Explanations for Skyline Query Results

Sean Chester and Ira Assent
Data-Intensive Systems Group
Aarhus Universitet, Åbogade 34, 8200 Århus N, Denmark
schester@cs.au.dk ira@cs.au.dk

ABSTRACT

Skyline queries are a well-studied problem for multidimensional data, wherein points are returned to the user iff no other point is preferable across all attributes. This leaves only the points most likely to appeal to an arbitrary user. However, some dominated points may still be interesting, and the skyline offers little support for helping the user understand why some interesting points are omitted from the results. In this paper, we introduce the *Sky-not query*. Given a query point p, a dataset $\mathcal{S}$, and constraints with bounding corners q_{L} and q_{U}, the Sky-not query returns the alternative constraints q'_{L} closest to q_{L} for which p is in the skyline. This equips the user with an understanding of not just that a point was dominated, but also how severely. He can then assess himself whether the point is competitive.

We first propose theoretical results that show how to drastically reduce the input processed by a Sky-not query, independent of any algorithm. We then offer a skyline-like and an efficient recursive algorithm for solving Sky-not queries, which we evaluate in an extensive experimental evaluation.

1. INTRODUCTION

When exploring unfamiliar data, the *skyline operator* [3] can identify balances among multiple (possibly conflicting) attributes. It selects only those data points for which no other point is preferable across all attributes. Consider the canonical example of selecting a hotel, given the fictitious ones in Table 1. Budget Sleepz (B) is both cheaper and closer to the beach than Cozy Cabin (C); so, it is said to *dominate* the latter. The skyline is exactly and only those points not dominated by any others, in this case not including Cozy Cabin, but including the remainder: $\{\mathrm{A}, \mathrm{B}, \mathrm{D}, \mathrm{E}\}$. For a user, however, this indicates only that C is dominated by some other point, not which, nor how severely. C may be a reasonably competitive choice, even if it is dominated.

More generally, a (skyline) query typically includes range constraints (i.e., a `WHERE` clause) for more sophisticated expression. The user searching for hotels may want the most affordable option, but not something "cheap." Or, he may want a location remote from the city, but reachable within a few hours. Setting these constraints perfectly can be a challenging and iterative process, since notions such as "cheap" and "a few hours" are intentionally under-specified by the user and data-dependent. Yet, the constraints can have substantial impact on the query results, both adding and removing skyline points [6]. For the user, there is no support for understanding why data points are dominated, nor by how much. Especially when comparing several similar queries, the differences can be quite perplexing [18].

ID	Name	Price/nt	Distance
A	Abode Abroad	45	2100m
B	Budget Sleepz	30	4200m
C	Cozy Cabin	40	4500m
D	Diamond Harbour Inn	175	300m
E	Extravagansium	325	100m

Table 1: A fictitious city's hotel offerings.

Furthermore, not all decision criteria are necessarily reflected in the data attributes. Other factors might contribute to preferring a non-skyline point. For example, a user may favour Cozy Cabin on recommendation from a friend or because of a successful advertising campaign. In such a case, the skyline may do a disservice by hiding results that the user expects, ones that match the constraints of his search and are valid options from his perspective. In this paper, we introduce *explanations* for data points excluded by skyline queries. The user can then appraise whether the extra 10 euro per night and 7% from the beach is a tolerable trade-off and adjust his query accordingly, if he wants.

Figure 1 illustrates the technical problem. Given range constraints $(q_{\mathrm{L}}, q_{\mathrm{U}})$, one retrieves a skyline of just hotels A and B, yet C also satisfies the constraints. How does one explain to the user C's absence? In other words, why is C dominated? Similarly to other questions of *why-not provenance* [4, 11, 19, 24], we wish to report an alternative query (i.e., new constraints) so that C is no longer omitted from the results. We find a new position (one candidate is denoted by a star in the figure) for the lower constraints, q_{L}, so that all points dominating C are eliminated. This illuminates the relationship of point C to the rest of the data and, furthermore, concretely recommends to the user a potentially better-suited query.

However, finding the best explanation—the best new position for q_{L}—is non-trivial: anywhere within the rectangle

10.5441/002/edbt.2015.31

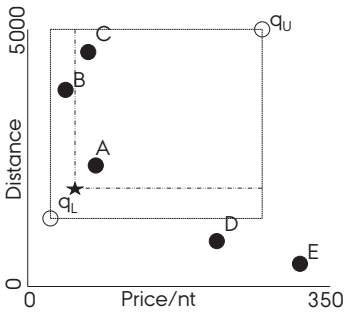


Figure 1: Hotels from Table 1 mapped to points in the plane, shown with a constrained region (q_L, q_U), enclosed by dashed lines. A Sky-not query for point C relocates q_L to a position, such as the star, after which C is in the skyline.

(q_L, C) could potentially improve C's query fate, since the only things that are obvious (later in this paper) is that q_L's old position should dominate its new position (so that every change is an improvement) and that the new position must still dominate C (so that C satisfies the constraints). The best position is the one closest to the original position but that promotes C to the skyline.

In this paper, we introduce and propose solutions for this novel query, the *Sky-not Query*, which quantifies *why* a query point p is *not* in a *sky*line. Algorithmically, we introduce a recursive algorithm that prioritizes cases that favour our pruning rules to efficiently relocate q_L. In comparison to a baseline algorithm and an adaptation of ideas from [12, 14, 17, 21], we show in a detailed experimental evaluation that the Sky-not query can be computed very efficiently.

Contributions and Outline

In this paper, we make the following contributions:

- we introduce the Sky-not Query to improve the useability of the skyline operator (Section 2);
- we derive theoretical properties of Sky-Not Queries, including duality with the SkyDist problem studied in [12,14], that lead to algorithm-independent improvements in efficiency (Section 4); and
- we give two novel algorithms, BRA and PrioReA, based on theoretical analysis (Section 5) for achieving impressive empirical performance (Section 6).

Additionally, we survey related literature in Section 3 and conclude in Section 7.

2. THE SKY-NOT QUERY

In this section, we introduce the novel *Sky-not query* for explaining the absence of a given point in a skyline result. We first recall some general concepts from literature about skylines (Definitions 1-4), before presenting the problem definition (Definition 5).

Generally, we assume a dataset of records represented by a set of Euclidean points, $\mathcal{S}$. For example, the hotels from Table 1 are represented as points in Figure 1. Given $\mathcal{S}$, we denote the dimensionality of the Euclidean space by d and the i'th attribute of a point $s \in \mathcal{S}$ by $s[i]$. Without loss of generality and for the sake of simplifying prose, we assume all attribute values are in the range $[0, M)$, for some positive real, M, and that smaller values are preferable.[1]

2.1 Constrained Skylines

To define the skyline formally, we first define *dominance* (Definition 1). One point *dominates* another if it is at least as desirable in every dimension, and strictly more desirable in at least one. Formally:

Definition 1 (Dominance [3])**.**
Point s *dominates* point t, denoted $s \prec t$, iff

$$\forall i \in [0, d), s[i] \leq t[i] \text{ and } \exists j \in [0, d) \text{ such that (s.t.) } s[i] < t[i].$$

If neither s dominates t nor t dominates s, then s and t are *incomparable*, denoted $s \prec\!\succ t$. The relation $s \preceq t$ denotes that either $s \prec t$ or $\forall i \in [0, d), s[i] = t[i]$.

For example, from Table 1, Hotel B dominates Hotel C because it is both cheaper and nearer (B is to the lower-left of C in Figure 1). A stronger case is strict dominance (Definition 2), in which strict equalities are used:

Definition 2 (Strict dominance)**.**
Point p *strictly dominates* point q, denoted $p \prec\!\!\prec q$, iff $\forall i \in [0, d), p[i] < q[i]$.

Given range constraints, where q_L denotes the lower bound (0 if unspecified) on every attribute and q_U denotes the upper bound on every attribute (M if unspecified), we call the subset of points satisfying the constraints the *constrained dataset* (Definition 3):

Definition 3 (Constrained Dataset)**.**
Given a set of points, $\mathcal{S}$ and a constraint region (q_L, q_U), the *constrained dataset*, denoted $\mathcal{S}_{(q_L, q_U)}$, is the set of points $\{s \in \mathcal{S} : q_L \prec\!\!\prec s \prec\!\!\prec q_U\}$. We say that each point $s \in \mathcal{S}_{(q_L, q_U)}$ is *inside* the constraint region.

For example, given the constraints in Figure 1, only A, B, C are in the constraint region. The skyline (Definition 4) is the set of points inside the constraint region that are not dominated by any other points inside the constraint region:

Definition 4 ((Constrained) Skyline Query($\mathcal{S}$) [3])**.**
Given a set of points, $\mathcal{S}$, and a constraint region, (q_L, q_U), a *skyline query* returns the set:

$$\text{SKY}(\mathcal{S}, (q_L, q_U)) = \{s \in \mathcal{S}_{(q_L, q_U)} : \nexists t \in \mathcal{S}_{(q_L, q_U)}, t \prec s\}.$$

The set $\text{SKY}(\mathcal{S}, (q_L, q_U))$ is called the *skyline* of $\mathcal{S}_{(q_L, q_U)}$.

For example, in Table 1, with $q_L = (0, 0)$ and $q_U = (350, 5000)$, Hotel C is dominated by Hotel B, but all other hotels are incomparable to each other. Therefore, the skyline is $\{A, B, D, E\}$. On the other hand, with constraints $q_L = (35, 1000)$ and $q_U = (250, 5000)$, as shown with the outer, dotted rectangle in Figure 1, only A and C satisfy the constraints and neither dominate each other; so, they are both in the skyline: $\{A, C\}$.

2.2 Problem Definition

We now define the *Sky-not query* (Definition 5). Informally, given a constrained skyline instance and a query point p, a Sky-not query, denoted $\text{SN}(\mathcal{S}, p, (q_L, q_U), \Delta)$ determines the minimum cost change to q_L after which p would be in the skyline. Formally:

[1]The first assumption is justified by normalization and the second assumption can hold by multiplying values by -1 where maximization is instead preferred.

Definition 5 (Sky-not Query, $\mathrm{SN}(\mathcal{S}, p, (q_\mathrm{L}, q_\mathrm{U}), \Delta)$)**.** Given $\mathcal{S}, p \in \mathcal{S}, (q_\mathrm{L}, q_\mathrm{U})$, s.t. $q_\mathrm{L} \prec\!\!\prec p \prec\!\!\prec q_\mathrm{U}$, distance function, $\Delta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$, [2]a *Sky-not Query* returns point:

$$q'_\mathrm{L} = \mathrm{argmin}_{q \in (q_\mathrm{L}, q_\mathrm{U}), p \in \mathrm{SKY}(\mathcal{S}, (q'_\mathrm{L}, q_\mathrm{U}))} \Delta(q_\mathrm{L}, q).$$

Again considering Figure 1, we could move q_L to the starred position, and then C is in the skyline (i.e., the starred position *solves* the Sky-not query). However, this is not the optimal solution: $q'_\mathrm{L} = (30, 1000)$ also solves the Sky-not query, but is closer to q_L than the starred position.

By learning the Sky-not query result, q'_L, a user can immediately understand how competitive p is relative to the skyline and evolve his subsequent queries accordingly.

3. RELATED WORK

The Sky-not Query (Section 2) ties together a couple of active research topics, which we survey in this section.

Skyline The skyline operator was introduced [3] with two disk-based algorithms, a block-nested loop (BNL) and a partitioning-based (D&C) approach. Since, BNL has been improved with the use of pre-sorting [8] and early termination conditions [1]. The D&C algorithm has been improved with progressively better partitioning schemes [15, 27]. Also, new index-based algorithms based on B-Trees [23] and R-Trees [20] have been proposed. Of these algorithms, BSkyTree [15] reports the best performance; although, this can be improved using multiple processing cores [7].

Although the skyline was proposed in the context of a general SQL query with a `WHERE` clause [3], Papadias et al. [20] were the first to propose algorithms specifically for handling constraints, a problem they term the *constrained skyline* and the subject of study here. The presence of constraints makes the skyline operator much more flexible and practical, because most real queries involve a `WHERE` clause.

Although the skyline is considered as a tool for interactivity, say in discovering user preferences, little research has considered this aspect. Literature on "interactive skylines" (e.g., [16]) seeks to learn user preferences. Our perspective on interactivity here is in helping the user understand *the data*, not their preferences over the attribute domains. To this end, existing work is quite limited. Magnani et al. [18] conducted a user study that showed users are perplexed when posing consecutive queries if new dominance relationships cause previous query results to disappear. Chester et al. [6] conducted an experimental study of how much the skyline changes when users evolve their constraints. Both these works assess the impact on the user of interacting with the skyline, but neither provide solutions for helping the user to evolve queries towards his/her objective.

Why-not Queries Providing the user with an explanation for a missing answer [4,11,19] is a relatively new goal in database research. The general idea of *why-not queries* is to provide a user, who has a specific solution record in mind, with specific details into the cause for its being omitted from the results. Why-not queries have been studied for relational (i.e., SPJUA, sort-project-join-union-aggregate) queries [2, 10], top-k queries [9], and reverse skyline queries [13],[3] but the definitions and techniques do not straight-forwardly apply for skyline queries because each query type excludes candidate results for quite different reasons (no common join key, low weighted sum, & dominance).

However, we do adapt two key ideas from other why-not query types. Because explanations can be arbitrarily complex, it is preferable to determine minimal explanations [28]. Our experiments (Section 6) reveal that Sky-not queries favour explanations involving fewer dimensions changing. Additionally, there are generally two ways of manipulating a query result [24]: either by changing the data or the query parametres. Here, we focus on changing the query, since this is more often under the user's control. With respect to changing the data, some research has gone into manipulating skyline query results [12,14,17,21] from a Business Intelligence perspective, where the objective is to modify one's product offerings in order to penetrate the skyline. We elaborate on the relationship between changing the query and the data in Theorem 5. Lastly, our problem differs from creating competitive products [25]: we aim to determine why *an existing* product is not in the skyline, not to create new (meta-)products from a *collection* of existing products.

Technically related papers We overlap some technical material from tangentially related papers. Our algorithms first find all the points that cause the absence of p in the skyline, then relocate q_L. To detect the causative points, we borrow the idea of *close dominance* [22] (Definition 6). Regarding relocation, Cheema et al. [5] introduce the idea of "safe zones" for dynamic skylines, wherein the safe zone of a point is those query positions in which the point is still a skyline point. Our objective is to find the nearest query point where all safe zones are violated. DeltaSky [26] maintains a view of skyline points to handle deletions.

4. PROPERTIES OF THE SKY-NOT QUERY

In this section, we introduce some algorithm-independent theoretical insights into the Sky-not query. In particular, we show both that the solution space (q_L, p) can be discretized and that the input size can be reduced (Section 4.1). This gives a smaller, finite search space for the problem. We also show duality of the problems of changing the data vs. the query to manipulate a skyline result (Section 4.2). Consequently, techniques for both problems are mutually exchangeable (and, indeed, we do exactly this in our experimental evaluation, Section 6).

4.1 Reduction and Discretization

We start with a couple properties of a Sky-not query solution that are quickly evident from the definitions.

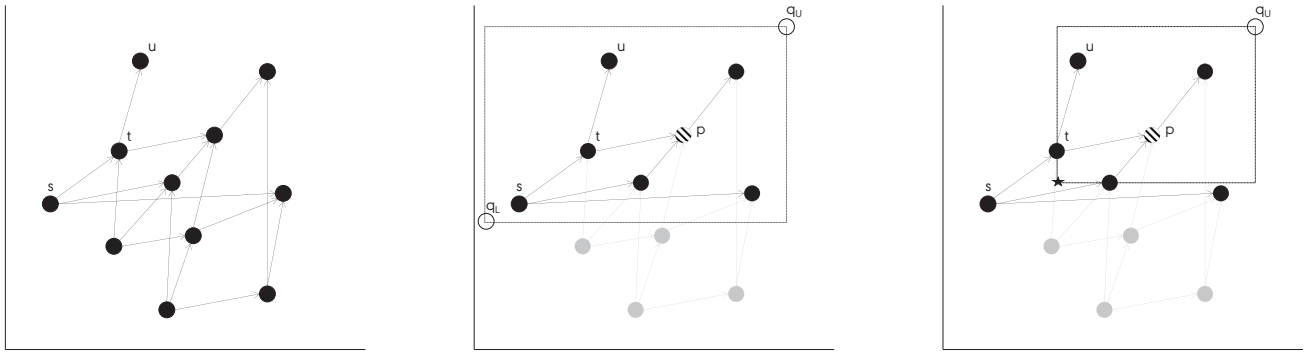
Proposition 1 (Transitive Order)**.** $q_\mathrm{L} \prec\!\!\prec q'_\mathrm{L} \prec\!\!\prec p \prec\!\!\prec q_\mathrm{U}$.

Proposition 2 (Undominated Data)**.** $\forall s \in \mathcal{S}_{(q_\mathrm{L}, q_\mathrm{U})}, s \prec p \implies q'_\mathrm{L} \not\prec s$.

Proposition 1 simply states that the solution q'_L must necessarily increase or keep the values of q_L and must strictly dominate neither p nor q_U; otherwise, p will not be in the skyline. Proposition 2 simply states that the solution q'_L

[2]Throughout this paper, we assume the distance function is L_1 (i.e., Manhattan) norm: $\Delta(s,t) = \sum_{i=0}^{d-1} |s[i] - t[i]|$; so, we will drop Δ from the notation. The ideas presented easily generalize to any weighted L_p norm.

[3]Despite the similar name, a reverse (dynamic) skyline query is quite different from a skyline query, focusing on dynamic, spatial proximity and an inversion of the skyline problem.



(a) Points in the plane with close dominance relationships indicated by arrows

(b) In this Sky-not query, p is dominated by three points in the constraint region

(c) By solving the points that closely dominate p, s is also solved transitively

Figure 2: An example of *close dominance* and how it relates to Sky-not queries

must be placed such that it does not dominate any point s that dominates p; otherwise, s will still satisfy the constraints, still dominating p, and thus p will not be in the skyline. Together, these propositions suggest that the solution will lie somewhere inside (q_{L}, p).

This hyper-rectangular solution space contains infinitely many points. Lemma 3 discretizes the search space. Specifically, it states that the value of q'_{L} on each dimension i is either the same as the original lower constraint, $q_{\mathrm{L}}[i]$ or as one of the data points, $s[i], s \in \mathcal{S}$. Thus, there is only (an exponential number of) finite possible solutions.

Lemma 3 (Sky-not Discretization)**.**
Let $q'_{\mathrm{L}} = \mathrm{SN}(\mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})}, p, (q_{\mathrm{L}}, q_{\mathrm{U}}))$. *Then,*

$$\forall i \in [0,d) \exists s \in \left(\mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})} \bigcup \{q_{\mathrm{L}}\}\right) \text{ s.t. } q'_{\mathrm{L}}[i] = s[i].$$

The basic idea of the proof is that if there were a solution that was not composed of existing values, then we could find a better one that was.

Proof. We prove this by contradiction. Let q'_{L} be a solution such that $\exists j \in [0,d) \forall s \in \left(\mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})} \bigcup \{q_{\mathrm{L}}\}\right), q'_{\mathrm{L}}[i] \neq s[i]$. Let $\mathcal{S}_<$ be the points $s \in \mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})} \cup \{q_{\mathrm{L}}\}$ with $s[j] < q'_{\mathrm{L}}[j]$. Let $\overline{s}$ be the point in $\mathcal{S}_<$ with the highest $s[j]$ value. Then, construct a point q''_{L} such that $q''_{\mathrm{L}}[i] = q'_{\mathrm{L}}[i]$ if $i = j$ and $q''_{\mathrm{L}}[j] = \overline{s}[j]$. Then, q''_{L} is closer to q_{L} than is q'_{L}. Moreover, $\forall s \notin \mathcal{S}_<, \exists i \neq j$ s.t. $s[i] < q'_{\mathrm{L}}[i] = q''_{\mathrm{L}}[i]$, because $q'_{\mathrm{L}}[i] \not\prec s$ and $q'_{\mathrm{L}}[j] < s[j]$. Also, $\forall s \in \mathcal{S}_<$, it is still the case that $q''_{\mathrm{L}} \not\prec s$, because $s[j] \leq q''_{\mathrm{L}}[j]$. Lastly, $q''_{\mathrm{L}} \in (q_{\mathrm{L}}, q_{\mathrm{U}})$, since $q'_{\mathrm{L}} \in (q_{\mathrm{L}}, q_{\mathrm{U}})$, and they differ only on the j'th attribute, for which $q''_{\mathrm{L}}[j] = q_{\mathrm{L}}[j]$. Therefore, $\forall s \in \mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})}, q''_{\mathrm{L}}[s] \not\prec s$ and $p \in \mathrm{SKY}(\mathcal{S}, [q''_{\mathrm{L}}, q_{\mathrm{U}}])$. □

Lemma 3 discretized the solution space based on the points $s \in \mathcal{S}$. Lemma 4 reduces the solution space even further by showing that only some of the points that dominate p also influence the solution. For this lemma, we borrow the concept of *close dominance* (Definition 6) from literature: if s dominates t, then it also closely dominates t if there is no other point "in between them":

Definition 6 (Close dominance [22])**.**
Given points $s, t \in \mathcal{S}$, we say that s *closely dominates* t, denoted $s \preccurlyeq t$, iff $s \preceq t \wedge \nexists u \in \mathcal{S} : s \prec u \preceq t$.

Figure 2a illustrates the close dominance relationship. Notice that, although $s \prec\!\!\!\cdot\, u$, $s \not\prec\!\!\!\cdot\, u$, because the dominance can be ascertained transitively through t. Lemma 4 states that it is only the values of points that closely dominate p that might be used in the Sky-not query solution. In Figure 2b, s need not be considered, because $s \not\prec\!\!\!\cdot\, p$.

Lemma 4 (Close dominance is sufficient)**.**
Let $C = \{s \in \mathcal{S} : s \prec\!\!\!\cdot\, p\}$ *be the set of points in* $\mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})}$ *that closely dominate* p. *Then,*

$$\mathrm{SN}(C, p, (q_{\mathrm{L}}, q_{\mathrm{U}})) = \mathrm{SN}(\mathcal{S}, p, (q_{\mathrm{L}}, q_{\mathrm{U}})).$$

The basic idea behind the proof is that by handling all the points in C, one handles all points in $\mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})}$ by means of transitivity.

Proof. We show that finding a point q'_{L} that does not dominate any point in C is both necessary and sufficient for upgrading p to the skyline.

Necessary: Note that $t \in C \implies t \in \mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})} \wedge t \prec p$, by construction. So, by Proposition 2, $q'_{\mathrm{L}} \not\prec t$ (i.e., they necessarily fall outside the constraints).

Sufficient: Consider any point $s \in C \setminus \mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})}$. Then, $\exists t \in C$ s.t. $\forall i \in [0,d), s[i] \leq t[i]$, because $s \preceq t$. But, because $t \in C \implies q'_{\mathrm{L}} \not\prec t$, then $\exists j \in [0,d)$ s.t. $t[i] < q'_{\mathrm{L}}[i]$. By transitivity, $s[i] < q'_{\mathrm{L}}[i]$; so, $q'_{\mathrm{L}} \not\prec s$. □

The algorithmic consequence of Lemma 4 is that we can first find the much smaller set C and then execute whichever algorithm on C instead of $\mathcal{S}_{(q_{\mathrm{L}},q_{\mathrm{U}})}$ for an immediate improvement in efficiency. In Figure 2c, it is sufficient to solve the two points that closely dominate p, because solving t transitively implies that s is also solved.

4.2 Duality

Finally, we show an interesting result, that the Sky-not query is a dual form of a generalization of the SkyDist problem (Definition 7) in literature. The SkyDist problem is to find the cheapest way to modify p so that it will be in the skyline. That is to say, the SkyDist problem changes the data, rather than the query, to upgrade p. We redefine it below with the addition of constraints, which are needed for the duality result.

Definition 7 (Skyline Distance Problem [14])**.**
Given $\mathcal{S}, p, (q_{\text{L}}, q_{\text{U}})$, find the point $p' \in (q_{\text{L}}, q_{\text{U}})$ closest to p such that $p' \in \text{SKY}(\mathcal{S} \cup \{p'\}, (q_{\text{L}}, q_{\text{U}}))$.

We show that the SkyDist and Sky-not queries are dual problems of each other. First, define a transform function $f(x) = M - x$, overloaded for points and sets, $f(p) = \hat{p}$: $\hat{p}[i] = f(p[i])$ and $f(\mathcal{S}) = \{f(s) : s \in \mathcal{S}\}$ by applying the function to each value of a point and each point of a set. Let $p' = \text{SD}(\mathcal{S}, p, (q_{\text{L}}, q_{\text{U}}))$ be an instance and solution to a SkyDist problem. Then we have Theorem 5, that the Sky-not query result on the transformed data is exactly the transformation of the SkyDist result on the original data:

Theorem 5 (SkyDist-SkyNot Duality)**.**
$\text{SN}(f(\mathcal{S}), f(p), [f(q_{\text{U}}), f(q_{\text{L}})]) = f(\text{SD}(\mathcal{S}, p, (q_{\text{L}}, q_{\text{U}})))$.

Proof. To show that the solutions are equivalent, we show first that distance is preserved by the transformation function; so that the optimality of any solution is consistent. We then show that the transform of a point that solves the Sky-Not instance solves the SkyDist instance, and vice versa.

Distance-preserving: Let s, t be points. Then:

$$\begin{aligned}\Delta(s,t) &= \sum_{i=0}^{d-1} |s[i] - t[i]| = \sum_{i=0}^{d-1} |(M - s[i]) - (M - t[i])| \\ &= \sum_{i=0}^{d-1} |f(s[i]) - f(t[i])| = \Delta(f(s), f(t)).\end{aligned}$$

This holds in the opposite direction, because $f(f(s)) = s$.

Mutually solving:

$$\begin{aligned} q &= \text{SN}(f(\mathcal{S}), f(p), (f(q_{\text{U}}), f(q_{\text{L}}))) \\ &\implies f(q_{\text{U}}) \prec q \prec f(q_{\text{L}}), \forall s \in f(\mathcal{S}), q \not\prec s \\ &\implies \exists i \in [0, d) : s[i] < q[i] \\ &\implies \exists i \in [0, d) : f(q[i]) < f(s[i]) \implies f(s) \not\prec f(q) \\ &\implies q_{\text{L}} \prec f(q) \prec q_{\text{U}}, \forall s \in \mathcal{S}, s \not\prec f(q). \\ f(p') &= f(\text{SD}(\mathcal{S}, p, (q_{\text{L}}, q_{\text{U}}))) \implies \forall s \in \mathcal{S}, s \not\prec p' \\ &\implies f(p') \not\prec f(s) \implies \forall s \in f(\mathcal{S}), f(p') \not\prec s. \end{aligned}$$

□

In fact, the above holds for any order-inverting and distance-preserving transformation function.

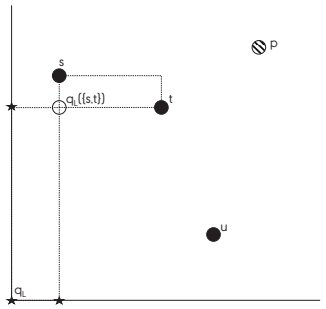
The signficance of this result is that techniques developed for SkyDist and Sky-not queries are mutually exchangeable by applying the transform. However, the addition of constraints enables new analytical approaches that lead to more efficient computation, as we will show in our experimental evaluation, where we apply the duality transform to existing SkyDist techniques (Section 6).

5. ANSWERING SKY-NOT QUERIES

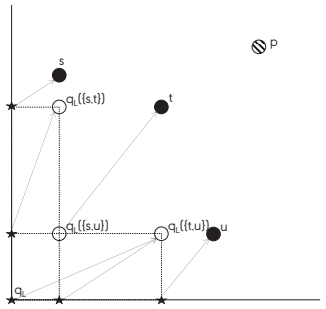
In this section, we present two algorithms for solving Sky-not queries, based on the insight from the previous section.

5.1 Bounding Rectangle Algorithm

Here, we introduce the Bounding Rectangle Algorithm (BRA). We begin with theoretical analysis to deduce a finite set of candidate positions to which q_{L} can be moved in order to position p in the skyline (Section 5.1.1). We then build an algorithm to efficiently calculate those positions and find the optimal solution from within them (Section 5.1.2).



(a) The creation of $q_{\text{L}}(\{s,t\})$ with the minimum value from $\{s,t\}$ on each attribute and of rectangle $(q_{\text{L}}, q_{\text{L}}(\{s,t\}))$. Stars represent the 2^d projections of the rectangle.



(b) All three possible combinations of two points are shown with their resultant rectangles. Arrows represent strict dominance relationships, the origin of which are points that are not valid solutions.

Figure 3: Bounding rectangles and their application in BRA.

5.1.1 Bounding Rectangles and Candidate Positions

Recall from Lemma 3 that solutions will take values from points in the dataset and from Lemma 4 that the set C contains all the points that need to be considered. We formalize that set with Corollary 6 below. We then show that this set can be further pruned by taking into account how these points relate to each other (Theorem 7).

First, however, we need to introduce a little more notation for use in this subsection. Let $\mathcal{P}_{\leq d}(C)$ be the subsets of C with at most d elements, and let $T \in \mathcal{P}_{\leq d}C$ be such a subset. Furthermore, let $q_{\text{L}}(T)$ be the corner of the minimum bounding rectangle of T that is closest to q_{L} (the bottom left corner in two dimensions). Notice that $\text{R}_T = (q_{\text{L}}, q_{\text{L}}(T))$ is itself a (possibly degenerate) hyper-rectangle. Finally, denote by $\llcorner(\text{R}_T)$ the 2^d corners of $(q_{\text{L}}, q_{\text{L}}(T))$.

Recall that a *minimum bounding rectangle* of a set of points T is the unique smallest hyper-rectangle that contains all points in T. The lowermost corner (i.e., the one closest to q_{L}) is the one containing the smallest value on each attribute of any point in T. It represents the best possible combination of values from points in T. Sets with more than d points are not interesting, because we only combine values for up to d dimensions.

Figure 3a illustrates these concepts. The set $C = \{s, t, u\}$ has $\mathcal{P}_{\leq 2}C = \{\{\}, \{s\}, \{t\}, \{u\}, \{s,t\}, \{s,u\}, \{t,u\}\}$. Let $T = \{s,t\}$. The minimum bounding rectangle of T is the dashed rectangle to the top-right in the figure, and its $q_{\text{L}}(T)$ is shown as a hollow circle. The dashed rectangle to the lower-left is $(q_{\text{L}}, q_{\text{L}}(T))$. The 2^d corners are marked with stars.

Each of the 2^d corners of hyper-rectangle $(q_\text{L}, q_\text{L}(T))$ has a unique set of attributes with values equal to those of q_L and the others equal to those of $q_\text{L}(T)$. They correspond to solutions with fewer than all values changed. So, Corollary 6 states that if we take all subsets T of up to d points from C and consider all the corners on the hyper-rectangle traced from q_L to $q_\text{L}(T)$, we will include all possible combinations of all points in C, and therefore also have the optimal solution somewhere in the (finite) set.

Corollary 6 (BRA search space)**.**
$q'_\text{L} = \text{SN}(\mathcal{S}, p, (q_\text{L}, q_\text{U})) \in \bigcup_{T \in \mathcal{P}_{\leq d}(C)} \llcorner(\text{R}_T)$.

Proof. This follows directly from Lemmata 3 and 6, because all possible combinations of all points in C are contained in $\bigcup_{T \in \mathcal{P}_{\leq d}(C)} \llcorner(\text{R}_T)$. □

Corollary 6 includes all possible solution points, but also many more. Theorem 7 gives the key idea for the BRA, that it is only those points in the set that do not dominate any others in the set that are possibilities. The property of not strictly dominating any other corner points is both necessary and sufficient for indicating that any given corner point correctly positions p in the skyline.

Theorem 7 (Central BRA postulate)**.**
q solves $\text{SN}(\mathcal{S}, p, (q_\text{L}, q_\text{U}))$ *iff* $\forall \llcorner \in \bigcup_{T \in \mathcal{P}_{\leq d}(C)} \llcorner(\text{R}_T), q \not\prec\!\!\prec \llcorner$.

The basic idea behind the proof is that it is certainly sufficient, because every point $c \in C$ is a corner point for some rectangle; so, not dominating any corner points for any rectangles implies that no point in c is dominated either. It is necessary because any dominated point will produce some rectangle with a corner point that is also dominated.

Proof. **Sufficient**:

$$\begin{aligned} & \forall \llcorner \in \bigcup_{T \in \mathcal{P}_{\leq d}(C)} \llcorner(\text{R}_T), q \not\prec\!\!\prec \llcorner \\ \implies \quad & \forall \llcorner \in \bigcup_{T \in \mathcal{P}_{\leq 1}(C)} \llcorner(\text{R}_T), q \not\prec\!\!\prec \llcorner \\ \implies \quad & \forall \llcorner \in \bigcup_{t \in C} \llcorner([q_\text{L}, t]), q \not\prec\!\!\prec \llcorner \implies \forall t \in C, q \not\prec\!\!\prec t. \end{aligned}$$

Necessary: Assume for the sake of contradiction that $\exists \llcorner \in \bigcup_{T \in \mathcal{P}_{\leq d}(C)} \llcorner(\text{R}_T)$ s.t. $q \prec\!\!\prec \llcorner$. Let T denote the subset of C that produced the hyper-rectange with that corner. Then, $\exists t \in T : q \prec\!\!\prec t \implies \exists s \in C : q \prec\!\!\prec s$ and, thus, q cannot be a valid solution of $\text{SN}(\mathcal{S}, p, (q_\text{L}, q_\text{U}))$. □

The concept behind Theorem 7 is illustrated in Figure 3b. Three of the hyper-rectangles are shown, along with all their corner points. Arrows depict that the originating corner strictly dominates the destination corner. One can verify in the figure that all positions with arrows dominate some point in C, whereas all positions without arrows (namely $q_\text{L}(\{s,t\})$ and $q_\text{L}(\{t,u\})$) do not.

5.1.2 Algorithm description

Theorem 7 suggests an elegant way to adapt existing skyline algorithms to solve a Sky-not query, since the candidate solutions are the points in $X = \bigcup_{T \in \mathcal{P}_{\leq d}(C)} \llcorner(\text{R}_T)$, and the optimal solution is the point $x \in X$ closest to q_L. So, we adapt the Block-Nested-Loop (BNL) [3] skyline algorithm, using a window sorted by proximity to q_L, to produce the set X. The head of the window once all points have been processed is the optimal solution, since Corollary 6 states that all possible solutions are in that set.

Algorithm 1 Bounding Rectangle Algorithm (BRA)

Input: $\mathcal{S}, p, q_\text{L}, q_\text{U}$
Output: q'_L, the optimal Sky-not solution on $(\mathcal{S}, p, (q_\text{L}, q_\text{U}))$

```
 1: Create empty set C
 2: for all s ∈ S do
 3:   if q_L ⪯ s ⪯ p ⪯ q_U then
 4:     for all c ∈ C do
 5:       if s ≺ c then
 6:         add← false; break
 7:       else if c ≺ s then
 8:         C ← C \ {c}
 9:     if add then
10:       C ← C ⋃ {s}
11: Create queue W sorted by ascending proximity to q_L
12: for all ⌞ ∈ ⋃_{T∈P_{≤d}(C)} ⌞(R_T) do
13:   for all p ∈ W do
14:     if ⌞ ≺≺ p then
15:       add←false; break
16:     else if p ≺≺ ⌞ then
17:       W ← W \ {p}
18:   if add then
19:     W ← W ⋃ {⌞}
20: Return W[0]
```

Specifically, Algorithm 1 first computes the set C (Lines $1-10$) and then iterates through all corner points to find those that are valid solutions (Lines $11-19$). By storing the valid solutions in a queue that is sorted by proximity to q_L, the head of the list is the optimal solution (Line 20).

Both steps follow the same control flow. We go through every point $s, \llcorner$ and compare it to every other point c, p in our current solution set. If s is dominated by c, then c is removed from the current solution set. If s does not dominate any c, then it is added. The difference with $\llcorner, p$ is that we require strict dominance.

The running time of the algorithm thus depends on the maximum size that the window W becomes. We know the final solution is correct and optimal on account of Theorem 7.

5.2 Prioritized Recursion Algorithm

In this section, we introduce our Prioritized Recursion Algorithm (PrioReA), which uses a few theoretical conclusions to discover good solutions very quickly, and use it to prune the majority of the search space.

5.2.1 Algorithmic Foundations

The objective in this section is to build towards a sound recursive formulation of the problem and a series of theoretical pruning rules that can be used to limit the search space. We can then describe a recursive algorithm, based on that formulation, which uses the pruning rules to pursue the optimal solution dramatically faster.

We begin with two lemmata that give rise to the pruning rules and recursion. First, we note that the problem is monotonic in the sense that solutions on subsets of points are necessarily at least as good as those on supersets. Specifically, Lemma 8 states that if one adds points to $\mathcal{S}$, the cost

of the solution cannot decrease. The key observation is that adding more points to an input set forces a solution to be farther from q_{L} if it is not to strictly dominate any of the points in either the subset nor the superset.

Lemma 8 (Monotonically increasing cost)**.**
Let $q'_{\mathrm{L}} = \mathrm{SN}(\mathcal{S}, p, (q_{\mathrm{L}}, q_{\mathrm{U}}))$ *and* $q''_{\mathrm{L}} = \mathrm{SN}(\mathcal{S}', p, (q_{\mathrm{L}}, q_{\mathrm{U}}))$. *Then,* $\mathcal{S} \subset \mathcal{S}' \implies \Delta(q_{\mathrm{L}}, q'_{\mathrm{L}}) \leq \Delta(q_{\mathrm{L}}, q''_{\mathrm{L}})$.

Proof. q'_{L} is the closest point to q_{L} that dominates all points in $\mathcal{S}$. q''_{L} must also dominate all points in $\mathcal{S}$, since $\mathcal{S} \subset \mathcal{S}'$; so, it cannot be closer to q_{L} than q'_{L} is. □

The second lemma defines a lower bound on the solution cost. Specifically, Lemma 9 states that, if one discovers the point s whose smallest attribute value (relative to q_{L}'s) is farthest from q_{L}'s value on that attribute, this difference lower bounds the cost of the optimal solution. Here, the observation is that this value is the smallest value that could conceivably guarantee that no point is strictly dominated.

Lemma 9 (Lower bound on solution cost)**.**
$\max_{s \in \mathcal{S}} \min_{i \in [0,d)} (s[i] - q_{\mathrm{L}}[i]) \leq \mathrm{SN}(\mathcal{S}, p, (q_{\mathrm{L}}, q_{\mathrm{U}}))$.

Proof. Since $\forall s \in \mathcal{S}, q'_{\mathrm{L}} \not\prec s$, q'_{L} must be larger or equal to at least one value of every point. □

Lemma 8 is valuable for pruning the search space, because it indicates that if we find a recursive call to be unpromising, then all recursive calls using a superset of those points will be likewise unpromising. Lemma 9 is useful because it allows estimating the optimal solution that will be returned by a recursive call without actually calculating it. Thus, we can often use the estimate to determine that a recursive call cannot possibly produce a better solution than what we have already seen.

This brings us to the main theorem for this section, which formulates the problem recursively. Specifically, Theorem 10 states that if we partition a set of points C based on whether they have a value $> x$ on dimension i, then the solution q'_{L} with $q'_{\mathrm{L}}[i] = x$ that is closest to q_{L} is exactly the difference between $q'_{\mathrm{L}}[i]$ and $q_{\mathrm{L}}[i]$ plus the cost of the best possible solution with $q'_{\mathrm{L}}[i] = q_{\mathrm{L}}[i]$ on the higher-valued partition.

Theorem 10 (Basis for recursion)**.**
Let $C_{i>x} = \{s \in C : s[i] > x\}$ *and* $Q_{i>x} = \{q \in [q_{\mathrm{L}}, p] : \forall s \in C_{i>x}, q \not\prec s\}$. *Then, for* $x \geq q_{\mathrm{L}}[i]$*:*

$$\begin{aligned} & \Delta(q_{\mathrm{L}}, \mathit{argmin}_{q \in Q_{i>q_{\mathrm{L}}[i]} : q[i]=x} \Delta(q_{\mathrm{L}}, q)) \\ = \; & \Delta(q_{\mathrm{L}}, \mathit{argmin}_{q' \in Q_{i>x} : q'[i]=q_{\mathrm{L}}[i]} \Delta(q_{\mathrm{L}}, q')) + (x - q_{\mathrm{L}}[i]). \end{aligned}$$

Proof. Note that for a point q with $q[i] = x$, $\nexists s \in C$ with $s[i] \leq x$ such that $q \prec s$. All other points, those in $C_{i>x}$, must have a lower value than q on some other attribute; i.e., must be dominated by the projection of q onto the point q' where $q'[i] = q_{\mathrm{L}}[i]$ (i.e., is not changed from the original constraint). So, $\Delta(q_{\mathrm{L}}, q)$ is exactly the distance of the projection q' from q_{L} plus the distance from q to q', since all distances are positive in all directions. □

Finally, we note the three pruning rules that are straightforward consequences of the earlier lemmata in this section. In particular, Corollary 11 states that if we have already seen a point whose cost (i.e., distance to q_{L}) is less than the sum of the distance of a given point from q_{L} and the lower bound on the recursive call, then, independent of the recursive call, the given point cannot produce a better solution than what has already been seen. This comes directly from Lemma 9 and Theorem 10.

Algorithm 2 Prioritized Recursion Algorithm (PrioReA)

Input: $C, p, q_{\mathrm{L}}, \mathcal{D}$
Output: q'_{L}, the optimal Sky-not solution on $(\mathcal{S}, p, (q_{\mathrm{L}}, q_{\mathrm{U}}))$

```
1: if C = ∅ then
2:    Return q_L
3: best← p
4: Sort d ∈ D by p[d] − q_L[d], ascending
5: for all d ∈ D do
6:    Sort s ∈ S by s[d] − q_L[d], descending
7:    maxmin ← 0
8:    for all s ∈ C do
9:       if s[d] − q_L[d] + maxmin < Δ(q_L, best) then
10:         rec←PrioReA(C_[0,...,s−1], D \ {d}, q_L, p)
11:         if Δ(q_L, rec) + s[d] − q_L[d] < Δ(q_L, best) then
12:            best←rec, best[d] ← s[d]
13:         else if Δ(q_L, rec) ≥ Δ(q_L, best) then
14:            Break {Pruning rule (2).}
15:      else if maxmin ≥best then
16:         Break {Pruning rule (2').}
17:      else
18:         Do nothing {Pruning rule (1).}
19:      if maxmin < min_{d'∈D} s[d'] − q_L[d'] then
20:         maxmin ← min_{d'∈D} s[d'] − q_L[d']
21: Return best
```

Corollary 11 (Pruning Rule (1))**.**
If $\exists q \in (q_{\mathrm{L}}, p), i \in [0, d), x \geq q_{\mathrm{L}}[i]$ *s.t.* $\Delta(q_{\mathrm{L}}, q) \leq (x - q_{\mathrm{L}}[i]) + \max_{s \in C_{i>x}} \min_{i \in [0,d)} (s[i] - q_{\mathrm{L}}[i]))$ *then* $\forall q' \in (q_{\mathrm{L}}, p) : q'[i] = x, \Delta(q_{\mathrm{L}}, q) \leq \Delta(q_{\mathrm{L}}, q')$.

The second pruning rule, Corollary 12, states that if we have already seen a point whose cost is less than the cost of the best solution returned by a recursive call on C', then we need never consider any recursive calls on supersets of C'. This comes directly from Lemma 8.

Corollary 12 (Pruning Rule (2))**.**
If $\exists q \in (q_{\mathrm{L}}, p), i \in [0, d), x \geq q_{\mathrm{L}}[i]$ *s.t.* $\Delta(q_{\mathrm{L}}, q) \leq \Delta(q_{\mathrm{L}}, \mathit{argmin}_{q' \in Q_{i>x} : q'[i]=q_{\mathrm{L}}[i]} \Delta(q_{\mathrm{L}}, q'))$ *then* $\forall x' \leq x, \Delta(q_{\mathrm{L}}, q) \leq \Delta(q_{\mathrm{L}}, \mathit{argmin}_{q' \in Q_{i>x'} : q'[i]=q_{\mathrm{L}}[i]} \Delta(q_{\mathrm{L}}, q'))$.

Finally, a variant of the second pruning rule, Corollary 13, states that if we have already seen a point with cost less than the lower bound estimate of the recursive call, we can also safely dismiss all recursive calls on supersets.

Corollary 13 (Pruning Rule (2'))**.**
If $\exists q$ *s.t.* $\Delta(q_{\mathrm{L}}, q) < \max_{s \in C} \min_{i \in [0,d)} (s[i] - q_{\mathrm{L}}[i])$ *then,* $\forall C', C \subseteq C'$, *q is the best possible solution.*

5.2.2 Algorithm Description

Algorithm 2 describes the PrioReA algorithm. At a high level, it is a recursive algorithm, based on Theorem 10, that selects a dimension d and fixes a value x for that dimension. Any point s with $s[d] \leq x$ is clearly not strictly dominated. With all the other points $s' \in C, s'[d] > x$, and the remaining dimensions $d' \neq d$, we recurse to find an optimal solution. The combination of the result from the recursion with the

value x on dimension d is $\text{argmin}_{q' \in Q: q'[i]=x} \Delta(q_L, q')$, the best possible solution with value x on dimension d.

To set the value x on a recursive call with points C', we sort the points in $c \in C'$ by descending $c[d]$ value (Line 6) and iterate the sorted list (Line 8). We know from the discretization lemma, Lemma 3, that these are the only values that need to be considered. By iterating in descending order, we prioritize recursive calls with smaller inputs: it is the set of all points preceding the current one that still need to be resolved on the recursive call.

If we first compute C as in Lines $1-10$ of Algorithm 1, and then we iterate through all dimensions (Line 5) and for all possible values to which each of those dimensions could be set (Line 8), we are guaranteed to find the optimal solution, because we will have covered the entire search space of BRA that was defined in Corollary 6, with performance $\mathcal{O}(n^d)$.

However, PrioReA uses Corollaries 11-13 to avoid the majority of that search space. We maintain track of the *best* solution globally seen, the point q in the corollaries. Then, whenever the conditions of the corollaries are met (Lines 13, 15, or 17), we can avoid the recursive call (Line 18) or even break the loop entirely (Lines 14 or 16).

Furthermore, we prioritize the recursive calls exactly to push the *best* seen point, q, as close to q_L as early as possible. Specifically, we always choose next the dimension wherein p is closest to q_L, because this increases the likelihood of finding a good value x on that dimension that is also close to q_L. We always choose points closest to p first, rather than q_L, so that the recursive calls have fewer points and we need to change fewer dimensions to reach a solution.

The effectiveness both of the pruning rules and of the prioritizations we evaluate next, in Section 6.

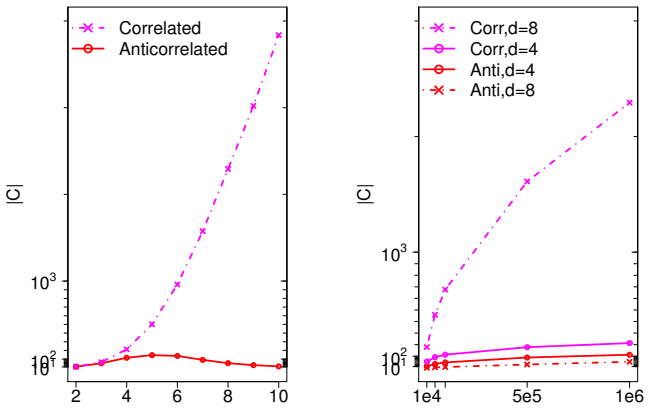
6. EXPERIMENTAL EVALUATION

In this section, we provide an extensive experimental evaluation of the contributions made thus far. We describe the basic, common setup for the experiments in Section 6.1. We evaluate the impact of Section 4.1 in Section 6.2 by measuring how large is the set C of points that closely dominate p, as a function of the input parametres. In Section 6.3, we compare the query performance of our two algorithms from Section 5 against adaptations using Theorem 5 of state-of-the-art algorithms for the SkyDist problem. Finally, in Section 6.4, we investigate our recursive algorithm in more depth, particularly the efficacy of its pruning rules and the average depth of the recursion.

6.1 Experimental Setup

Algorithms: We implement four algorithms in C++ for comparison. This includes both BRA and PrioReA from Section 5. We also adapt and implement the Sort-Projection (SORT_PROJ) and Space-Partition (SPACE_PART) methods of Huang et al. [12], since these were shown to be the most efficient SkyDist algorithms [12]. All four implementations start from the same reduced input set, C, based on the theoretical analysis in Section 4.

Environment: All experiments are run on a commodity machine with an i7-2700 quad-core processor clocked at 3.4 GHz, 16 GB of memory, and Ubuntu on kernel version 3.13.0. The code is compiled using the GNU C++ compiler version 4.8.2 with full optimization. Since BRA is embarassingly parallel (and the slowest running), we run it on 8 threads (hyperthreading is enabled). All other algorithms are single-threaded.[4]



(a) $|C|$ vs. d ($|\mathcal{S}| = 10^6$). (b) $|C|$ vs. $|\mathcal{S}|$ ($d = 4, 8$).

Figure 4: Variation of the size of the reduced input, C, relative to the full input, $\mathcal{S}$.

Datasets: We generate random datasets using the data generator standard to skyline research [3] to produce normalized datasets of *anticorrelated* (A) and *correlated* (C) distributions. The selection of dataset cardinality (n) and dimensionality (d), query constraints (q_L, q_U), and query points (p) varies according to the objective of each sub-study.

6.2 Regarding close dominance and $|C|$

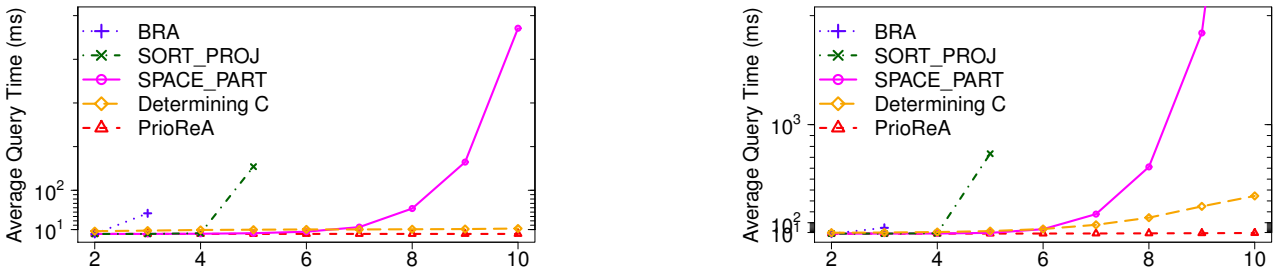
Experiment description: By Lemma 4, it is not the size of the input dataset that governs performance, but the size of the set C, the points that closely dominate p. We begin by empirically gauging the impact on the input size it has to apply this lemma.

We run 10^4 random trials for each configuration and report the average. For each trial, we pick a point $p \in \mathcal{S}$ uniformly at random. We then select each lower constraint, $q_L[i]$, uniformly from the data range, $[0, p[i]]$), independently for each attribute. Since q_U does not influence the set C, we set it to the maximum value (i.e., 1) on each attribute. With this setup, we then compute the set C using Lines $1-10$ of Algorithm 1 and record the number of points, $|C|$.

Results and discussion: The results are reported in Figure 4. In Figure 4a, we vary data dimensionality (d) from $2-10$ in increments of 1 and hold the data cardinality ($|\mathcal{S}|$) fixed at one million points. The pink line with x's shows the results for correlated data, and the orange line with o's, for anticorrelated data. Note that the y-axis, representing $|C|$, is logarithmic.

The results in this plot are very surprising, because correlated data exhibits more challenging behaviour than the anticorrelated data. In typical skyline experiments, anticorrelated data produces larger skylines (output). This slows performance because performance is typically dependent on the size of the output. Similarly, increases in d also hinder performance, because they also increase the size of the output. Here, however, we see that the reduced input set for

[4]Parallelizing the slowest running algorithm allows us to scale up the experiments without compromising fairness among the competitive algorithms.



(a) Anticorrelated

(b) Correlated

Figure 5: Execution time of the algorithms and the computation of C as a function of input parametres ($|\mathcal{S}| = 10^6$).

Sky-not queries is larger on correlated data, and grows exponentially with d. On anticorrelated data, it peaks around $d = 6$, then decreases with subsequent increases in d.

This counter-intuitive behaviour can be understood clearly by considering the extra requirement imposed by close dominance (Definition 6): that $c \in C$ iff $\nexists c' \in \mathcal{S}$ s.t. $c \prec c' \prec p$. On correlated data, it is likely that, for a randomly chosen point p, many other points dominate it. However, there are also many dominance relationships among them. As d increases, the points that dominate p become incomparable to each other. For points $c \prec c'$ that both dominate p, only $c \in C$. However, if $c \prec\succ c'$, then $c' \in C$ as well.

Considering anticorrelated data, on the other hand, there are much fewer points that originally dominate p. So, for low-dimensional increases to d, we observe the same trend as with correlated data, but less pronounced. However, by $d = 6$, this effect has been saturated, and most points that dominate p are incomparable to each other. For subsequent increases in d, many points in C that dominate p become incomparable to it; so, the size of the set shrinks.

Figure 4b, on the other hand, shows behaviour with increases data cardinality ($|\mathcal{S}|$). Because different values of d exhibited quite different results, we plot twice as many curves here: two for $d = 4$ and two for $d = 8$ (either side of the peak for the anticorrelated data). The pink lines are, again, correlated data; the orange, anticorrelated. The x's correspond to $d = 8$ and the o's, $d = 4$.

Here, we see easily predicted behaviour. By increasing the number of points in $\mathcal{S}$, we consequently increase the number of points in C. For anticorrelated data and for $d = 4$ on correlated data, the relationship is roughly linear, but the savings offered by Lemma 4 is dramatic. Only $1/10^4$ of the points need to be processed. All possible solutions involving values from any of the other 9999/10000 of the data points can be discarded early in preprocessing by all algorithms.

The exceptional case is the higher-dimensional, correlated data. This again grows steeply in accordance with Figure 4a. Of the additional points, many dominate p because the data is correlated, but many also are incomparable to each other because of the higher dimensionality.

In summary, this study shows that Lemma 4 can dramatically reduce the input, and thus also the search space. Counter-intuitively, it also demonstrates that for Sky-not queries, correlated data is much more challenging than anticorrelated data, with the former becoming exponentially more challenging with increases in $d > 6$ and the latter becoming easier under the same conditions.

6.3 On the scalability of the algorithms

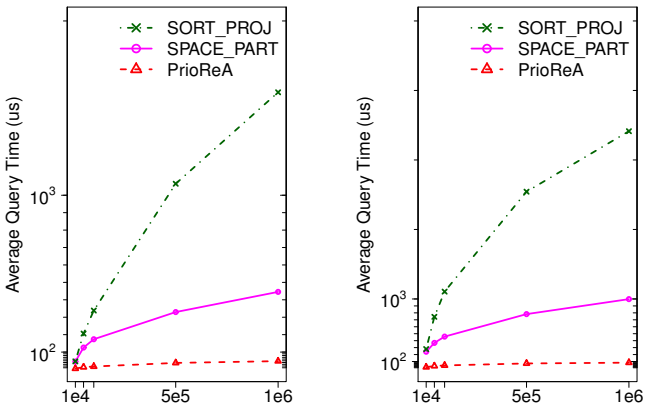
Experiment description: We next compare the four algorithms in terms of execution time, using the same experimental configurations as in Section 6.2. In contrast to the query generation methodology of [12], which first chooses m skyline points, and then generates a virtual query point that is dominated by all of them, our methodology ensures that query points still come from the original underlying distribution. So, we expect to observe performance following the trends illustrated in Figure 4. For readability, we separate the cardinality plots using different dimensionality.

Results and discussion: Figure 5 shows performance of the algorithms with respect to d. We also include the time taken to compute the set C, which is not included in the time for any of the four algorithms. Generally speaking, it is relatively efficient compared to all the algorithms. The exception, relative to PrioReA, on correlated data for higher d, is quite interesting. Recall that C is computed BNL-style with a window of current points to which each candidate is compared. Much like anticorrelated data slows skyline computation in BNL with increasing d, the correlated data slows the generation of C with increasing d.

Figure 5a gives performance on anticorrelated data, and Figure 5b, correlated. A first observation is the different scales on the y-axis. As expected from the larger C input set, the correlated data generally takes longer to compute for all algorithms. Another immediate observation is that, excepting PrioReA, all the algorithms deterioate rapidly after a threshold dimensionality. For BRA, this is unfortunately just $d = 3$; so, we exclude it from all subsequent experiments due to its poor scalability. Evidently, the combination of a large window size, `W`, and a still quite large search space, make BRA prohibitively slow.

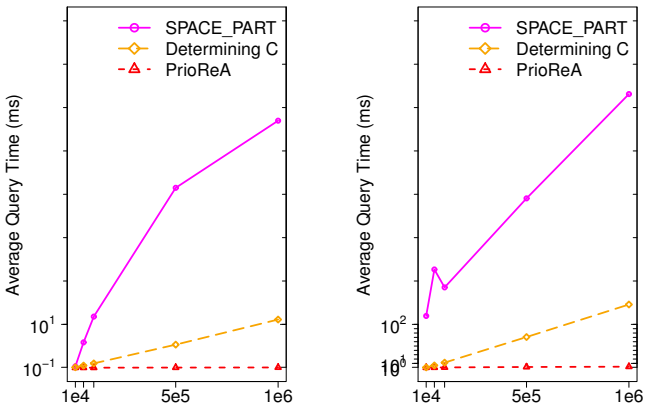
In agreement with the findings in [12], we observe SPACE_PART to be both faster and more scalable than SORT_PROJ. They both deteriorate rapidly after a threshold dimensionality– SORT_PROJ at $d = 5$ and SPACE_PART at $d = 8$. Interestingly, these thresholds are independent of the data distribution, which suggests that it is not $|C|$, which is decreasing on the anticorrelated data, that is the cause. Rather, it is strictly a function of the dimensionality.

In contrast, PrioReA scales quite gracefully with increasing dimensionality, independent of the data distribution. We investigate why in Section 6.4.



(a) Anticorrelated (b) Correlated

Figure 6: Query performance vs. $|\mathcal{S}|$ ($d = 4$).



(a) Anticorrelated (b) Correlated

Figure 7: Query performance vs. $|\mathcal{S}|$ ($d = 8$).

Next, we consider increasing $|C|$. Figure 6 shows results with $d = 4$ and Figure 7 shows results with $d = 8$. Because BRA did not scale to these dimensionalities, we exclude it from these plots. Similarly, SORT_PROJ did not scale to $d = 8$; so, we instead show the preprocessing time.

With the exclusion of the slower-performing algorithms, we can portray a more granular comparison of the algorithms. We see now that, even at $d = 4$, PrioReA is already an order-of-magnitude faster than the next fastest algorithm. We will investigate this difference in the next subsection. All three algorithms scale gracefully with increases in cardinality, but SORT_PROJ and SPACE_PART have a much larger scaling factor than PrioReA.

In summary, this study shows that PrioReA has much better scalability than the other three algorithms, with respect to both d and $|\mathcal{S}|$. We also see that the preprocessing phase is efficient for the savings in Section 6.2.

6.4 Granular Analysis of Recursion

We saw in Section 6.3 that SPACE_PART substantially outperforms both BRA and SORT_PROJ and that PrioReA outperforms SPACE_PART by an additional order of magnitude. Here, we conduct more granular experiments to explain the performance. Since both SPACE_PART and PrioReA are recursive algorithms, we compare them in Section 6.4.1 based on the number of recursive calls each makes. Then, in Section 6.4.2, we evaluate the success rate of PrioReA's pruning rules from Section 5.2.1.

6.4.1 *Number of recursive calls*

Experiment description: In order to better understand the discrepancy in performance between SPACE_PART and PrioReA, we study here the average number of recursive calls made by each algorithm. To do this, we insert a global counter variable into the source code of the recursive method for each algorithm. We adopt the same experimental configurations as in the earlier tests.

Results and discussion: Figure 8 gives results of the recursion experiments. Here, the pink lines represent SPACE_PART and the orange lines represent PrioReA. The lines with x's show correlated data and the lines with o's, anticorrelated. The y-axis is the number of times the recursive method is invoked per Sky-not query, plotted on a logarithmic scale.

Figure 8a shows the variation with respect to d with $|\mathcal{S}| = 10^6$. It is worth contrasting this to Figure 5, the query times for the same configurations. On the correlated data, we observe that the number of recursive calls made by SPACE_PART grows dramatically, even on the logarithmic scale. Note that the growth pattern of $|C|$ in Figure 4a has the same inflection point at $d = 4$. Intuitively, the number of recursive calls is increasing with the effective input size, $|C|$.

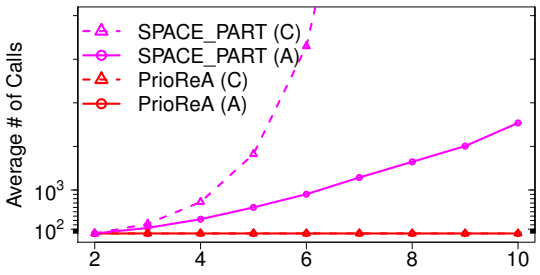
For the anticorrelated data, the number of recursive calls still grows exponentially for SPACE_PART, but more controllably. With each addition of a dimension, the number of partitions created by SPACE_PART doubles, but the occupancy of these partitions shrinks (because the same number of points are distributed among a larger number of partitions).

In contrast, the number of recursive calls made by PrioReA is stable with increasing d, indicating that the condition on Line 9 of Algorithm 2, which tests whether or not to recurse on a subset of dimensions, is not affected strongly by the *number of* dimensions, only by the quality of the solutions yet seen. This effect is distribution-independent.

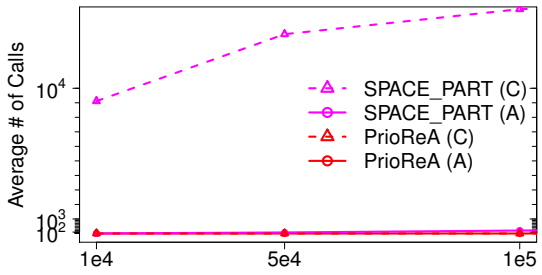
Figure 8b shows the variation with respect to $|\mathcal{S}|$ with $d = 8$. Note that the scale of the y-axis is different from the previous plot. We only show the results for $d = 8$, since they are mirrored, but less pronounced, at $d = 4$. Contrast these results to the query times in Figure 7. Here, we observe that, in contrast, to increases in d, the behaviour of both algorithms is relatively stable. In fact, they almost exactly mirror the trends in Figure 4b, which show $|C|$. With increasing input size, the algorithms both incur proportionately more recursive calls.

Note, importantly, that there is a large deviation, over an order of magnitude, between the number of recursive calls made by SPACE_PART and PrioReA on anticorrelated data. It is difficult to see in the plot because the range of the scale must be very large to fit SPACE_PART on correlated data.

In summary, we see the query time performance of SPACE_PART exhibits the same behaviour as the number of recursive calls, with sharp inflection points as d increases and more stable growth with $|\mathcal{S}|$. For PrioReA, the stable query times are matched by a stable number of recursive calls.

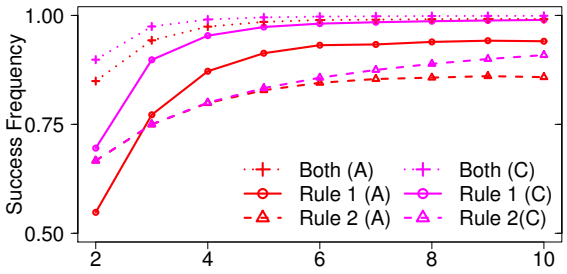


(a) # of recursive calls vs. d ($|\mathcal{S}| = 10^6$).

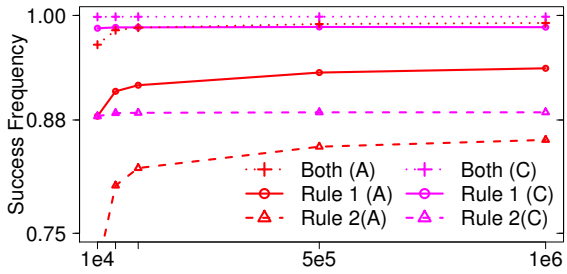


(b) # of recursive calls vs. $|\mathcal{S}|$ ($d = 8$).

Figure 8: Number of recursive calls relative to input configurations.



(a) Success rate of pruning rules vs. d ($|\mathcal{S}| = 10^6$).



(b) Success rate of pruning rules vs. $|\mathcal{S}|$ ($d = 8$).

Figure 9: Success rate of pruning rules relative to input configurations.

6.4.2 *Pruning efficacy*

Experiment description: Our last experiment investigates why the number of recursive calls for PrioReA is so stable. In particular, because recursive calls are averted by the pruning rules introduced in Section 5.2.1, we evaluate their success rates. We use, again, the same configurations, and add global counters at Lines 12, 14, 16, and 18 of Algorithm 2 to profile the percentage of invocations taking each path through the possible conditions.

Results and discussion: Figure 9 presents the results. Pruning rules 2 and 2′ are combined, since they both break the loops early based on the result of, or estimation of, the quality of the solution on the recursion, respectively. Pruning rule 1, on the other hand, does not break the loop, and is based on an estimate of the cost for a current value, $s[d]$.

Here, the y-axis indicates the fraction of times that the relevant condition evaluates favourably for the given pruning rule. The pink lines represent correlated data and the orange lines, anticorrelated. Rule 1 is depicted with o's and Rule 2, with x's. The combined success rate, $1 - (1 - \text{Rule1}) * (1 - \text{Rule2})$, is computed analytically and plotted.

Figure 9a shows the success rates with respect to d, with $|\mathcal{S}| = 10^6$. Interestingly, as d increases, so does the success rate of both pruning rules. This is the effect of our prioritizing the most promising (i.e., closest) dimensions first, obtaining good solutions very quickly. As the dimensionality increases, so do the number of choices of dimensions to prioritize. Additionally, the reasonably good solutions are found quite quickly, regardless of how many iterations through the loops we need, and can subsequently improve pruning power. As d increases, so too does the number of iterations through the loops on the first recursive call. However, the first few find the good solutions and boost the pruning potential for the increased number of subsequent iterations.

Rule 1 is, for $d > 2$, more effective than Rule 2. However, it is also tested first in Algorithm 2. So, easily pruneable cases are pruned by Rule 1 and never evaluated by Rule 2. All cases evaluated by Rule 2 were missed by Rule 1; so, it is not surprising that the success rate of pruning these cases is slightly lower.

Both pruning rules are more often successful on correlated data than on anticorrelated data. As with increases in d, this is because there are more tests in general (the loop on Line 6 of Algorithm 2 is larger), but a relatively constant number of unsuccessful, early evaluation of the pruning conditions.

By $d = 4$, we see that the combined success rate of the pruning rules is very nearly 100% on both data distributions, which clearly explains why the number of recursive calls observed in Section 6.4.1 was stable with increasing d.

Figure 9b shows the success rates with respect to $|\mathcal{S}|$, with $d = 8$. As before, we omit highly similar results for $d = 4$. Here we see that, aside from initial relatively large jumps in input size, increases in cardinality do not especially affect the pruning rates. At $d = 8$, they are very high, nearly 100% in combination, on smaller datasets as well. Note here that the scale on the y-axis is smaller than Figure 9a, and that the success rates are consistently above 80% for both rules.

In summary, the evaluation of the pruning rules shows

that, indeed, they are responsible for pruning the majority of the recursive calls, which, in turn, was shown to reflect the query performance. As the size of C grows, so too do the pruning rule success frequencies. Consequently, PrioReA is able to outperform the competing algorithms consistently and scale gracefully.

7. CONCLUSION AND OUTLOOK

In this paper, we introduced the Sky-not query to empower a user to better understand skyline results. Given a point p and a constrained skyline query, the Sky-not query returns the minimal change to the user's constraints that places p in the skyline. This can be used both to understand how competitive p is relative to the skyline and to dynamically adapt queries to fit user needs and expectations.

Towards this, we first conducted a theoretical study of Sky-not queries, showing that the space of possible solutions can be discretized and the set of relevant input points can be dramatically reduced. We also showed Sky-not queries are dual to the minimum skyline distance problem, implying techniques for each can be shared. We then presented two novel algorithms: BRA transforms the Sky-not query one highly resembling a new skyline instance, and adapts the BNL skyline algorithm; PrioReA uses theoretically motivated pruning rules in a recursive framework. The latter we showed has excellent empirical performance, largely on account of the success rate of the pruning rules.

We focused in this paper on improving the usefulness of skyline queries that include selection, perhaps the most fundamental element of database queries. There is much interesting work to be done in expanding this research with other clauses, such as joins and aggregations, where there is yet more complexity for the user to understand query results. Furthermore, there is a broad range of applications for skyline queries, such as road networks and social network graphs, and these settings may provide unique, specific challenges for supporting user understanding of skyline results.

8. ACKNOWLEDGMENTS

This research was conducted as part of the WallViz project (`http://wallviz.dk/`), supported by the Danish Council for Strategic Research, grant 10-092316. We would like to thank Darius Šidlauskas, Michael Lind Mortensen, and Kenneth S Bøgh for discussions that inspired this research.

9. REFERENCES

[1] Bartolini, I., Ciaccia, P., and Patella, M. Efficient sort-based skyline evaluation. *TODS 33*, 4 (2008), 31:1–49.

[2] Bidoit, N., Herschel, M., and Tzompanaki, K. Query-based why-not provenance with NedExplain. In *Proc. EDBT* (2014), pp. 145–156.

[3] Börzsönyi, S., Kossman, D., and Stocker, K. The skyline operator. In *Proc. ICDE* (2001), pp. 421–430.

[4] Chapman, A., and Jagadish, H. V. Why not? In *Proc. SIGMOD* (2009), pp. 523–534.

[5] Cheema, M. A., Lin, X., Zhang, W., and Zhang, Y. A safe zone based approach for monitoring moving skyline queries. In *Proc. EDBT* (2013), pp. 275–286.

[6] Chester, S., Mortensen, M. L., and Assent, I. On the suitability of skyline queries for data exploration. In *Proc. ExploreDB* (2014), pp. 7:1–6.

[7] Chester, S., Šidlauskas, D., Assent, I., and Bøgh, K. S. Scalable parallelization of skyline computation for multi-core processors. In *Proc. ICDE* (2015).

[8] Chomicki, J., Godfrey, P., Gryz, J., and Liang, D. Skyline with presorting. In *Proc. ICDE* (2003), pp. 717–719.

[9] He, Z., and Lo, E. Answering why-not questions on top-k queries. In *Proc. ICDE* (2012), pp. 750–761.

[10] Herschel, M., and Hernández, M. A. Explaining missing answers to SPJUA queries. *PVLDB 3*, 1–2 (2010), 185–196.

[11] Huang, J., Chen, T., Doan, A., and Naughton, J. F. On the provenance of non-answers to queries over extracted data. *PVLDB 1*, 1 (2008), 736–747.

[12] Huang, J., Jiang, B., Pei, J., Chen, J., and Tang, Y. Skyline distance: a measure of multidimensional competence. *Knowl. Inf. Syst. 34*, 2 (2013), 373–396.

[13] Islam, M. S., Zhou, R., and Liu, C. On answering why-not questions in reverse skyline queries. In *Proc. ICDE* (2013), pp. 973–984.

[14] Kim, Y., You, G.-w., and Hwang, S.-w. Ranking strategies and threats: a cost-based pareto optimization approach. *Distributed and Parallel Databases 26*, 1 (2009), 127–150.

[15] Lee, J., and Hwang, S.-w. Scalable skyline computation using a balanced pivot selection technique. *Inf. Syst. 39* (2014), 1–21.

[16] Lee, J., You, G.-w., Hwang, S.-w., Selke, J., and Balke, W.-T. Interactive skyline queries. *Inf. Sci. 211*, 30 (2012), 18–35.

[17] Lu, H., and Jensen, C. S. Upgrading uncompetitive products economically. In *Proc. ICDE* (2012), pp. 977–988.

[18] Magnani, M., Assent, I., Hornbæk, K., Jakobsen, M. R., and Larsen, K. F. SkyView: a user evaluation of the skyline operator. In *Proc. CIKM* (2013), pp. 2249–2254.

[19] Meliou, A., Gatterbauer, W., and Moore, K. F. Why so? or Why no? Functional causality for explaining query answers. Tech. Rep. UW-CSE-09-12-01, University of Washington, Seattle, WA, USA, 2009.

[20] Papadias, D., Tao, Y., Fu, G., and Seeger, B. Progressive skyline computation in database systems. *PODS 30*, 1 (2005), 41–82.

[21] Peng, Y., and Wong, R. C.-W. Finding competitive price. In *Proc. SIGSPATIAL* (2013), pp. 144–153.

[22] Santoso, B. J., and Chiu, G.-M. Close dominance graph: an efficient framework for answering continuous top-k dominating queries. *TKDE PP* (2013), 1–14.

[23] Tan, K.-L., Eng, P.-K., and Ooi, B. C. Efficient progressive skyline computation. In *Proc. VLDB* (2001), pp. 301–310.

[24] Tran, Q. T., and Chan, C.-Y. How to ConQueR why-not questions. In *Proc. SIGMOD* (2010), pp. 15–26.

[25] Wan, Q., Wong, R. C.-W., Ilyas, I. F., Ozsu, M. T., and Peng, Y. Creating competitive products. *PVLDB 2*, 1 (2009), 898–909.

[26] Wu, P., Agrawal, D., Eğecioğlu, Ö., and El Abbadi, A. DeltaSky: optimal maintenance of skyline deletions without exclusive dominance region generation. In *Proc. ICDE* (2007), pp. 486–495.

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

[28] Zong, C., Yang, X., Wang, B., and Zhang, J. Minimizing explanations for missing answers to queries on databases. In *Proc. DASFAA* (2013), pp. 254–268.