BSkyTree: Scalable Skyline Computation Using A Balanced Pivot Selection *

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

ABSTRACT

Skyline queries have gained a lot of attention for multi-criteria analysis in large-scale datasets. While existing skyline algorithms have focused mostly on exploiting data *dominance* to achieve efficiency, we propose that data *incomparability* should be treated as another key factor in optimizing skyline computation. Specifically, to optimize both factors, we first identify common modules shared by existing non-index skyline algorithms, and then analyze them to develop a cost model to guide a *balanced pivot point selection*. Based on the cost model, we lastly implement our balanced pivot selection in two algorithms, BSkyTree-S and BSkyTree-P, treating both dominance and incomparability as key factors. Our experimental results demonstrate that proposed algorithms outperform state-of-the-art skyline algorithms up to two orders of magnitude.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

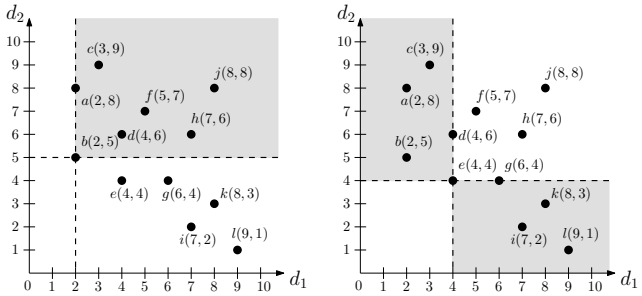
Algorithms, Experimentation, Performance

Keywords

skyline, dominance, incomparability, pivot selection

1. INTRODUCTION

Skyline queries have gained a lot of attention for multi-criteria decision making in large-scale datasets. Skyline query formulation is well suited for capturing intuitive user preferences using *dominance*– if a point p is better than another point q in at least one dimension and not worse than q in all other dimensions, it is said that p *dominates* q. Given a multi-dimensional dataset, the skyline query thus returns a subset of "interesting" points that are not dominated by any other points.



(a) a dominance region (b) incomparable regions

Figure 1: A toy dataset in two dimensional space

To illustrate skyline queries, we describe an example using an SQL-style expression, *i.e.*, `SKYLINE OF` [4, 5].

Example 1 (Skyline queries) Consider a hotel retrieval system using a database called Hotel(*hno*, *name*, *price*, *distance*, *city*). To find cheap hotels close to the lake on 'Lausanne', a user could formulate a skyline query as follows:

```
SELECT * FROM Hotel
WHERE city = 'Lausanne'
SKYLINE OF price MIN, distance MIN;          (Q1)
```

We present how existing algorithms typically compute skyline with a toy dataset as shown in Figure 1. As indicated by the MIN keyword in the query, the user prefers hotels with lower *price* and *distance* values. We can thus remove the hotels that have a higher *price* and *distance* compared to other hotels. For instance, given a "pivot" point b in Figure 1(a), six points in its "dominance region" (the shadowed rectangle) can be pruned out. The pruning then continues until the dataset converges to a subset of "not dominated" points $\{b, e, i, l\}$.

Existing algorithms, though they vary on how to choose and exploit pivot points, share a key structure as shown in Figure 2. Specifically, existing algorithms can be categorized into the following two categories: (1) *sorting-based* algorithms that focus on optimizing the pivot point ordering to prune more dominated points early on and (2) *partitioning-based* algorithms that focus on dividing an entire region into several subregions to enable "region-level" optimization.

*This work was supported by Microsoft Research Asia and Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology (MEST) / Korea Science and Engineering Foundation (KOSEF), grant number R11-2008-007- 03003-0.

EDBT 2010, March 22–26, 2010, Lausanne, Switzerland.

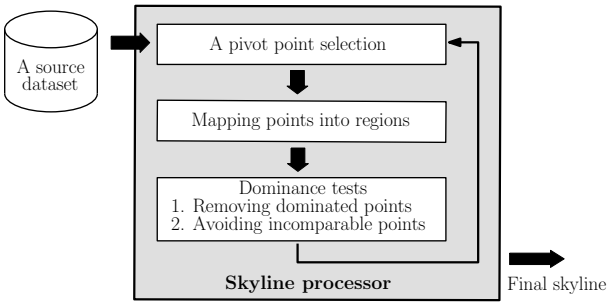


Figure 2: A generalized framework for skyline query processing

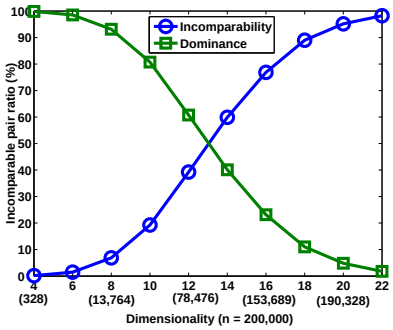


Figure 3: The effect of incomparability

The primary goal of sorting-based algorithms [6, 7, 8, 1] is to optimize pivot ordering so that a "desirable pivot" that prunes out most of the non-skyline points early on is accessed first. Specifically, to enable this ordering, existing algorithms attempt to access the pivot points that maximize dominance regions first.

Meanwhile, that of partitioning-based algorithms [9, 14, 11, 16] is to group points into subregions that share a value proximity. For instance, if spatial indices like an *R-tree* exist a priori, points are naturally aggregated into a minimum-bounding rectangle (MBR) as a subregion. With this aggregation, the points in an MBR can be pruned at once, by only using single dominance comparison [9, 14]. Recently, non-index partitioning algorithms [11, 16] also make use of region-level access to reduce "dominance tests" by using object-based region partitioning.

Based on this overview, existing algorithms can be summarized into the taxonomy shown in Table 1, from which we make the following two key observations:

Incomparability is critical to achieve scalability. While existing skyline algorithms have focused mostly on reducing point-wise dominance tests, *incomparability*, *i.e.*, two points p and q do not dominate each other, is another key factor in optimizing skyline computation. Especially, in high-dimensional space, most point pairs become incomparable. To illustrate this, Figure 3 depicts the number of point pairs with dominance and incomparability. The dataset was synthetically generated with 200,000 uniformly distributed points with independent attributes, and was used as a synthetic dataset. The numbers in parentheses indicate the average number of skyline points. It is clear that, in low dimensionality, almost all of the point pairs have dominance relations. In contrast, for a dimensionality higher than 13, incomparable pairs start to dominate. The graph empirically demonstrates that, to enable skyline query processing scalable over dimensionality, considering both the dominance and incomparability is crucial.

Balancing dominance and incomparability is non-trivial. To optimize both dominance and incomparability, the pivot point selection should be carefully designed for a balanced optimization of these two factors. Existing work mostly optimized for dominance by choosing a pivot point that maximized the dominance region as depicted in Figure 1(a). In clear contrast, an alternative extreme is to solely optimize for incomparability, by picking a pivot point e that "evenly" partitions the entire region as shown in Figure 1(b), in order to maximize incomparable subregions (marked by shaded rectangles). With this alternative pivot selection, even though the dominance region is reduced, we can still bypass nine dominance tests on point pairs across the two incomparable regions, $\{a, b, c\}$ and $\{i, k, l\}$, as they are guaranteed to be incomparable. However, it is non-trivial to find a *cost-optimal* strategy between these two extremes.

This paper thus aims to find the cost-optimal strategy for a systematic pivot point selection, considering both dominance and incomparability in "non-index" skyline computation without pre-computed structures, *e.g.*, *B-tree* or *R-tree*. To achieve this goal, we first identify common modules, *pivot point selection* and *pruning*, in existing skyline algorithms, and then derive a cost model of an optimal implementation Opt for pruning. This cost model guides our balancing act to pick a cost-optimal pivot point, by considering both dominance and incomparability as key optimization factors. We implement BSkyTree-S and BSkyTree-P, using our systematic pivot point selection. Our evaluation demonstrates that these two proposed algorithms outperform state-of-the-art algorithms up to two orders of magnitude.

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

- We develop a conceptual lattice structure illuminating region-level dominance and incomparability relations.
- We build an optimal pruning skeleton Opt of non-index skyline algorithms by using the lattice structure, and analyze a cost model of Opt.
- We design a systematic pivot point selection driven by the cost model, which optimizes both dominance and incomparability as key optimization factors.
- We implement two instances of Opt, BSkyTree-S and BSkyTree-P, by leveraging our proposed pivot point selection.
- We evaluate our proposed algorithms by comparing them with state-of-the-art algorithms for both real-life and extensive synthetic datasets.

The rest of this paper is organized as follows. Section 2 reviews existing skyline algorithms, and Section 3 presents preliminaries on skyline query processing. Section 4 introduces a lattice structure for mapping points into subregions, and presents a common skeleton Opt for pruning and its cost analysis. Section 5 designs a systematic pivot selection guided by the cost model of Opt. Section 6 then proposes two instances of Opt by implementing our pivot selection. Section 7 reports on experimental results for both real-life and synthetic datasets. Finally, Section 8 concludes our work.

2. RELATED WORK

Skyline queries have been first studied as *maximal vectors* [10, 2, 3] in the context of theoretical analysis. Later, pioneered by [4], many skyline algorithms have been proposed in the database community. Although skyline queries are extensively used in various settings, *i.e.*, distributed, dynamic, and probabilistic data, we focus only on existing work for exact skyline computation in static data in centralized environments. Specifically, this can be categorized into two categories, sorting- and partitioning-based algorithms.

2.1 Sorting-based Algorithms

Sorting-based algorithms aim to optimize pivot ordering to prune out non-skyline points early on. An early block-nested-loop algorithm [4] could also be viewed as a sorting-based algorithm accessing points in stored order. Chomicki et al. [6, 7] proposed SFS sorting points in decreasing order of the size of dominance regions, using a monotone scoring function. Later, Godfrey et al. [8] developed LESS as a combination of both BNL and SFS, which accesses points in stored order (as in BNL) but keeping skyline candidates in sorted order of dominance regions (as in SFS) to reduce the dominance tests for non-skyline points. Recently, Bartolini et al. [1] enhanced SFS by using *minC* function to achieve early termination in distributed data scenarios. More recently, Park et al. [15] proposed SSkyline that dynamically changes pivot points in stored order if dominated.

Drawbacks: These algorithms focus solely on optimizing data dominance, and neglect data incomparability factor in skyline query computation. Recall that, the number of incomparable pairs dominate in high-dimensional space, as depicted in Figure 3.

2.2 Partitioning-based Algorithms

Partitioning-based algorithms aim to group points into subregions that share commonality to carry out region-based dominance tests. An early algorithm, D&C [4] simply divided the problem into multiple sub-problems, and merged the local skyline points into global skyline. To use a more efficient region-level access, namely NN [9] and BBS [14] built upon pre-constructed spatial indices like the R-tree. Recently, Lee et al. [12] proposed ZSearch using *ZB-tree* as a new variant of *B-tree*, and Morse et al. [13] proposed LS using a static lattice structure for the special case of low-cardinality datasets.

In contrast, we handle non-index skyline algorithms which partition at the run time. Specifically, Zhang et al. [16] and Lee et al. [11] proposed partitioning-based algorithms without pre-computed indices. These algorithms outperformed sorting-based algorithms, by considering both dominance and incomparability.

Drawbacks: While existing partitioning-based algorithms explored the potential of optimizing for incomparability, they did not fully address its potential by using heuristic pivot selections, *i.e.*, dominance-based [11] and random [16] pivot selection, as we will empirically validate in Section 7.

2.3 Our Work

In this paper, we first identify common modules for sorting- and partitioning-based non-index algorithms, and then use them to analyze a cost model. Specifically, we use the model to balance both dominance and incomparability in pivot point selection. This systematic pivot point selection enables us to enhance all the algorithms in the taxonomy shown in Table 1, by exploring untapped room for further optimization.

	Sorting-based	Partitioning-based
Dominance	BNL [4], SFS [6, 7], LESS [8], SaLSa [1], SSkyline [15]	D&C [4], NN [9], BBS [14], LS [13], ZSearch [12]
Incomparability		OSPS [16], SkyTree [11]

Table 1: Taxonomy of existing skyline algorithms

3. PRELIMINARIES

We first introduce some basic notations to formally present skyline queries. Let $\mathcal{D}$ be a finite d-dimensional space, *i.e.*, $\{d_1, \ldots, d_d\}$, where each dimension has a domain of non-negative real number $\mathbb{R}^+$, denoted as $dom(d_i) \rightarrow \mathbb{R}^+$. Let $\mathcal{S}$ be a set of finite n points that is a subset of $dom(\mathcal{D})$, *i.e.*, $\mathcal{S} \subseteq dom(\mathcal{D})$. A point p in $\mathcal{S}$ is represented as $(p_1, \ldots, p_d)$ in which $\forall i \in [1, d] : p_i \in dom(d_i)$. For simplicity, $dom(d_i)$ has normalized values of [0,1].

Based on these notations, we formally define *dominance*, *incomparability*, and *skyline* respectively. These definitions are consistent with existing skyline work. Throughout this paper, we consistently use *min* operator for skyline computation.

Definition 1 (Dominance) Given $p, q \in \mathcal{S}$, p *dominates* q on $\mathcal{D}$, denoted as $p \prec_{\mathcal{D}} q$, if and only if $\forall\ i \in [1, d] : p_i \leq q_i$ and $\exists\ j \in [1, d] : p_j < q_j$.

Definition 2 (Incomparability) Given $p, q \in \mathcal{S}$, p and q are *incomparable* on $\mathcal{D}$, denoted as $p \sim_{\mathcal{D}} q$ if and only if $p \not\prec_{\mathcal{D}} q$ and $q \not\prec_{\mathcal{D}} p$.

Definition 3 (Skyline) A point p is a *skyline point* on $\mathcal{D}$ if and only if any other point q ($\neq p$) does not dominate p on $\mathcal{D}$. Given dataset $\mathcal{S}$ on $\mathcal{D}$, *skyline* is a set of skyline points such that $\text{SKY}_{\mathcal{D}}(\mathcal{S}) = \{p \in \mathcal{S} | \nexists q \in \mathcal{S} : q \prec_{\mathcal{D}} p\}$.

The skyline computation depends heavily on point-wise dominance tests. Given $\mathcal{S}$, a naive skyline algorithm might perform exhaustive pair-wise dominance tests on $n(n-1)/2$ pairs and incur quadratic costs. In other words, each point would be compared by up to $n-1$ other points. Ideally, an efficient algorithm would reduce the comparisons of non-skyline points down closely to 1, by scheduling dominance tests in such a way that a non-skyline point is only compared with a skyline point dominating the non-skyline point.

To make progress towards the goal, we extend point-level notions to "region-level" notions. Suppose that a hyper-rectangle region R on $\mathcal{D}$ is represented as $[u_1, v_1) \times \ldots \times [u_d, v_d)$ in which a virtual best point and a virtual worst point are denoted as $R_b = (u_1, \ldots, u_d)$ and $R_w = (v_1, \ldots, v_d)$ respectively. For this regional unit, we formally present the notions of dominance and incomparability, as done in [12].

Definition 4 (Region-level dominance) Given two regions R and R' on $\mathcal{D}$, R *dominates* R' if $R_w \prec_{\mathcal{D}} R'_b$.

Definition 5 (Region-level incomparability) Given two regions R and R' on $\mathcal{D}$, they are *incomparable* if $R_b \not\prec_{\mathcal{D}} R'_w$ and $R'_b \not\prec_{\mathcal{D}} R_w$.

For brevity of representation, we replace notations $\prec_{\mathcal{D}}$, $\not\prec_{\mathcal{D}}$, $\sim_{\mathcal{D}}$, and $\text{SKY}_{\mathcal{D}}(\mathcal{S})$ with $\prec$, $\not\prec$, $\sim$, and $\text{SKY}(\mathcal{S})$ if it is clear from the context.

We now explain how this region-level extension can be used to reducing dominance tests.

First, region-level dominance is used for reducing dominance tests of non-skyline points. If R dominates R', a point p in R dominates all points in R' after only one dominance test. This property intuitively inspires the idea of index-based algorithms [9, 14] by pruning a group of points in R', *i.e.*, an MBR, by a single dominance test between p and R'_b. Similar intuitions are used in non-index algorithms [6, 7, 8, 1] to access pivot point maximizing dominance regions first. These points would be effective in pruning out points in its dominance region early on.

Second, region-level incomparability is used to avoid dominance tests between incomparable points (not necessarily skyline points). Suppose that two regions R and R' are incomparable. In this case, the two points corresponding to each region are also incomparable, *i.e.*, if $R \sim R'$, then $\forall p \in R, \forall q \in R' : p \sim q$. We can thus save computation cost by bypassing dominance tests for point pairs between incomparable regions. This property inspires the idea of partitioning-based algorithms [16, 11], using heuristic pivot selection for incomparability. Recall that, optimizing for incomparability becomes more and more critical in high-dimensional space, where most point pairs become incomparable.

4. A COST MODEL OF SKYLINE ALGORITHMS

This section first introduces a conceptual structure illuminating the dominance and incomparability relations between partitioned subregions. Based on this structure, we then develop a cost model for non-index skyline algorithms, by measuring the number of dominance tests that affects the overall performance of skyline computation.

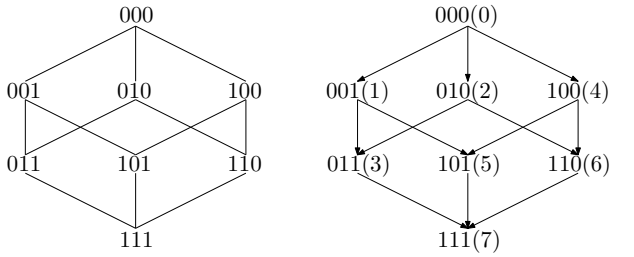
4.1 A Lattice Mapping Points into Regions

Given a pivot point p^V, the entire region can be divided into disjoint 2^d subregions. For instance, Figure 1(b) describes four subregions partitioned by a point e. Every point in $\mathcal{S}$ is thus contained in a subregion, *e.g.*, $\{\}$, $\{a, b, c\}$, $\{i, k, l\}$ and $\{d, e, f, g, h, j\}$. Formally, let $\mathcal{R}$ denote a set of subregions on $\mathcal{D}$, *i.e.*, $\mathcal{R} = \{R_0, \ldots R_{2^d-1}\}$, based on which we represent region-level relations.

More specifically, to simplify region-level relations, we introduce a d-dimensional vector B that corresponds to R. Let $\mathcal{B}$ denote a set of all d-dimensional vectors *i.e.*, $\mathcal{B} = \{B_0, \ldots, B_{2^d-1}\}$, where each vector is mapped into a subregion such that $\forall B_i \in \mathcal{B} : B_i \rightarrow R_i$. Let $B.d_i$ denote a value on d_i of vector B, where a value d_i on B corresponds to the i^{th} most significant bit. Formally, given a pivot point p^V and a point q in R_i, $B.d_i$ is represented as a binary value:

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

We then explain how to infer region-level relations from binary vectors presenting subregions. For instance, when d = 3, p^V divides the entire region into eight subregions, *i.e.*, $\mathcal{B} = \{B_0 = 000, B_1 = 001, \ldots, B_7 = 111\}$. If $B.d_i$ is 0, the range of possible point values is $[0, p_i^V)$. Otherwise, the range is $[p_i^V, 1]$. Thus, vectors B_0 and B_1 describe subregions $[0, p_1^V) \times [0, p_2^V) \times [0, p_3^V)$ and $[0, p_1^V) \times [0, p_2^V) \times [p_3^V, 1]$ respectively. The binary vectors are thus suited for presenting subregions concisely.



(a) A binary lattice (b) An encoded binary lattice

Figure 4: A lattice for mapping points into regions when $d = 3$

Using binary vectors, region-level relations can be formally represented as the following three cases:

Definition 6 (Dominance in $\mathcal{B}$) Given two vectors B and B', B *dominates* B' on $\mathcal{D}$, denoted as $B \prec B'$, if and only if $\forall i \in [1, d] : B.d_i < B'.d_i$, *i.e.*, $\forall i \in [1, d] : B.d_i = 0$ and $B'.d_i = 1$.

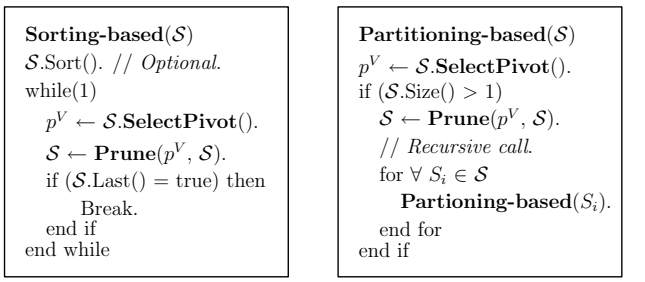
Definition 7 (Partial dominance in $\mathcal{B}$) Given two vectors B and B', B *partially dominates* B' on $\mathcal{D}$, denoted as $B \prec_{Par} B'$, if and only if $\forall i \in [1, d] : B.d_i \leq B'.d_i$ and $\exists j \in [1, d] : B.d_j = B'.d_j$.

Definition 8 (Incomparability in $\mathcal{B}$) Given two vectors B and B', B is *incomparable* with B' on $\mathcal{D}$, denoted as $B \sim B'$, if and only if $\exists i \in [1, d] : B.d_i < B'.d_i$ and $\exists j \in [1, d] : B'.d_j < B.d_j$.

The relations between binary vectors can thus be organized as a *partially ordered set*, represented as a *lattice*. To illustrate this, Figure 4 describes a binary lattice and its binary encoding when $d = 3$. In this lattice, adjacent node pairs connected by an arrow represent partial dominance relations. By the *transitivity*, node pairs reachable by a path of multiple paths also have partial dominance relations. Among these partially dominated pairs, (B_0, B_{2^d-1}) shows a dominance relation according to Definition 6. All of the remaining non-reachable pairs have incomparability relations.

More specifically, for all relations such that $\forall i, j \in [0, 2^d - 1] : (B_i, B_j) \in \mathcal{B} \times \mathcal{B}$, we explain how region-level relations are related to point-wise dominance tests. For the sake of representation, let S_i denote a subset of points in R_i mapped into corresponding vector B_i.

- **Dominance**: For dominance pair (B_0, B_{2^d-1}), if a point p in S_0 exists, any point q in S_{2^d-1} can be immediately pruned out after a point-wise dominance test on (p, q).
- **Partial dominance**: The relation can be classified into two subcases: (1) a self-pair (B_i, B_i) and (2) (B_i, B_j) with an arrow from B_i to B_j. Guided by these region-level relations, we then perform point-wise dominance tests to effectively identify point-level relations. First, for self-pairs B_i and B_i, we perform dominance



(a) Sorting-based scheme (b) Partitioning-based scheme

Figure 5: Skeletons of non-index skyline algorithms

tests in S_i itself. Second, for B_i and B_j with partial dominance relations, since points in S_i are likely to dominate those in S_j, we perform dominance tests between S_i and S_j. Recall that, we take the transitivity into account to find all of the partial dominance relations– If an arrow exists from B_i to B_j and an arrow from B_j to B_k, then B_i also partially dominates B_k. We thus perform dominance tests for partial dominance pairs (B_i, B_k).

- **Incomparability**: For all the remaining pairs, B_i and B_j are incomparable, which suggests that point sets S_i and S_j in corresponding regions R_i and R_j are also incomparable, *i.e.*, if $B_i \sim B_j$, then $\forall p \in S_i, q \in S_j : p \sim q$. We can thus bypass point-wise dominance tests between S_i and S_j.

4.2 Algorithm Skeletons and Cost Model

This section presents a cost model for dominance tests in non-index skyline algorithms. To achieve this goal, we abstract common skeletons of a class of non-index algorithms $\mathcal{A}$ from which common modules can be identified. Recall that, non-index skyline algorithms do not require pre-constructed indexes. Figure 5 depicts common skeletons of sorting-based algorithms such as BNL, SFS, SaLSa, and SSkyline, and partitioning-based algorithms such as OSPS and SkyTree, respectively. In these skeletons, two key common modules, **SelectPivot**() and **Prune**(), are identified.

We first model the cost of **Prune**() for these skeletons. Specifically, given a pivot point p^V, we can abstract **Prune**() as an optimal implementation Opt. To illustrate this, Algorithm 1 describes the pseudo code for Opt, based on which we measure the cost of Opt as the number of point-wise dominance tests. Since dominance tests monopolize the overall computation time (as argued in [12, 16]), our model can thus closely reflect the overall cost.

We then prove that Opt only requires minimal dominance tests to find correct skyline results, by showing that the cost of Opt is no higher than that of any arbitrary algorithm $A \in \mathcal{A}$. Suppose that the same pivot is used. Given an algorithm A and a dataset $\mathcal{S}$, let denote the cost model as $Cost(A, \mathcal{S})$. We formally state the optimality of Opt as follows:

THEOREM 1 (OPTIMALITY OF Opt) Given a set of any non-index algorithms $\mathcal{A}$, Algorithm Opt incurs the minimal cost, *i.e.*, $\forall A \in \mathcal{A} : Cost(\mathsf{Opt}, \mathcal{S}) \leq Cost(A, \mathcal{S})$.

PROOF. We prove this by contradiction. In other words, if A skips any dominance test performed by Opt, it may no longer guarantee that A finds correct skyline results. Specifically, we shows the dominance tests of Opt for the following three region-level relations described in the lattice.

Algorithm 1 Opt(p^V, $\mathcal{S}$)

```
1: S ← MapPointToRegion(p^V, S).
2: // Remove all points in S_{2^d−1} dominated by p^V.
3: S ← S − Dominance(p^V, S_{2^d−1}).
4: B ← {B_0, ..., B_{2^d−2}}.
5: for ∀(B_i, B_j) ∈ B × B do
6:    // Check partial dominance, and remove dominated points.
7:    if B_i ≺_Par B_j and S_i ≠ {} then
8:       S ← S − Dominance(S_i, S_j).
9:    else if B_i ∼ B_j then
10:      Continue. // Skip dominance tests between S_i and S_j.
11:   end if
12: end for
13: return S
```

- **Dominance (lines 2-3)**: When performing dominance tests between a point $q \in S_{2^d-1}$ and a pivot point p^V, Opt only requires one dominance test for each point. For S_{2^d-1}, the cost of Opt thus equals to $|S_{2^d-1}|$. To contradict, assume that an algorithm A exists with less dominance tests. For a skipped dominance test, A can include $q \in S_{2^d-1}$ as final skyline if q is only the point dominated by p^V. As a result, A causes incorrect results, which incurs a contradiction.

- **Partial dominance (lines 6-8)**: Opt needs to check dominance tests between p and q if S_i and S_j have a partial dominance relation. To contradict, assume that A can skip dominance tests between p and q. In this case, A can contain q as final skyline if q is the only point dominated by p. As a result, the result of A is incorrect, which incurs a contradiction.

- **Incomparability (lines 9-10)**: Given two regions R_i and R_j, Opt bypasses all the point-wise dominance tests corresponding to S_i and S_j. In this case, A can save as many equal dominance tests as Opt.

To sum up, an algorithm A performing less point-wise dominance tests cannot guarantee to find correct skyline results. In other words, a non-index algorithm has to perform at least as many dominance tests as Opt. □

To analyze Opt, we now adopt the general assumption used in prior analysis [6, 7, 8] for data distributions. Suppose that points are uniformly and independently distributed in d-dimensional space, *i.e.*, *uniform independence (UI)*. This UI condition has been widely adopted in prior work to estimate the number of points in R_i.

Based on this assumption, we analyze two important factors, dominance and incomparability, in Opt. We measure the power of dominance and incomparability to select a good pivot point. Specifically, a good pivot point maximally prunes out non-skyline points by dominance, while bypassing unnecessary dominance tests on incomparable point pairs. Let denote the number of dominated points by N_D, and number of incomparable pairs by N_I. For the sake of presentation, let $[B_i \sim B_j]$ denote a boolean condition, *i.e.*, 1 if B_i and B_j are incomparable; 0, otherwise. Given a pivot point p^V and a dataset $\mathcal{S}$, N_D and N_I in Opt are formally stated as follows:

THEOREM 2 (THE POWER OF A PIVOT POINT p^V IN Opt) Given a pivot point p^V and a dataset $\mathcal{S}$ under UI condition, we represent the number of dominated points and incomparable point pairs as $N_D(p^V, \mathcal{S}) = |S_{2^d-1}|$ and $N_I(p^V, \mathcal{S}) = \sum_{i=0}^{2^d-1} \sum_{j=i+1}^{2^d-2} |S_i||S_j|[B_i \sim B_j]$ respectively.

PROOF. First, suppose that a point q is mapped into B_{2^d-1}. In this case, q is dominated by a pivot point p^V. For S_{2^d-1}, we can represent the number of dominated points $N_D(p^V, \mathcal{S})$ as $|S_{2^d-1}|$. Second, for q in S_i, the number of incomparable pairs is $\sum_{j=0}^{2^d-2} |S_j|[B_i \sim B_j]$. By the symmetric property of the incomparability, we only consider B_j such that $i < j$. Considering every S_i, we can represent $N_I(p^V, \mathcal{S})$ as $\sum_{i=0}^{2^d-1} \sum_{j=i+1}^{2^d-2} |S_i||S_j|[B_i \sim B_j]$. □

According to Theorem 2, the cost function is thus inversely proportional to the sum of N_D and N_I as follows:

$$Cost(p^V, \mathcal{S}) \propto \frac{1}{N_D(p^V, \mathcal{S}) + N_I(p^V, \mathcal{S})}.$$

5. A COST-BASED PIVOT SELECTION

This section designs a systematic pivot point selection. Desirably, an ideal pivot point maximizes the power of dominance and incomparability discussed in Theorem 2. At each iteration, how do we select a pivot point p^V maximize both $N_D(p^V, \mathcal{S})$ and $N_I(p^V, \mathcal{S})$? Formally, the pivot point selection is presented as the optimization of selecting a point p that maximizes both N_D and N_I:

$$\begin{aligned} p^V &= \text{argmin}_{p \in \mathcal{S}} Cost(p, \mathcal{S}). \\ &= \text{argmax}_{p \in \mathcal{S}} N_D(p, \mathcal{S}) + N_I(p, \mathcal{S}). \end{aligned}$$

A naive solution to this cost-based optimization would be to compute the function for every point, which would involve performing dominance tests across all pairs of $2^d - 1$ subregions. This naive solution thus incurs a prohibitive cost of $O(4^d)$. As a result, estimating the exact function scores defeats the whole purpose of optimizing the pivot point selection.

To solve this problem, we propose a two-phase approximation approach, *i.e.*, *pruning-then-optimization*. In the *pruning phase*, we first reduce the search space into a subset of points with high N_D. In the *optimization phase*, we then pick a pivot point p^V from the subset identified as the first phase to maximize N_I. The property naturally suggests to find a pivot point p^V that balances both N_D and N_I consistently.

More specifically, each phase works as follows:

1. **Pruning phase**: To maximize N_D, we consider only skyline points as pivot candidates, since N_D of a skyline point is always larger than that of a non-skyline point it dominates.

2. **Optimization phase**: To maximize N_I, we find a skyline point that maximizes the number of incomparable pairs. This optimization can select a pivot point p^V that balances both N_D and N_I.

We now discuss how to find a pivot point p^V maximizing N_I in skyline points.

LEMMA 1 Given incomparable two regions R and R', there exists at least one two-dimensional subspace $\{d_i, d_j\}$ such that $R \sim_{\{d_i,d_j\}} R'$.

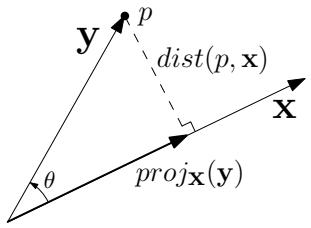


Figure 6: The projection of vector y onto vector x

PROOF. By Definition 8, if $R \sim R'$, then $\exists d_i \in \mathcal{D} : B.d_i < B'.d_i$ and $\exists d_j \in \mathcal{D} : B'.d_j < B.d_j$. □

LEMMA 2 Given $\mathcal{S}$ in two-dimensional space $\{d_i, d_j\}$ under UI condition, a pivot point $p^V = (p_i^V, p_j^V)$ maximally bypasses dominance tests for incomparability, if p^V has equal values such that $p_i^V = p_j^V$.

PROOF. A pivot point p^V divides $\mathcal{S}$ into four subregion, where R_1 and R_2 are incomparable. We can quantity the dominance tests bypassed by incomparability as $|S_1| \times |S_2|$. Under UI condition, $|S_1|$ and $|S_2|$ are proportional to the size of subregions R_1 and R_2 such that $p_i^V \times (1 - p_j^V)$ and $(1 - p_i^V) \times p_j^V$ respectively. In this case, when p^V has the equal values such that $p_i^V = p_j^V$, p^V maximizes dominance tests for incomparability. □

Based on Lemmas 1 and 2, we formally present a desirable pivot point p^V under UI condition.

THEOREM 3 (MAXIMIZATION OF INCOMPARABILITY) Given $\mathcal{S}$ under UI condition, p^V maximally bypasses dominance tests for incomparability, if pivot point $p^V = (p_1^V, \ldots p_d^V)$ has equal values such that $p_1^V = \ldots = p_d^V$,

PROOF. By Lemma 1, we first project d-dimensional space into incomparable two-dimensional subspace $D = \{d_i, d_j\}$ such that $D \subseteq \mathcal{D}$. By Lemma 2, we then select a pivot point p^V such that $p_i^V = p_j^V$ to maximize dominance tests for incomparability. By extending into all possible incomparable two-dimensional subspaces, p^V such that $p_1 = \ldots = p_d$ maximizes dominance tests on d-dimensional space. □

According to Theorem 3, the incomparability is maximized when p^V lies on a *diagonal* line. That is, we select a pivot point p^V as a point p with the the smallest "coordinate spread" from the diagonal. Specifically, to quantify how close a point p is to the diagonal line, we thus compute the distance $dist(p, \mathbf{x})$ between a point p and a *diagonal vector* $\mathbf{x} = (x_1, \ldots, x_d)$ such that $x_i = \ldots = x_d$, using the projection vector depicted in Figure 6.

$$dist(p, \mathbf{x}) = \sqrt{||p||^2 - ||proj_{\mathbf{x}}(\mathbf{y})||^2} \quad (1)$$

Let $proj_{\mathbf{x}}(\mathbf{y})$ denote the projection of a vector $\mathbf{y}$ onto a vector $\mathbf{x}$. The projection vector $proj_{\mathbf{x}}(\mathbf{y})$ is calculated as follows:

$$proj_{\mathbf{x}}(\mathbf{y}) = ||\mathbf{y}|| \cdot \cos\theta = \left(\frac{\mathbf{x} \cdot \mathbf{y}}{\mathbf{x} \cdot \mathbf{x}}\right) \mathbf{x}. \quad (2)$$

As $p = (p_1, \ldots, p_d)$ and $\mathbf{x}$ has equal values, projection vector $proj_{\mathbf{x}}(\mathbf{y})$ is represented as:

$$proj_{\mathbf{x}}(\mathbf{y}) = \frac{x_i(p_1 + \ldots + p_d)}{d \cdot x_i^2} \cdot \mathbf{x}, \quad (3)$$

Algorithm 2 SelectPivotPoint($\mathcal{S}$)

Input: A dataset $\mathcal{S} = \{p^1, \ldots, p^n\}$
Output: A pivot point p^V

```
1: head ← 1, cur ← 2, tail ← n. // Initialize variables.
2: mindist ← dist(S[head]).
3: while cur ≤ tail do
4:    if S[head] ≺ S[cur] then
5:       S.Remove(cur). // Remove a dominated point.
6:       tail ← tail − 1.
7:    else if S[cur] ≺ S[head]. then
8:       S[head] ← S[cur]. // Change a head point.
9:       S.Remove(cur).
10:      tail ← tail − 1, cur ← head + 1.
11:      mindist ← dist(S[head]).
12:   else
13:      // Compare mindist with curdist of S[cur].
14:      curdist ← dist(S[cur]).
15:      if curdist < mindist then
16:         if ∀q ∈ S[2, cur − 1] : q ⊀ S[cur] then
17:            Swap S[cur] and S[head].
18:            curdist ← mindist.
19:         end if
20:      end if
21:      cur ← cur + 1.
22:   end if
23: end while
24: S ← S[1, tail]. // Update S as the remaining points.
25: p^V ← S[head].
26: return p^V
```

According to Equation 3, the distance of $proj_{\mathbf{x}}(\mathbf{y})$ is calculated as:

$$||proj_{\mathbf{x}}(\mathbf{y})|| = \sqrt{\frac{(p_1 + \ldots + p_d)^2}{d}}. \tag{4}$$

Since $||p||$ is $\sqrt{p_1^2 + \ldots + p_d^2}$, Equation 1 is represented as follows:

$$\begin{aligned} dist(p, \mathbf{x}) &= \sqrt{(p_1^2 + \ldots + p_d^2) - \frac{(p_1 + \ldots + p_d)^2}{d}}. \\ &= \sqrt{\frac{(p_1 - p_2)^2 + \ldots + (p_{d-1} - p_d)^2}{d}}. \end{aligned} \tag{5}$$

In this case, the numerator of the second equation requires us to subtract all (p_i, p_j) combinations, which incurs an quadratic computation cost, *i.e.*, $\binom{d}{2}$.

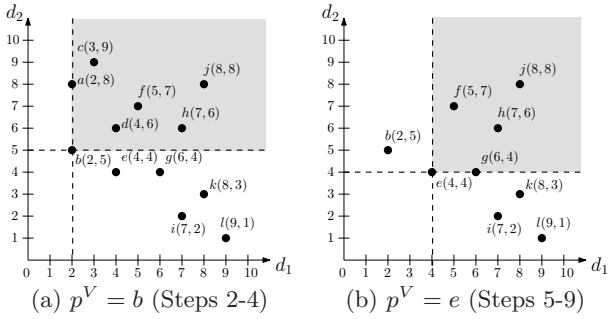
As the exact computation requires expensive quadratic computation, we consider its upper bound as an alternative for more efficient distance computation. Observe that, in Equation 5, $|p_i - p_j|$ $(i \neq j)$ is no less than $p_{max} - p_{min}$ where $p_{max} = \text{argmax}_{i \in [1,d]} p_i$ and $p_{min} = \text{argmin}_{i \in [1,d]} p_i$. We can thus derive the upper bound of $dist(p, \mathbf{x})$:

$$\begin{aligned} dist(p, \mathbf{x}) &\leq \sqrt{\frac{(p_{max} - p_{min})^2 \times \binom{d}{2}}{d}}. \\ &\leq dist(p) \times \sqrt{\frac{d-1}{2}}, \end{aligned} \tag{6}$$

where $dist(p) = p_{max} - p_{min}$. Since $\sqrt{\frac{d-1}{2}}$ is a constant regardless of points, we solely consider $dist(p)$ instead of $dist(p, \mathbf{x})$.

To sum up, we would exploit $dist(p)$ presenting the upper bound of $dist(p, \mathbf{x})$, only requiring $O(d)$ computation, to select a desirable pivot point p^V.

While our approach is two-phase in principle, its implementation does not necessarily have to follow two phases.



(a) $p^V = b$ (Steps 2-4) (b) $p^V = e$ (Steps 5-9)

Figure 7: Balanced pivot point selection in two dimensional space

Instead, we can interleave the two phases into an efficient one-phase algorithm. Specifically, we perform point-wise dominance tests while maintaining (1) a skyline point with (2) the shortest upper bound distance.

Algorithm 2 describes the pseudo code of our proposed one-phase implementation for pivot point selection. To capture a updated pivot point, we maintain a head point as a pivot point p^V. Specifically, when the head point does not dominate the currently accessed point in stored order, we check whether the distance of the current point is shorter than that of the head point.

More specifically, we classify the relations between head point p^V and current point q into three cases:

1. $p^V \prec q$ (lines 4-6): As the pruning phase, q is removed without calculating $dist(q)$.

2. $q \prec p^V$ (lines 7-11): As the pruning phase, p^V is replaced with q, and $dist(p^V)$ is updated.

3. $p^V \sim q$ (lines 12-22): As the optimization phase, we first compute $dist(q)$. If $dist(q)$ is smaller than $dist(p^V)$, we check whether q is not dominated by the previously accessed points. If this is true, we swap p^V with q, and $dist(p^V)$ is updated. Note, while performing dominance tests between q and previous accessed points, we can remove additionally dominated points.

To illustrate our pivot point selection, Figure 7 depicts an example with 12 points on two dimensional space. (The shadowed rectangles present the dominance regions of current pivot points.) Suppose that data points are accessed in alphabetical order. For each step, Table 2 presents the actions of the pivot point selection. Note, when $p^V \prec q$, the computation of $dist(q)$ is skipped, represented as "-" in Table 2. First, since $b \prec a$, p^V is replaced with b (Step 1). Next, b is compared with c and d, where $b \prec c$ and $b \prec d$ (Steps 2-3). When b is compared with e, two points are swapped, since $dist(e) < dist(b)$. (Step 4). By comparing e with other points, we remove dominated points $\{f, g, h, j\}$ in order (Steps 5-9). Point e is finally selected as p^V, since $dist(e)$ is the shortest distance (Steps 10-11). Maintaining p^V as a skyline point with the shortest distance, we simultaneously remove dominance points, where $\mathcal{S}$remains five points $\{e, b, i, k, l\}$. As a result, we have to only consider the remaining points to identify final skyline points.

Step	p^V	$dist(p^V)$	q	$dist(q)$	Relation
1	a(2,8)	6	b(2,5)	3	$b \prec a$
2	b(2,5)	3	c(3,9)	-	$b \prec c$
3	b(2,5)	3	d(4,6)	-	$b \prec d$
4	b(2,5)	3	e(4,4)	0	$b \sim e$
5	e(4,4)	0	f(5,7)	-	$e \prec f$
6	e(4,4)	0	g(6,4)	-	$e \prec g$
7	e(4,4)	0	h(7,6)	-	$e \prec h$
8	e(4,4)	0	i(7,2)	5	$e \sim i$
9	e(4,4)	0	j(8,8)	-	$e \prec j$
10	e(4,4)	0	k(8,3)	5	$e \sim k$
11	e(4,4)	0	l(9,1)	8	$e \sim l$

Table 2: Actions of pivot point selection in Figure 7

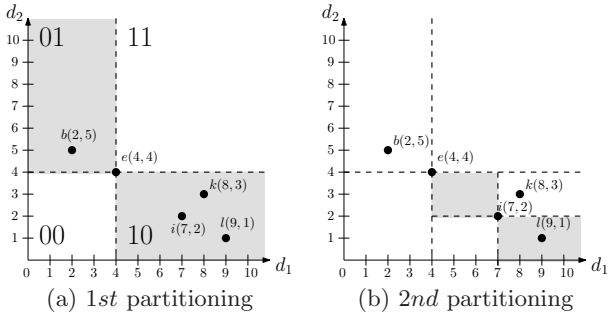
Algorithm 3 BSkyTree-S($\mathcal{S}$)

Input: A dataset $\mathcal{S} = \{p^1, \ldots, p^n\}$
Output: A set of skyline points SKY($\mathcal{D}$)
1: $p^V \leftarrow$ **SelectPivotPoint**($\mathcal{S}$). // *Perform Algorithm 2.*
2: $\mathcal{S} \leftarrow$ **MapPointToRegion**(p^V, $\mathcal{S}$). // *Region-level mapping*
3: $\mathcal{S} \leftarrow \mathcal{S} - S_{2^d-1}$. // *Remove dominated points in* S_{2^d-1}.
4: $head \leftarrow 2$, $tail \leftarrow |\mathcal{S}|$. // *Initialize variables.*
5: **while** $head < tail$ **do**
6: $cur \leftarrow head + 1$.
7: **while** $cur \leq tail$ **do**
8: **if** $\mathcal{S}[head].B \prec_{Par} \mathcal{S}[cur].B$ or vice verse **then**
9: **if** $\mathcal{S}[head] \prec \mathcal{S}[cur]$ **then**
10: $\mathcal{S}$.Remove(cur). // *Remove a dominated point.*
11: $tail \leftarrow tail - 1$.
12: **else if** $\mathcal{S}[cur] \prec \mathcal{S}[head]$ **then**
13: $\mathcal{S}[head] \leftarrow \mathcal{S}[cur]$. // *Change a head point.*
14: $\mathcal{S}$.Remove(cur).
15: $tail \leftarrow tail - 1$, $cur \leftarrow head + 1$.
16: **else**
17: $cur \leftarrow cur + 1$. // *Point-level incomparability*
18: **end if**
19: **else**
20: $cur \leftarrow cur + 1$. // *Region-level incomparability*
21: **end if**
22: **end while**
23: **if** $head < tail$ **then**
24: $head \leftarrow head + 1$.
25: **end if**
26: **end while**
27: SKY($\mathcal{D}$) $\leftarrow \mathcal{S}[1, tail]$.
28: **return** SKY($\mathcal{D}$)

6. INSTANTIATIONS OF Opt

This section proposes two instantiations of Opt, BSkyTree-S and BSkyTree-P, using our balanced pivot point selection. Specifically, these instantiations can be viewed as enhanced versions for each scheme: the sorting- and partitioning-based algorithms, as we describe below.

- BSkyTree-S: This scheme accesses points in stored order after the pivot point selection. This instantiation can improve existing sorting-based algorithms, by (1) bypassing dominance tests between incomparable subregions and (2) dynamically changing dominated points.
- BSkyTree-P: This scheme adopts a divide-and-conquer strategy by recursively partitioning the entire data into 2^d subsets. This instantiation can improve existing partitioning-based algorithms, by adopting the pivot point selection considering both dominance and incomparability.



(a) 1*st* partitioning (b) 2*nd* partitioning

Figure 8: Recursive partitioning in two dimensional space

6.1 Algorithm BSkyTree-S

Algorithm 3 depicts the pseudo code of BSkyTree-S. In lines 1-3, we select a pivot point p^V using Algorithm 2. Accordingly, each point is then mapped to partitioned subregions, and dominated points in S_{2^d-1} are pruned out. In lines 5-28, we sequentially access points in stored order updated by the pivot point selection, and perform point-wise dominance tests, *i.e.*, by comparing a fixed head point with all of the other points. Note, in line 8, we check the region-level relations unlike the existing sorting-based algorithms–if no region-level partial dominance relations exist, the point-wise dominance tests can be bypassed, implying region-level incomparability. This enables us to significantly save computation costs, compared to existing sorting-based algorithms [6, 7, 1, 15].

We now illustrate BSkyTree-S using an example dataset in Figure 8. Recall that, after pivot point e is determined by leveraging Algorithm 2, the remaining points in $\mathcal{S}$ are $\{e, b, i, k, l\}$. Since e has already been determined as a skyline point, we only need to check the remaining four points. Table 3 describes the actions of BSkyTree-S. (Let $\mathcal{S}[head].B$ and $\mathcal{S}[cur].B$ denote binary vectors of a head point and a current point respectively.) For point b, we can skip the dominance tests for other points, since 01 $\sim$ 10 (Steps 1-3). For the other points, the point-wise dominance tests are performed (Steps 4-6). As a result, final skyline points are $\{e, b, i, l\}$.

Step	$\mathcal{S}[head]$	$\mathcal{S}[head].B$	$\mathcal{S}[cur]$	$\mathcal{S}[cur].B$	SKY($\mathcal{S}$)
1	b(2,5)	01	i(7,2)	10	$\{e\}$
2	b(2,5)	01	k(8,3)	10	$\{e\}$
3	b(2,5)	01	l(9,1)	10	$\{e, b\}$
4	i(7,2)	10	k(8,3)	10	$\{e, b\}$
5	i(7,2)	10	l(9,1)	10	$\{e, b, i\}$
6	l(8,2)	10	-	-	$\{e, b, i, l\}$

Table 3: Actions of BSkyTree-S in Figure 8

6.2 Algorithm BSkyTree-P

Algorithm 4 describes the pseudo code of BSkyTree-P. In lines 3-10, we determine a pivot point p^V using Algorithm 2, based on which points are mapped to partitioned subregions. A key difference from BSkyTree-S is that, BSkyTree-P recursively rearranges points into subsets in a divide-and-conquer manner. Let $\mathcal{H}$ denote a list of point subsets mapped to binary vectors. In lines 11-21, given a subset $\mathcal{H}[i]$, we check a subset $\mathcal{H}[j]$ with partial dominance relations. (That is, $\mathcal{H}[i]$ and $\mathcal{H}[j]$ are a subset of points mapping to B_i and

Algorithm 4 BSkyTree-P($\mathcal{S}$)

Input: A dataset $\mathcal{S} = \{p^1, \ldots, p^n\}$
Output: A set of skyline points SKY($\mathcal{S}$)

```
1: max ← 2^d − 2. // Set the size of a lattice.
2: H[1, max] ← {}. // Initialize a list H.
3: p^V ← SelectPivotPoint(S). // Perform Algorithm 2.
4: SKY(S).Add(p^V). // p^V is a skyline point.
5: for ∀ p ∈ S do
6:    i ← MapPointToRegion(p^V, p).
7:    if i ≤ max then
8:       H[i].Add(p). // Add points mapped into B to H[i].
9:    end if
10: end for
11: for i ← 1 to max do
12:    if H[i].Size() > 0 then
13:       for ∀ j ∈ [1, i]: B_j ≺_Par B_i do
14:          H[i] ← Dominance(H[j], H[i]).
15:       end for
16:       if H[i].Size() > 0 then
17:          T ← BSkyTree(H[i]). // Recursive partitioning.
18:          SKY(S).Add(T).
19:       end if
20:    end if
21: end for
22: return SKY(S)
```

B_j respectively.) If this is true, point-wise dominance tests are performed, and the dominated points in $\mathcal{H}[i]$ are eliminated. Otherwise, the dominance tests can be skipped. We recursively conquer $\mathcal{H}[i]$ in the same fashion until all skyline points are identified.

To illustrate BSkyTree-P, we explain recursive region-level partitioning of BSkyTree-P as depicted in Figure 8. Given a pivot point e, the remaining points are divided into two subgroups $\mathcal{H}[1] = \{b\}$ and $\mathcal{H}[2] = \{i, k, l\}$. We recursively partition them into finer subsets until all the selected pivot points become skyline points. Finally, the skyline points are $\{e, b, i, l\}$.

As we will report on in Section 7, recursive partitioning of BSkyTree-P (compared to BSkyTree-S) can cause significant computation overhead when the partitions are (1) too many (2) loosely populated. For instance, in high dimensional space, BSkyTree-P can have too many loosely populated partitions, *i.e.*, $2^d >> n$, where partitioning overhead may outweigh its cost benefit.

We thus optimize BSkyTree-P to avoid the two problems. First, when the partitions are too many, we reduce dimensionality d into sub-dimensionality d' until $2^{d'} < n$. Second, to remove underpopulated partitions, we set cardinality threshold δ and simply execute BSkyTree-S instead of partitioning further. (Empirically, we set the threshold δ as 1000, as the performance of BSkyTree-S is better than that of BSkyTree-P.) We observed that applying the above optimization may increase dominance tests, but save the overall response time by reducing the computation overhead for partitioning.

7. EXPERIMENTS

This section presents the empirical evaluation results for our proposed algorithms, BSkyTree-S and BSkyTree-P. We first explain the experimental settings. We then validate the effect of pivot point selections, and evaluate the scalability of our proposed algorithms by comparing them with state-of-the-art skyline algorithms in extensive synthetic datasets and real-life datasets.

7.1 Experimental Settings

To validate our algorithms, we synthetically generated extensive synthetic datasets, varying three parameters– distribution, cardinality, and dimensionality. All the attributes values are positive real numbers in (0, 1). Specifically:

- Distribution: We generated three datasets, *i.e.*, Correlated (COR), Independent (IND), and Anti-correlated (ANT), following data generation instructions in [4]. (We do not report results for correlated distribution in this paper, as the findings are consistent with the results from other distributions.)
- Dimensionality: We varied dimensionality d from 4 to 22. (Default: $d = 12$)
- Cardinality: We varied cardinality n from $200K$ to $1{,}000K$. (Default: $n = 200K$)

Figures 9 and 10 describe the average number of skyline points in our synthetic datasets. As the number of skyline points directly affects the performance, we will later revisit these numbers as references when explaining the experimental results.

We also evaluate our algorithms with real-life datasets such as NBA and Household [1]. Specifically, these datasets have 8-dimensional 17,264 points and 6-dimensional 127,931 points respectively.

Using these settings, we compare our algorithms against the following state-of-the-art skyline algorithms. Note, when we compare ours with SFS and SaLSa, assume that they can obtain sorted order for free. We stress that this assumption is impractical and unfavorable to our algorithms. However, our intention is to show how our algorithms outperform these algorithms even in such unfavorable settings.

- Sorting-based algorithms:
  - Algorithm SFS [6, 7]: We implement SFS that accesses points in descending order of the size of dominance regions. To generate this order, SFS uses the entropy function for sorting, *i.e.*, $f(p) = \sum_{i=1}^{d} ln(p_i + 1)$, which is proportional to the size of dominance regions.
  - Algorithm SaLSa [1]: SaLSa improves SFS by using $minC$ function $\min_{i \in [1,d]} p_i$ for early termination combining the following sum function $\sum_{i=1}^{d} p_i$ as a tie-breaker. In particular, we further optimize the implementation of SaLSa, by maintaining skyline points in decreasing order of the dominance regions. This optimization enables to reduce dominance tests for non-skyline points by taking advantage of SFS.
  - Algorithm SSkyline [15]: SSkyline is a non-index skyline algorithm, which performs nested-loop computation as point-wise dominance tests. However, SSkyline, by moving a dominated point to the end of data list after each dominance test, guarantees to avoid redundant dominance comparisons for non-skyline points.

[1]These datasets were crawled from http://www.nba.com and http://www.ipums.org respectively.

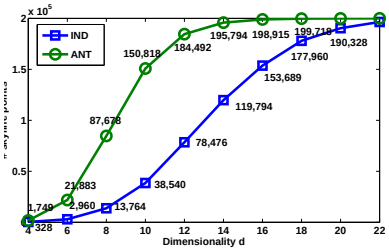


Figure 9: The number of skyline points s varying d

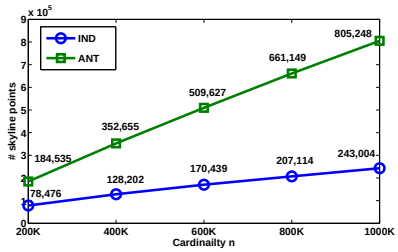


Figure 10: The number of skyline points s varying n

- Partitioning-based algorithms: Existing partitioning-based algorithms [11, 16] can be viewed as instantiations of BSkyTree-S and BSkyTree-P that adopt two heuristics pivot selection schemes MaxDom and Random respectively. To compare our proposed pivot selection with these algorithms, Section 7.2 will report on a comparison of different pivot selections in extensive settings.

To validate the performance, we introduce two measures, *response time* and *the number of dominance tests per point* (DT). Formally,

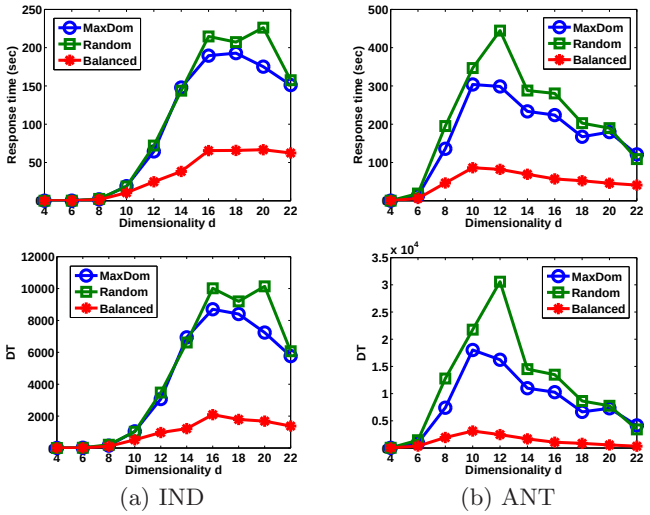
$$DT = \frac{\text{The total number of dominance tests}}{n}.$$

All experiments were conducted through Windows XP with an Intel Core Duo 2.66 GHz CPU and 2GB RAM over our C++ implementations of these algorithms. Here, to focus on the CPU-cost, we assume that all the points can fit in the main memory for all algorithms, though these algorithms can be straightforwardly modified into disk-based algorithms.

7.2 The Effect of Pivot Selection

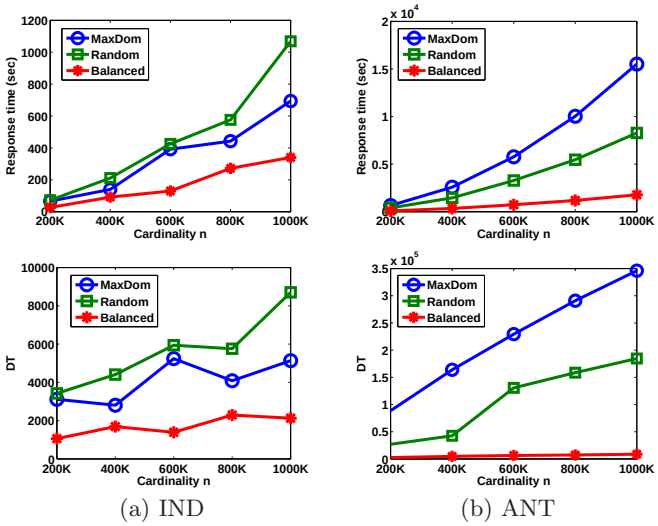
We first evaluate the effect of pivot selection for our proposed algorithms BSkyTree-S and BSkyTree-P, by comparing three different pivot selections– MaxDom, Random, and Balanced. As explained earlier, recall that, these comparison results can be alternatively viewed as comparing our proposed algorithms with existing partitioning-based algorithms [11, 16] adopting MaxDom and Random respectively.

- MaxDom: This selection heuristic selects a pivot point as a skyline point maximizing the dominance region [11]. That is, we select a point minimizing the sum of the point values such that $p^V = \text{argmin}_{p \in \mathcal{S}} \sum_{i=1}^{d} p_i$. This selection thus requires a linear scan on the whole dataset. While this pivot selection guarantees to maximize the pruning power for dominance, it does not provide any guarantee on maximizing incomparability.



(a) IND (b) ANT

Figure 11: Varying d and pivot point selections in BSkyTree-S



(a) IND (b) ANT

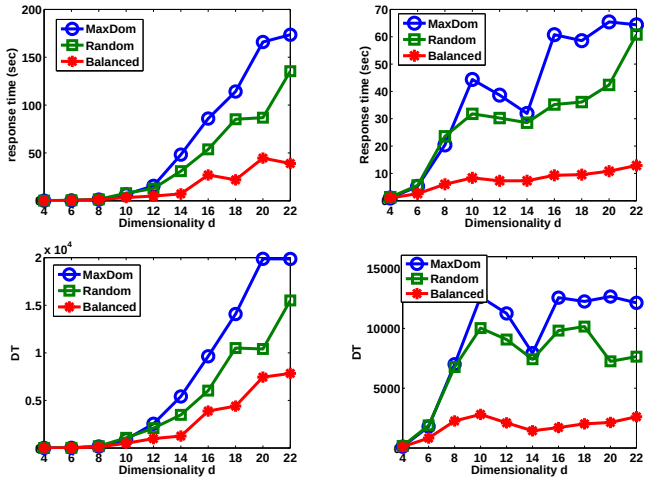
Figure 12: Varying n and pivot point selections in BSkyTree-S

- Random: This selection heuristic selects a pivot point as a random skyline point [16]. First, we randomly set a weight vector $w = (w_1, \ldots w_d)$ such that $\sum_{i=1}^{d} w_i = 1$, and then select a point minimizing the following weight function such that $p^V = \text{argmin}_{p \in \mathcal{S}} \sum_{i=1}^{d} w_i \cdot p_i$. Note, when weight values are are identical, Random is the same as MaxDom. This heuristic cannot provide any guarantee on maximizing both dominance and incomparability.

- Balanced: This selection selects a pivot point by considering both dominance and incomparability, using Algorithm 2.

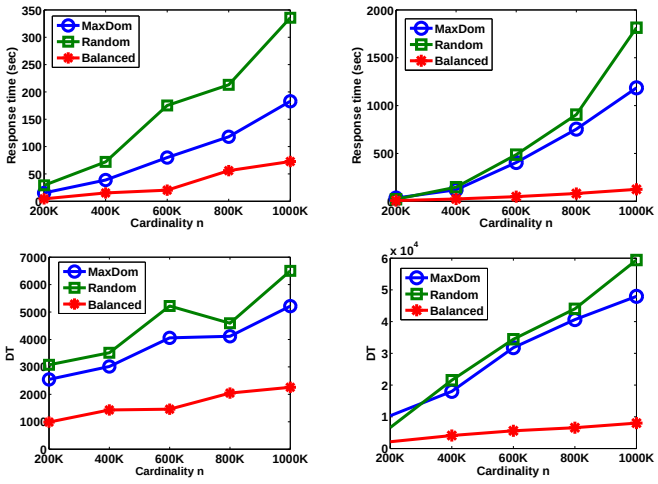
7.2.1 *Evaluation of* BSkyTree-S

We first evaluate three pivot point selection schemes implemented on BSkyTree-S over varying d, as depicted in Figure 11. Observe that the heuristic pivot selections, MaxDom and Random, incur significantly higher costs than Balanced.



(a) IND (b) ANT

Figure 13: Varying d and pivot point selections in BSkyTree-P



(a) IND (b) ANT

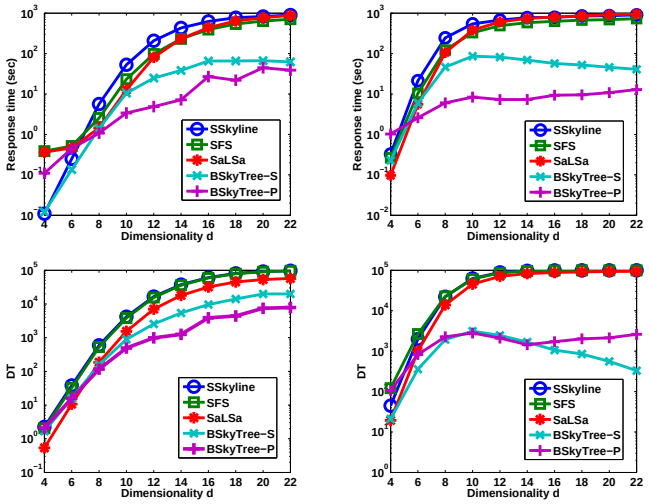
Figure 14: Varying n and pivot point selections in BSkyTree-P

In particular, this gap tends to increase as d increases, as the number of skyline points also increases over d as depicted in Figure 9. The gap is maximized when $d = 12$ in Figure 11(b). In this case, Balanced achieves about 5 times speedup over Random. However, the gap starts to decrease, when $d > 12$. While the increase in number of skyline points flattens as depicted in Figure 9, the number of subregions 2^d keeps increasing exponentially. Since BSkyTree-S performs implicit partitioning, this phenomenon helps to fully exploit the incomparability.

We also compare three pivot point selections over varying n, as depicted in Figure 12. Balanced consistently outperforms other heuristic pivot selections. In particular, while Balanced shows near-constant performances, other heuristics incur linear costs, as n increases.

7.2.2 *Evaluation of* BSkyTree-P

We then compare the pivot point selections implemented on BSkyTree-P. Figures 13 and 14 depict the comparison



(a) IND (b) ANT

Figure 15: Varying d

results over varying d and n respectively. Unlike BSkyTree-S optimizing global pivot ordering, BSkyTree-P recursively partitions a region, and divides the pivot selection problem into the same problem in its subregions. Although this partitioning requires additional overhead, the overhead starts to pay off in high dimensionality, by leveraging region-specific incomparability. (We will further explain this trade-off in Section 7.3). In particular, observe that Balanced saves up to 10 and 14 speedups over MaxDom and Random, respectively, when n=1,000K as depicted in Figure 14(b). Note that, we do not observe a peak in performances around $d = 12$ in our previous evaluation as reported in Figure 11(b), since BSkyTree-P recursively performs explicit partitioning unlike BSkyTree-S. Meanwhile, we leverage the use of subdimensionality and the execution of BSkyTree-S to optimize the overhead for partitioning. For every pivot point selection, the optimizations help to prevent exponential overhead increases in high dimensional space.

7.3 Scalability

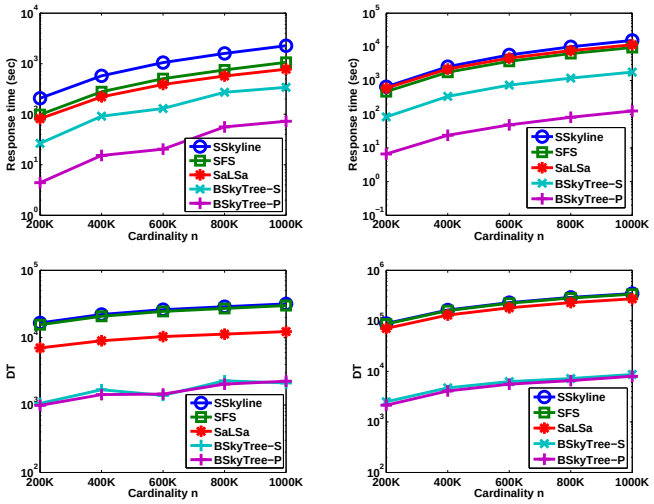
We now compare BSkyTree-S and BSkyTree-P with state-of-the-art algorithms such as SFS, SaLSa, and SSkyline. Recall that, we compared our algorithms only with these sorting-based algorithms, as our results in Section 7.2 have already demonstrated the scalability of our algorithm over existing partition-based algorithms. Similarly to Section 7.2, we compared BSkyTree-S and BSkyTree-P with these sorting-based algorithms over varying d and n. Note that, all experimental results in these settings are reported in log-scaled graphs, as the performance gaps are of two orders of magnitude.

7.3.1 *Dimensionality*

We compare BSkyTree-S and BSkyTree-P with existing algorithms over varying d, as depicted in Figure 15. Observe that existing algorithms are sensitive over d, since they do not fully exploit incomparability. In clear contrast, our algorithms demonstrate high scalability over d.

We also observe that our two proposed algorithms show the following complementary strengths over d:

- Low dimensionality (when $d < 8$): BSkyTree-S, by not



(a) IND (b) ANT

Figure 16: Varying n

requiring explicit partitioning overhead, demonstrates a higher performance over BSkyTree-P requiring explicit recursive partitioning.

- High dimensionality (when $d \geq 8$): BSkyTree-P starts paying off in high-dimensional space, by decreasing the size of the problem in partitioning (which near-quadratically affects the performance). For high dimensional space, this benefit starts to outweigh the overhead for partitioning. In particular, BSkyTree-P demonstrates about 100 times speedup over existing algorithms when $d = 22$.

7.3.2 *Cardinality*

We also report our evaluation results over varying n, as depicted in Figure 16. In a Similar way to d, our algorithms significantly outperform existing algorithms over all n, especially in high cardinality data. For instance, in all cardinality on the anti-correlated dataset, BSkyTree-P achieves up to two orders of magnitude.

7.4 Real-life datasets

This section reports on the response time and DT of all algorithms over real-life datasets. Specifically, Table 4 compares the response time (and DT in parentheses) for varying pivot selections implemented on BSkyTree-S and BSkyTree-P. We can observe that Balanced consistently outperforms other pivot point selections in real-life datasets as well. Table 5 reports the comparison results along with other existing algorithms. Just as we observed in synthetic datasets, BSkyTree-P has an advantage over BSkyTree-S in high cardinality data. In other words, BSkyTree-S and BSkyTree-P show a higher performance in NBA (smaller) and Household (larger) datasets respectively.

8. CONCLUSION

This paper studied the optimization problem of skyline query processing. To achieve this goal, we first identified common skeletons for existing sorting- and partitioning-based algorithms from which a cost model was derived. We then devised a systematic pivot selection driven by this cost model that exploited both the dominance and incomparability as key optimization factors. We implemented two algorithms BSkyTree-S and BSkyTree-P using the pivot point selection, which significantly outperformed existing algorithms by up to two orders of magnitude.

Algorithms	Household $d = 6$, $n = 127{,}931$ $s = 5{,}774$ (4.51%)	NBA $d = 8$, $n = 17{,}264$ $s = 1{,}796$ (10.40%)
BSkyTree-S+ MaxDom	0.648 ($DT = 124$)	0.050 ($DT = 16$)
BSkyTree-S+ Random	0.749 ($DT = 134$)	0.053 ($DT = 16$)
BSkyTree-S+ Balanced	0.360 ($DT = 59$)	0.033 ($DT = 7$)
BSkyTree-P+ MaxDom	0.575 ($DT = 129$)	0.173 ($DT = 12$)
BSkyTree-P+ Random	0.421 ($DT = 96$)	0.162 ($DT = 10$)
BSkyTree-P+ Balanced	0.303 ($DT = 83$)	0.157 ($DT = 8$)

Table 4: Varying pivot point selections in BSkyTree-S and BSkyTree-P

Algorithms	Household $d = 6$, $n = 127{,}931$ $s = 5{,}774$ (4.51%)	NBA $d = 8$, $n = 17{,}264$ $s = 1{,}796$ (10.40%)
SSkyline	0.899 ($DT = 193$)	0.050 ($DT = 16$)
SFS	0.302 ($DT = 179$)	0.041 ($DT = 16$)
SaLSa	0.329 ($DT = 94$)	0.053 ($DT = 15$)
BSkyTree-S	0.360 ($DT = 59$)	0.033 ($DT = 7$)
BSkyTree-P	0.303 ($DT = 83$)	0.157 ($DT = 8$)

Table 5: Comparisons with state-of-the-art algorithms

9. REFERENCES

[1] I. Bartolini, P. Ciaccia, and M. Patella. Efficient sort-based skyline evaluation. *ACM TODS*, 2008.

[2] J. Bentley, H. Kung, M. Schkolnick, and C. Thompson. On the average number of maxima in a set of vectors and applications. In *JACM*, 1978.

[3] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In *SODA*, 1991.

[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[5] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, 2006.

[6] J. Chomicki, P. Godfery, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.

[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting: Theory and optimizations. In *Intelligent Information Systems*, 2005.

[8] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, 2005.

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

[10] H. T. Kung, F. Luccio, and F. Preparata. On finding the maxima of a set of vectors. *JACM*, 1975.

[11] J. Lee and S. Hwang. SkyTree: Scalable skyline computation for sensor data. In *SensorKDD*, 2009.

[12] K. C. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the skyline in Z order. In *VLDB*, 2007.

[13] M. Morse, J. M. Patel, and H. Jagadish. Efficient skyline computation over low-cardinailty domains. In *VLDB*, 2007.

[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progessive algorithm for skyline queries. In *SIGMOD*, 2003.

[15] S. Park, T. Kim, J. Park, J. Kim, and H. Im. Parallel skyline computation on multicore architectures. In *ICDE*, 2009.

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