} \textsc{Sky}_{\mathcal{V}}(\mathcal{S})$.

Proof. By skyline monotonicity, a skyline point p on $\mathcal{U}$ has to be a skyline point on all superspaces of $\mathcal{U}$. p is thus subsumed by an intersected set of all parents of $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$. As a result, $\textsc{Sky}_{\mathcal{U}}(\mathcal{S}) \subseteq \bigcap_{\mathcal{U} \subset \mathcal{V}} \textsc{Sky}_{\mathcal{V}}(\mathcal{S})$ holds. □

C. DETAILED ALGORITHM

This section presents implementation details of our proposed algorithm. First, we describe the pseudo-code of Algorithm QSkycube (Algorithm 1).

Algorithm 1 QSkycube($\mathcal{S}$, $\mathcal{D}$)

```
Input: a dataset $\mathcal{S}$ and a full-space dimension set $\mathcal{D}$
Output: a skycube $\textsc{Skycube}(\mathcal{S}, \mathcal{D})$
 1: for all subspace $\mathcal{U}$ in a level-wise and top-down manner do
 2:    $\mathcal{C} \leftarrow \{\}$ // Initialize a skyline candidate set.
 3:    if $\mathcal{U}$ has any superspace $\mathcal{V}$ then
 4:       for all $\mathcal{V} \supset \mathcal{U}$ such that $|\mathcal{V} - \mathcal{U}| = 1$ do
 5:          $\mathcal{C} \leftarrow$ SSkyTree($\mathcal{T}_{\mathcal{V}}$, $\mathcal{U}$). // Algorithm 2
 6:          $\mathcal{L} \leftarrow \mathcal{L} \cap \mathcal{C}$. // Sharing multiple parents
 7:       end for
 8:    else
 9:       $\mathcal{U} \leftarrow \mathcal{D}$, $\mathcal{L} \leftarrow \mathcal{S}$. // When $\mathcal{U}$ is $\mathcal{D}$
10:    end if
11:    $\mathcal{T}_{\mathcal{U}} \leftarrow$ ComputeSkyline($\mathcal{L}$, $\mathcal{U}$). // Sharing dominance
12:    Insert $\textsc{Sky}_{\mathcal{U}}(\mathcal{D})$ by traversing $\mathcal{T}_{\mathcal{U}}$.
13: end for
14: return $\textsc{Skycube}(\mathcal{S}, \mathcal{D})$.
```

We discuss efficient implementation of QSkycube. As inputs, QSkycube takes an entire dataset $\mathcal{S}$ and full space $\mathcal{D}$. QSkycube computes each cuboid of $\textsc{Skycube}(\mathcal{S}, \mathcal{D})$ in a level-wise and top-down manner. Specifically, to compute a cuboid $\textsc{Sky}_{\mathcal{U}}(\mathcal{S})$, QSkycube consists of the following steps:

1. **Checking a superspace $\mathcal{V}$ of $\mathcal{U}$ (lines 3–10)**: By skyline monotonicity, we can exploit $\textsc{Sky}_{\mathcal{V}}(\mathcal{S})$ instead of $\mathcal{S}$. For efficient implementation, we can map all possible subspaces to numbers from 0 to $2^d - 2$ in sequential order. Using this mapping, we can efficiently check a superspace $\mathcal{V}$ of $\mathcal{U}$, such that $\mathcal{V} \supset \mathcal{U}$ and $|\mathcal{V} - \mathcal{U}| = 1$, using bit-wise operators. In addition, we can exploit a *topological* order that preserves level-wise and top-down traversal.
2. **Sharing computation (lines 5–6)**: Given a skytree $\mathcal{T}_{\mathcal{V}}$ on $\mathcal{V}$, we first prune out non-skyline points on $\mathcal{U}$ using Algorithm SSkyTree (Algorithm 2), which exploits the principle of *sharing structure* by leveraging $\mathcal{T}_{\mathcal{V}}$. We then intersect all skyline candidates to exploit the principle of *sharing result* by leveraging multiple parents.
3. **Computing the skyline on $\mathcal{U}$ (lines 11–12)**: After performing SSkyTree, we can obtain the first-level lattice on $\mathcal{U}$. Using the lattice, we can compute the skyline on $\mathcal{U}$ using a skyline algorithm [6] based on recursive point-based space partitioning, which generates a skytree $\mathcal{T}_{\mathcal{U}}$ as an output (Algorithm 5). In addition, we can safely prune out non-skyline points that do not appear in $\mathcal{T}_{\mathcal{U}}$ by sharing dominance (Theorem 1).

Next, we describe the pseudo-code of Algorithm SSkyTree (Algorithm 2). As inputs, SSkyTree takes a skytree $\mathcal{T}_{\mathcal{V}}$ and a subspace $\mathcal{U}$. Specifically, SSkyTree consists of the following steps:

1. **Selecting a pivot point**: we traverse a skytree $\mathcal{T}_{\mathcal{V}}$ in a depth-first order, and check whether a child node is associated with B such that $\forall d_i \in \mathcal{U} : B.d_i = 0$. After selecting a pivot point p^V, we make use of the pivot point not only to eliminate non-skyline points but also to organize the first-level lattice sharing incomparability from $\mathcal{T}_{\mathcal{V}}$.

Algorithm 2 SSkyTree($\mathcal{T}_{\mathcal{V}}$, $\mathcal{U}$)

```
$p^V \leftarrow$ SelectPivotPoint($\mathcal{T}_{\mathcal{V}}$, $\mathcal{U}$).
$\mathcal{C} \leftarrow \{p^V\} \cup$ Evaluate($\mathcal{T}_{\mathcal{V}}$, $\mathcal{U}$).
return $\mathcal{C}$.
```

2. **Evaluating skyline points on $\mathcal{V}$**: we traverse $\mathcal{T}_{\mathcal{V}}$ in *inorder*. In this process, we exclude non-skyline points on $\mathcal{U}$, by comparing p^V and other points based on both vertical and horizontal relationships (Lemmas 3–7). In addition, for the remaining points in $\mathcal{T}_{\mathcal{V}}$, we can simultaneously update binary vectors mapped to points with respect to p^V. We can thus infer the first-level lattice on $\mathcal{U}$ from the binary vectors, which enables us to share incomparability between $\mathcal{U}$ and $\mathcal{V}$ (Theorem 2).

We describe the detailed pseudo-code of each module used in SSkyTree. Let $\mathcal{T}_{\mathcal{V}}[q]$ denote a partial skytree of $\mathcal{T}_{\mathcal{V}}$ on which q is a root node.

Algorithm 3 SelectPivotPoint($\mathcal{T}_{\mathcal{V}}$, $\mathcal{U}$)

```
$p \leftarrow$ a root node of $\mathcal{T}_{\mathcal{V}}$.
if $p$ has any child node $q$ then
   if a link between $p$ and $q$ is $\forall d_i \in \mathcal{U} : B_i = 0$ then
      $p^V \leftarrow$ SelectPivotPoint($\mathcal{T}_{\mathcal{V}}[q]$, $\mathcal{U}$). // Recursive call.
   else
      $p^V \leftarrow p$. // When $q$ does not dominate $p$ on $\mathcal{U}$.
   end if
else
   $p^V \leftarrow q$. // When $p$ has no child node.
end if
return $p^V$
```

Algorithm 4 Evaluate(p^V, $\mathcal{T}_{\mathcal{V}}$, $\mathcal{U}$)

```
$\mathcal{C} \leftarrow \{\}$. // Initialize a skyline candidate set on $\mathcal{U}$.
$p \leftarrow$ a root node of $\mathcal{T}_{\mathcal{V}}$.
$S \leftarrow$ points at child nodes in $\mathcal{T}_{\mathcal{V}}$.
if $|S| > 0$ then
   Set a subspace $\mathcal{W}$ such that $\mathcal{W} \subset \mathcal{U}$ and $p^V \succ_{\mathcal{W}} p$.
   // Based on $\mathcal{W}$, traverse $\mathcal{T}_{\mathcal{V}}$ in inorder.
   for all $q \in S$ such that $q \succ_{\mathcal{U}-\mathcal{W}} p$ do
      $\mathcal{C} \leftarrow \mathcal{C} \cup$ Evaluate($p^V$, $\mathcal{T}_{\mathcal{V}}[q]$, $\mathcal{U}$).
      if a link between $p$ and $q$ is $\forall d_i \in \mathcal{U} : B_i = 0$ then
         Check $p$ as a non-skyline point on $\mathcal{U}$. // Lemma 4
      else if $p^V \succ_{\mathcal{W}} q$ then
         Check $p$ as a non-skyline point on $\mathcal{U}$. // Lemma 7
      end if
   end for
   if $p^V \not\succ_{\mathcal{U}} p$ or $p$ is still a skyline point on $\mathcal{U}$ then
      Update $p.B$ w.r.t $p^V$. // Sharing incomparability
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$.
   end if
   for all $q \in S$ such that $p \succ_{\mathcal{U}-\mathcal{W}} q$ do
      if a link between $p$ and $q$ is $\forall d_i \in \mathcal{U} : B_i = 1$ then
         Continue. // Lemma 3
      else if $p^V \succ_{\mathcal{U}-\mathcal{W}} p$ then
         Continue. // Lemma 6
      else
         $\mathcal{C} \leftarrow \mathcal{C} \cup$ Evaluate($p^V$, $\mathcal{T}_{\mathcal{V}}[q]$, $\mathcal{U}$).
      end if
   end for
else if $p^V \not\succ_{\mathcal{U}} p$ then
   Update $p.B$ w.r.t $p^V$. // Sharing incomparability
   $\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$. // When $p$ is a leaf node in $\mathcal{T}_{\mathcal{V}}$
end if
return $\mathcal{C}$
```

We discuss efficient implementation of modules used in SSkyTree. For efficient tree implementation, each node in $\mathcal{T}_{\mathcal{V}}$ stores both a point p and a binary vector B representing a relationship for its parent node. The binary vector B is updated by comparing p with a pivot point p^V, and used to organize the first-level lattice on $\mathcal{U}$. In particular, when performing the intersection between skyline candidates (line 6 in Algorithm 1), we can also update binary vectors for each point.

In addition, we can associate a binary vector B with the number from 0 to $2^{|\mathcal{U}|}-2$ on $\mathcal{U}$ similar to subspace mapping. Using this mapping, we can identify a subspace $\mathcal{W}$, such that $\mathcal{W} \subset \mathcal{V}$, on which we can efficiently check dominance relationships between p^V and other points using bit-wise operators. Also, we can make use of a topological order that preserves the partial dominance relationships between sibling nodes.

We lastly describe the pseudo-code of computing skyline on $\mathcal{U}$ (used in Algorithm 1) as below. Note that this computation is consistent with a skyline algorithm [6] using recursive point-based space partitioning, except that it generates a skytree on $\mathcal{U}$, which we use as an input of our skycube computation.

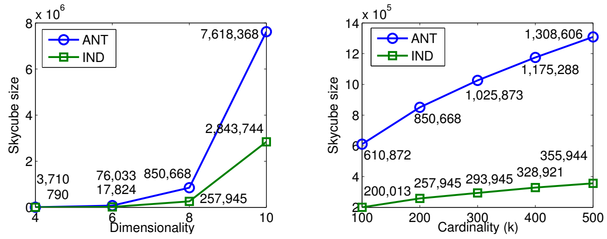
Algorithm 5 ComputeSkyline($\mathcal{S}, \mathcal{U}$)

```
$\mathcal{T}_{\mathcal{U}} \leftarrow \{\}$. // Initialize a skytree $\mathcal{T}_{\mathcal{U}}$ on $\mathcal{U}$.
$max \leftarrow 2^{|\mathcal{U}|} - 2$. // Set the size of a lattice.
$\mathcal{L}[1, max] \leftarrow \{\}$. // Initialize a lattice $\mathcal{L}$.
if a pivot point in $\mathcal{S}$ is decided then
    $p^V \leftarrow \mathcal{S}[1]$. // Initialize a pivot point $p^V$.
    Distribute $\mathcal{S}$ to a lattice $\mathcal{L}$.
else
    Select a balanced pivot point $p^V$ from $\mathcal{S}$ [6].
    for all $p \in \mathcal{S}$ do
        $i \leftarrow$ a binary vector mapped to $p$ w.r.t $p^V$.
        if $i \leq max$ then
            $\mathcal{L}[i] \leftarrow \mathcal{L}[i] \cup \{p\}$. // Add $p$ mapped to $B_i$ into $\mathcal{L}[i]$.
        end if
    end for
end if
$\mathcal{T}_{\mathcal{U}}$.add($p^V$). // Add $p^V$ into a current node in $\mathcal{T}_{\mathcal{U}}$.
for $i \leftarrow 1$ to $max$ do
    if $|\mathcal{L}[i]| > 0$ then
        for $\forall\ j \in [1, i]$: $B_j \prec_{Par} B_i$ do
            Remove non-skyline points from $\mathcal{L}[i]$ w.r.t $\mathcal{L}[j]$.
        end for
        if $|\mathcal{L}[i]| > 0$ then
            $\mathcal{T}' \leftarrow$ ComputeSkyline($\mathcal{L}[i]$). // Recursive call.
            $\mathcal{T}_{\mathcal{U}}$.add($\mathcal{T}'$). // Add a child node $\mathcal{T}'$ into $\mathcal{T}_{\mathcal{U}}$.
        end if
    end if
end for
return $\mathcal{T}_{\mathcal{U}}$
```

D. EXPERIMENTAL SETTINGS

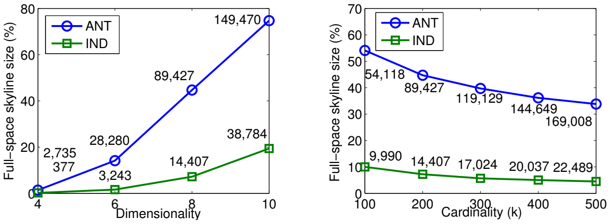
To validate our algorithm, we generate extensive synthetic datasets. Because real-life datasets limit the evaluation in various data environments, we employ the most popular benchmark datasets [1] with three parameters – distribution, cardinality, and dimensionality, which are widely adopted in the skyline literatures. Specifically:

- Distribution: We generate three datasets with different distributions, named as Correlated (COR), Independent (IND) and Anti-correlated (ANT). To generate each distribution, we follow the instructions in [1]. Note that we do not report the results for the correlated distribution in this paper, as the results are consistent with independent distribution.
- Dimensionality: We vary dimensionality d from 4 to 10. (Default: $d = 8$)
- Cardinality: We vary cardinality n from $100K$ to $500K$. (Default: n=200k)



(a) Varying d (n = 200k)　(b) Varying n (d = 8)

Figure 10: The size of the skycube



(a) Varying d (n = 200k)　(b) Varying n (d = 8)

Figure 11: The ratio of full-space skyline size

Figure 10 depicts the average size of skycubes in our synthetic datasets, which directly affects the overall performances of the algorithms. Figure 11 depicts the ratio of full-space skyline size $|\text{SKY}_{\mathcal{D}}(\mathcal{S})|$ to cardinality n in our synthetic datasets. The numbers (Figure 11) are $|\text{SKY}_{\mathcal{D}}(\mathcal{S})|$. The ratio of full-space skylines is used to infer the distribution of real-life datasets.

Next, we compared QSkycube with existing skycube algorithms BUS and TDS. Specifically, we implemented the skycube algorithms with some optimizations:

- BUS: We first sorted d dimensions in memory, and then performed BUS. Specifically, we selected the first dimension on $\mathcal{U}$ to compute $\text{SKY}_{\mathcal{U}}(\mathcal{S})$, and used a heuristic filter function [11, 16].
- TDS: We implemented TDS using the principles of sharing computation [11, 16]. In our implementation, we observed that too much recursive partitioning in small-size datasets deteriorates the overall performance of TDS, though the number of dominance tests is reduced. To prevent this phenomenon, we optimized TDS by combining it with a skyline algorithm. When the size of the partitioned dataset is less than a threshold δ, TDS executes a skyline algorithm SFS [2]. Based on our empirical optimization, we set δ as 10.
- BSkyTreeS: As a naive version of our proposed algorithm, we implemented BSkyTreeS, which serially computes each cuboid by our baseline algorithm BSkyTree [6].

- QSkycube: The proposed algorithm employed the principles of sharing structure and result by leveraging a skytree and sharing multiple parents. We select a pivot point using a pre-constructed skytree to preserve point-wise incomparability. As an exceptional case ($\mathcal{U} = \mathcal{D}$), we use a balanced pivot point selection proposed in [6]. Recall that the computation time of QSkycube includes this pivot point selection.

Due to the CPU-bound nature of skycube computation, we adopt CPU-cost measures, *response time* and *the number of dominance tests per point* (DT) while constructing the skycube, keeping all the points on the main memory. Specifically:

$$DT = \frac{\text{Total number of dominance tests}}{n}$$

All algorithms were implemented by C++ language. Also, all experiments were conducted using Windows 7 with an Intel Core2 Quad 2.33 GHz CPU and 4 GB main memory. All the values reported are the average of 10 runs.

E. EVALUATION IN REAL-LIFE DATA

This section evaluates the efficiency of QSkycube in real-life datasets. We collected "NBA" and "Household" datasets from *http://www.nba.com* and *http://www.ipums.org* respectively. NBA consists of 8-dimensional 17,264 points, representing a player's performance per year. Next, Household consists 6-dimensional 127,931 points, representing the ratio of the expenditure to an American family's annual income. These datasets are widely adopted to evaluate the efficiency of skyline queries [6, 17] and skycube queries [11].

We report on the response time and DT of all the skycube algorithms over these real-life datasets. Specifically, Table 3 depicts the response time (and DT in parentheses) for each skycube algorithm. We can observe that QSkycube (bold font) consistently outperforms all the other skycube algorithms in real-life datasets as well. However, unlike our observations from synthetic datasets, BUS and TDS outperforms BSkyTreeS in one or both real-life datasets. This can be explained by the ratio of $|\text{Sky}_{\mathcal{D}}(\mathcal{S})|$ (Household = 4.51% and NBA = 10.40%) and dimensionality, which suggests that real-life datasets are close to correlated and independent distribution in our synthetic datasets. This observation suggests that the optimization margin of sharing computation (used in TDS and QSkycube) is relatively more effective in such real-life settings.

Table 3: Comparisons with state-of-the-art algorithms in real-life datasets

Algorithms	Household	NBA
	$d = 6$, $n = 127{,}931$	$d = 8$, $n = 17{,}264$
	$\lvert\text{Sky}_{\mathcal{D}}(\mathcal{S})\rvert = 5{,}774$	$\lvert\text{Sky}_{\mathcal{D}}(\mathcal{S})\rvert = 1{,}796$
BUS	1.725 ($DT = 148$)	1.129 ($DT = 2103$)
TDS	1.293 ($DT = 65$)	0.758 ($DT = 312$)
BSkyTreeS	1.519 ($DT = 157$)	2.288 ($DT = 2967$)
QSkycube	**0.276 ($DT = 31$)**	**0.231 ($DT = 205$)**

Acknowledgements

This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology (MEST) / National Research Foundation of Korea (NRF) (Grant 2010-0001728) and Microsoft Research Asia (MSRA).