Finding the Influence Set through Skylines

Xiaobing Wu[1] Yufei Tao[1] Raymong Chi-Wing Wong[2] Ling Ding[1] Jeffrey Xu Yu[1]

Chinese University of Hong Kong[1]
Sha Tin, New Territories, Hong Kong

Hong Kong University of Science and Technology[2]
Clear Water Bay, Hong Kong

ABSTRACT

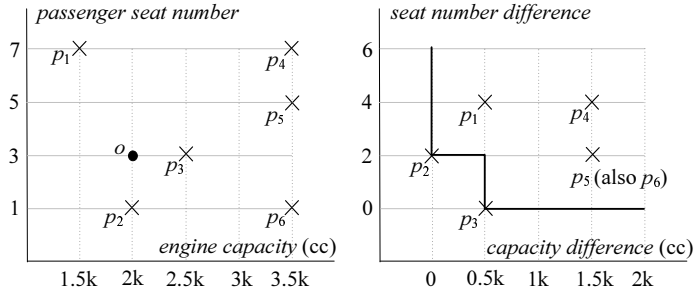
Given a set P of *products*, a set O of *customers*, and a product $p \in P$, a *bichromatic reverse skyline* query retrieves all the customers in O that do not find any other product in P to be absolutely better than p. More specifically, a customer $o \in O$ is in the *reverse skyline* of $p \in P$ if and only no other product in P better matches the preference of o on all dimensions.

The only existing bichromatic reverse skyline algorithm, which we refer to as *basic*, is designed for *uncertain* data. This paper focuses on traditional datasets, where each object is a *precise* point. Since a precise point can be regarded as a special uncertain object, *basic* can still be applied. However, as precise data are inherently easier to handle than uncertain data, one should expect that *basic* can be further improved by taking advantage of the reduced problem complexity. Indeed, we observe several non-trivial heuristics that can optimize the access order to achieve stronger pruning power. Motivated by this, we propose a new algorithm called *BRS*, and prove that *BRS* never entails more I/Os than *basic*. Besides our theoretical analysis, we also perform extensive experiments to show that in practice *BRS* usually outperforms *basic* by a large factor. For example, when both P and O follow the anti-correlated distribution, *BRS* is faster than *basic* by an order of magnitude. Finally, we address a new variation of bichromatic reverse skyline search where the conventional definition of dynamic skylines no longer makes sense.

1. INTRODUCTION

The skyline operator and its variants (as surveyed in Section 2) have been extensively studied in the past ten years, due to their significant importance in a large number of multi-criteria decision making applications. Specifically, a point p *dominates* another p', if the coordinate of p is not larger than that of p' on every dimension, and strictly smaller on at least one dimension. The *skyline* of a point set S includes all the points in S that are not dominated by any other point.

EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.



(a) The raw space (b) The transformed space

Figure 1: A dynamic skyline example

Some applications require the skyline of a dataset *dynamically* generated based on a given query. Consider a car dealer that is promoting six vehicle models, denoted by p_1, p_2, ..., p_6 in Figure 1a. Here, each point captures the engine *capacity* and the *number* of passenger seats of a vehicle. Point o represents the profile of a customer that prefers a vehicle with 3 passenger seats and a 2000-cc engine. Since no vehicle matches the profile exactly, the dealer makes recommendations by resorting to skyline analysis. For this purpose, each vehicle p_i ($1 \leq i \leq 6$) is transformed into a new space, where the horizontal (vertical) dimension equals the *absolute difference* between the capacities (seat-numbers) of p_i and o. Figure 1b demonstrates the resulting points (notice that p_5 and p_6 are converted to the same location). The skyline in the transformed space includes p_2 and p_3. The other vehicles would not be attractive to o. For instance p_1 would not attract o, because p_3 (dominating p_1) suits the profile of o at least as nicely on all aspects, and better on one attribute (i.e., *seat-number*). The dealer can thus recommend p_2 and p_3 to the customer.

The skyline in Figure 1b is called the *dynamic skyline* [24] of o. In general, if a point p is in the dynamic skyline of o, then o is said to be in the *reverse skyline* of p [19]. For example, in Figure 1, o is in the reverse skylines of p_2 and p_3. Intuitively, the reverse skyline of a vehicle p includes those customers that do not find any other vehicle to be better than p on both axes.

Generally, given a set P of products, a set O of customers, and a product $p \in P$, a *bichromatic reverse skyline* query retrieves all the customers in O whose dynamic skylines contain p. For example, let P be the set of vehicles in Figure 1a, and O be the set of customers in Figure 2. It is not hard

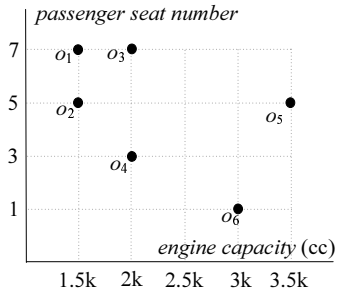


Figure 2: Customer dataset for bichromatic skyline retrieval

to verify that the dynamic skylines of o_1, o_2, o_3, o_4, o_5, o_6 are $\{p_1\}$, $\{p_1, p_5\}$, $\{p_1, p_2\}$, $\{p_2, p_3\}$, $\{p_5\}$, and $\{p_6\}$, respectively. Therefore, the reverse skyline of p_1 is $\{o_1, o_2, o_3\}$.

Bichromatic reverse skyline retrieval is useful in many other applications. As another example, consider a *personality-based matching* service, where P is a set of men and O a set of women. Every man (woman) is a point in a data space, where each dimension describes an aspect of personality. For instance, a dimension may indicate how much, in a range from 0 to 1, a person likes sports. Similarly, other dimensions may represent the degree of interest in ancient music, art, investment, etc. The bichromatic reverse skyline of a man $p \in P$ includes all the women in O that cannot find any other man matching their personalities better than p in all aspects. Bichromatic reverse skylines are also interesting in *profile-based investment*. Imagine P as a set of stocks and O as a set of customer profiles. The dimensions are different properties of stocks, such as risk (on the scale from 0 to 1), volatility, daily turn-over, and so on. Thus, the bichromatic reverse skyline of a stock $p \in P$ consists of all the customers in O that cannot find any other stock better than p in all dimensions.

The only existing algorithm for bichromatic reverse skyline queries is due to Lian and Chen [19]. That algorithm, which we refer to as *basic*, is designed for *uncertain* datasets, where each object is modeled as a probability distribution function (pdf). In this work, we focus on traditional datasets, where each object is a *precise* point. All the applications mentioned earlier are examples of bichromatic reverse skyline search on precise points.

Since a precise point can be regarded as a special uncertain object, *basic* is still applicable. However, as precise data are inherently easier to handle than uncertain data, one should expect that *basic* can be further improved by taking advantage of the reduced problem complexity. Indeed, we observe several non-trivial heuristics that can optimize the access order to achieve stronger pruning power. Motivated by this, we propose a new algorithm called *BRS*, and prove that *BRS never entails more I/Os than basic*. Besides our theoretical analysis, we also perform experiments to show that in practice *BRS* usually outperforms *basic* by a large factor. For example, when both participating datasets follow the anti-correlated distribution (a benchmark distribution for skyline research [3]) *BRS* is faster than *basic* by an order of magnitude.

Our last contribution is to point out that the conventional definition of dynamic skylines, hence also bichromatic reverse skylines formulated based on it, does not make sense in some applications. Specifically, we show that some dimensions have a fixed optimization direction, making it unreasonable to minimize the absolute difference between coordinates as in Figure 1. We address this problem by extending the notions of dynamic skylines and reverse skylines to the general scenario where the data space can have any mixture of dimensions, i.e., either those conventionally targeted by the previous work or the new ones identified in this paper. We extend both the *basic* and *BRS* algorithms to such a scenario.

The rest of the paper is organized as follows. Section 2 surveys the existing literature on skylines. Section 3 reviews the *basic* algorithm in [19]. Section 4 proposes our new algorithm *BRS* and proves its superiority over *basic*. Section 5 addresses a drawback of bichromatic reverse skylines when some dimensions of the data space have an obvious preferential direction. Section 6 evaluates the efficiency of our techniques with extensive experimentation. Section 7 concludes the paper with directions for future work.

2. RELATED WORK

The skyline literature has expanded to a vast scale. Next, we point to several major topics in the literature. The skyline operator is first introduced to databases by Borzsonyi et al. [3]. Since then, numerous algorithms have been proposed for its efficient computation. These solutions can be classified into two categories, depending on whether they assume an index. The no-index category includes *BNL* [3], *SFS* [9], *LESS* [13], *D&C* [3], *Bitmap* [32], and *Lattice* [22]. They demand no preprocessing at all, but must scan the entire database at least once. The with-index category, which includes *Index* [32], *B-tree* [3], *NN* [16], and *BBS* [24], achieves better efficiency by examining only the part of the dataset that may contain skyline points. In particular, the *BBS* algorithm is I/O-optimal, in the sense that it requires the fewest I/O accesses among all the algorithms using the same index. It is shown in [17] that *BBS* is especially efficient when applied on an R-tree built from the Z-order space filling curve.

The cardinality of the skyline increases rapidly with the dimensionality (see a detailed analysis in [7]). After the dimensionality has reached a certain level, service recommendation by skyline is ineffective, as the skyline becomes too gigantic for a customer to scrutinize manually. This calls for innovative approaches for eliminating certain skyline objects of relatively poor quality. These approaches include influence analysis [6], *k-dominant skyline* [5], and *k-representative skyline* [21].

There are also substantial works on skyline computation in arbitrary subspaces of the data space. In particular, SUBSKY [33] considers centralized databases, while SKYPEER [35] discusses distributed systems. An interesting concept in this regard is *skyline cube* [27, 40], which enumerates the skyline in every possible subspace. Fast algorithms for deriving the skyline cube are available in [25, 39].

Several works [20, 23, 29, 37] tackle the challenge of real-time skyline maintenance on data streams. The solutions of [14, 36] efficiently produce skylines from peer-to-peer networks. Skyline algorithms are also studied in web search [1], parallel computation [10, 34, 38], partially-ordered domains [4], spatial databases [30], road networks [12], uncertain databases [26], and multiple relations [15]. The skyline operator has led to the *dada cube* [18], which is an effective tool for optimizing product specifications. Furthermore,

skylines have been applied for other purposes such as bargaining [31] and privacy protection [8].

The concept of *reverse skyline* is first introduced by Dellis and Seeger [11], but in a *monochromatic* context involving a single dataset. Lian and Chen [19] extend the definition of reverse skyline to the bichromatic scenario, and propose an algorithm for its efficient computation on uncertain data. In the next section, we will discuss their technique in detail.

3. PROBLEM DEFINITION AND THE BASIC ALGORITHM

We consider two sets, denoted as P and O, of points in the same d-dimensional space Ω. Each point in P is referred to as a *product*, and each point in O a *customer*. Given any dimension X of Ω, we use $p[X]$ to denote the coordinate of a point p on X. Without loss of generality, assume that all the coordinates be non-negative.

The next definition describes, from the perspective of a customer $o \in O$, the superiority between two products p and p' of P.

DEFINITION 1 (DYNAMIC DOMINANCE). *In the view of a customer $o \in O$, a product $p \in P$* dynamically dominates *$p' \in P$ if (i) for each dimension X of Ω, $|o[X] - p[X]| \leq |o[X] - p'[X]|$, and (ii) equality does not hold on at least one dimension X.* □

We emphasize that the notion of dynamic dominance is meaningful only *in the view* of a customer. The dominance relation between two products p and p' may actually be reversed in the views of different customers.

DEFINITION 2 (BICHROMATIC REVERSE SKYLINE). *Given a product $p \in P$, its* bichromatic reverse skyline *is the set of customers $o \in O$ such that, in the view of o, p is not dynamically dominated by any other product $p' \in P$.* □

All "reverse skylines" in the sequel are bichromatic (as opposed to its monochromatic counterpart in [11]).

Suppose that we want to find the reverse skyline of $p \in P$. Let us cut the space Ω into 2^d *quadrants* with d orthogonal hyper-planes. For example, given p_3 in Figure 3a, we partition Ω using a horizontal and a vertical line. Note that the boundary between two quadrants belongs to both quadrants. Let us extract a skyline of P in every quadrant, and call it a *quadrant skyline*. For example, in Figure 3a, the quadrant skyline in the first quadrant is $\{p_5\}$, and that in the second quadrant is $\{p_6\}$, and so on. The next definition formalizes the notions of "dominance in a quadrant", and "quadrant skyline".

DEFINITION 3 (QUADRANT SKYLINE). *Let p be a product in P, and Ω_{quad} any quadrant of p. Given two points x and y in Ω_{quad}, x* dominates *y in Ω_{quad} if $|x[X] - p[X]| \leq |y[X] - p[X]|$ on all dimensions X, and equality does not hold on at least one dimension.*

Let P_{quad} be the set of products in $P - \{p\}$ that are in Ω_{quad}. The quadrant skyline *of p in Ω_{quad} includes all the products in P_{quad} that are not dominated by any other product of P_{quad} in Ω_{quad}.* □

Let S_{quad} be a quadrant skyline of p. Then, for every product $\hat{p} \in S_{quad}$, take the mid-point $\hat{p}'$ of (the segment connecting) $\hat{p}$ and p. Replacing each $\hat{p}$ with $\hat{p}'$, we derive a new point set S'_{quad}, and call it a *midway quadrant skyline* of p. To illustrate, consider the first quadrant in Figure 3a, where the quadrant skyline of p_3 is $\{p_5\}$. The midway skyline of p_3 in this quadrant contains a single point p'_5, which, as shown in Figure 3b, is the mid-point of p_3 and p_5. Similarly, the midway skylines of p_3 in quadrants 2, 3, 4 are $\{p'_6\}$, $\{p'_2\}$, and $\{p'_1\}$, respectively.

In each quadrant, there is a portion, referred to as the *anti-dominance area* (ADA), which is not dominated by any point in the *midway* skyline of p in this quadrant. Note that the ADA includes every midway skyline point, since a point does not dominate itself. The *search region* of p is the union of the ADAs of all the quadrants. In Figure 3b, the shaded area represents the search region of p_3.

We are ready to give an important property of reverse skylines.

LEMMA 1. *(Proved in [19]; see Lemmas 5.1 and 5.2 there) The reverse skyline of a product $p \in P$ is exactly the set of customers of O, which fall in the search region of p.* □

Let O the set of dots in Figure 3c. Given the search region of p_3 shown in Figure 3b, by Lemma 1, we immediately conclude that the reverse skyline of p_3 contains only o_4. Figures 3d and 3e present the search regions of p_1 and p_6, respectively. It is easy to see that the reverse skyline of p_1 is $\{o_1, o_2, o_3\}$, and that of p_6 is $\{o_6\}$.

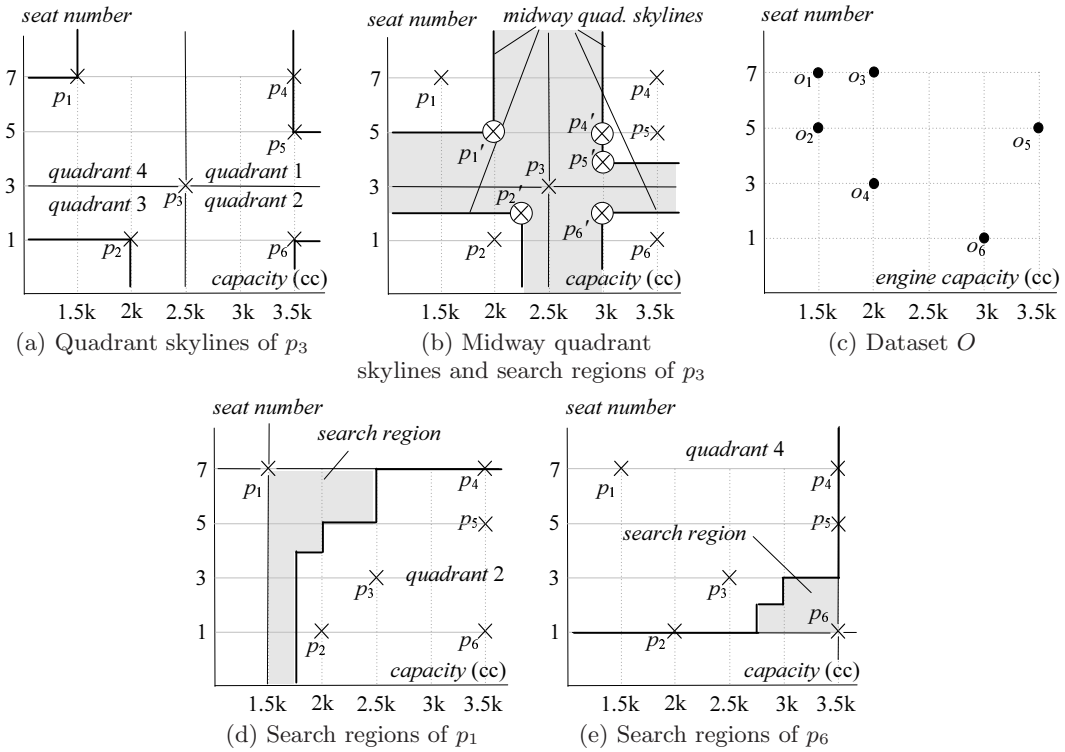
Leveraging Lemma 1, Lian and Chen [19] propose an algorithm *basic* for computing the reverse skyline of p. *Basic* is originally designed for an *uncertain* dataset, where each object is described by a pdf. However, since a point can be regarded as a special pdf, *basic* trivially applies to precise datasets too.

Specifically, given a precise dataset, *basic* works in two steps. The first *P-step* retrieves all the quadrant skylines of p, which can be accomplished using, for example, the *BBS* algorithm reviewed in Section 2. Once the quadrant skylines are derived, all the midway quadrant skylines of p are determined as well — simply replacing each product $\hat{p}$ in a quadrant skyline with the mid-point of $\hat{p}$ and p. Then, the second *O-step* fetches all the customers that are inside the search region of p. This can be done efficiently using an index (e.g., an R-tree) on O.

Finally, we note that *basic* as described in [19] interleaves the two P- and O-steps in its execution, instead of separating them. The interleaving is necessary in the context of [19], where each object is represented by a large number of points describing its probability density function. In our scenario where each object is a precise point, the original algorithm in [19] achieves exactly the same performance as the *basic* algorithm described above.

4. THE BRS ALGORITHM

The *basic* algorithm has two drawbacks. First, it lacks *progressiveness* in result outputting, because no reverse skyline point can be returned until the P-step has completed. Second, its P-step totally ignores the dataset O, and therefore, misses the opportunity of using the characteristics of O



(a) Quadrant skylines of p_3 (b) Midway quadrant skylines and search regions of p_3 (c) Dataset O

(d) Search regions of p_1 (e) Search regions of p_6

Figure 3: Search regions of bichromatic reverse skyline retrieval

to prune the data of P. Remember that *basic* is designed for uncertain data, where the high complexity of the problem limits the variety of heuristics that can be used. As precise data are easier to process, intuitively one should expect the existence of a faster algorithm.

In this section, we develop a new algorithm *BRS* (bichromatic reverse skyline) that overcomes the drawbacks of *basic*. *BRS* aims at minimizing the I/O cost, since reverse skyline retrieval is I/O-bounded, as shown in the experiments. It assumes that each of P and O is indexed by a multi-dimensional access method. Our discussion uses R-trees, but it can be easily adapted to other data-partitioning or space-partitioning indexes such as Quad-trees, kdB-trees, etc. In Section 4.1, we describe a transformation that significantly simplifies our analysis. Then, Section 4.2 elaborates the proposed heuristics, based on which Section 4.3 presents the details of *BRS*, and proves that it *never* entails higher I/O overhead than *basic*.

4.1 More on Midway Conversion

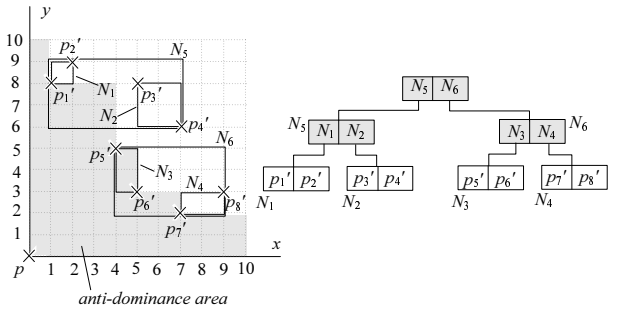
For simplicity, we will use 2D examples, since our solution can be generalized to any dimensionality in a straightforward manner. Recall that, given a *query product* $p \in P$, its search region is the union of the ADAs (anti-dominance areas) of four quadrants (see Figure 3b). Since all quadrants are symmetric, it suffices to focus on only quadrant 1 of p. For this reason, we will treat the quadrant as a new data space Ω' directly, which has p as its origin. Accordingly, P (O) is assumed to contain only the products (customers) of this quadrant. In Section 4.3, we will come back to this issue and remove the assumption.

Further recall that the search region of p is bounded by its midway skylines, which, in turn, are derived from its quadrant skylines (again, consult Figure 3b). To eliminate this complication, let us replace each product $\hat{p} \in P$ ($\hat{p} \neq p$) with the mid-point $\hat{p}'$ of $\hat{p}$ and p. This way, we obtain a new point set P', which consists of all the mid-points created from P. In the sequel, we will work with P' only; the benefit is that the notion of "midway skyline" is no longer needed — the midway skyline of p (in the first quadrant) is simply the skyline of P' in Ω'.

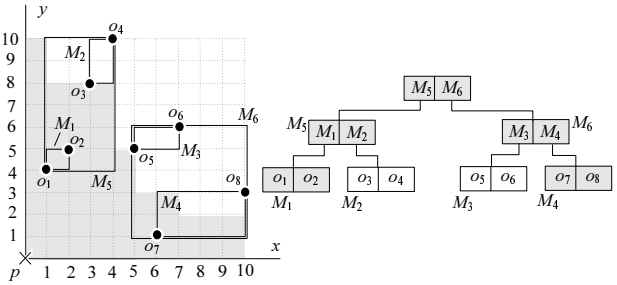
To illustrate the above *midway conversion*, let $p = p_3$ in Figure 3b. Thus, Ω' is the first quadrant of p_3, and P, after eliminating all the products outside quadrant 1, is $\{p_4, p_5\}$. Hence, P' includes two points p'_4 and p'_5 as shown in the figure. The midway skyline of p_3 in quadrant 1 is $\{p'_5\}$, which is the skyline of P' in Ω'.

It is important to note that our midway conversion is *conceptual*, namely, P' is not materialized; every time a product $\hat{p} \in P$ is encountered, its mid-point $\hat{p}'$ is computed on the fly with no I/O overhead. This also applies to the R-tree on P. Specifically, the MBR N of any node in the tree can be trivially transformed to an MBR N' of a node in an R-tree on P' — each corner of N' is the mid-point of p (i.e., the query) and the corresponding corner of N. Hence, from now on, we will consider that an R-tree on P' is available.

Now, we re-visit the *basic* algorithm using a more complex example in Figure 4. Specifically, Figure 4a demonstrates a P' including eight points p'_1, ..., p'_8, together with the MBRs in the R-tree on P', which is given in Figure 4b. Notice that the origin of the coordinate system is the query p. Figure 4c shows a set O involving eight customers o_1, ..., o_8, and the MBRs of the R-tree on O in Figure 4d. The P-step of *basic* first extracts the skyline $S_{sky} = \{p'_1, p'_5, p'_6, p'_7\}$ of P' (i.e., the midway skyline of p in the first quadrant). The shaded area of Figure 4a indicates the ADA of the skyline.



(a) The P' dataset (b) The R-tree on P' (derived from the R-tree on P with no I/O cost)



(c) The O dataset (d) The R-tree on O

Figure 4: The running example for Section 4

Then, the O-step fetches all the customers that are inside the shaded area. Therefore, the reverse skyline of p contains o_1, o_2, and o_7 (note that o_3 and o_5 are not in the ADA, because they are dominated by p'_1 and p'_5, respectively).

Let us implement the two steps using the R-trees. The best way to find S_{sky} is to employ the state-of-the-art algorithm *BBS* [24]. As a property of this algorithm [24], *BBS* visits all the nodes whose MBRs intersect the ADA. Hence, in our example the P-step explores all the nodes in Figure 4b except N_2. Then, the O-step simply accesses the shaded nodes in Figure 4d, whose MBRs intersect the ADA.

Which I/Os can be avoided? Certainly not the nodes (in the tree of O) visited in the O-step, as skipping any of them potentially incurs false misses. However, we can hope to avoid accessing some nodes fetched by *BBS* because, as explained later, it is possible to retrieve the reverse skyline *without getting the entire* S_{sky}. As shown later in Section 4.3, our algorithm *BRS* explores only the shaded nodes in the tree of Figure 4b, i.e., saving three node accesses (N_1, N_3, N_4) from *BBS*.

4.2 Heuristics

Before discussing our pruning strategies, we first introduce two crucial concepts: *P-safe collection* and *O-safe collection*, which are sets of (intermediate/leaf) entries in the R-trees of P and O, respectively. Informally, these entries have an important feature: even without further information from either tree, we can still compute the reverse skyline correctly, by resorting to only those collections. Next, we provide the formal definitions.

Definition 4 (P-safe Collection). *A* P-safe collection *is a set* $\mathcal{C}_P$ *of entries in the R-tree of* P' *such that, any point in the skyline* S_{sky} *of* P' *is either in* $\mathcal{C}_P$*, or in the subtree of an entry in* $\mathcal{C}_P$*.* □

For example, as explained earlier, in Figure 4a the skyline S_{sky} of P' is $\{p'_1, p'_5, p'_6, p'_7\}$. A legal $\mathcal{C}_P$ can be $\{p'_1, N_3, N_4\}$. Obviously, any point in S_{sky} is either in $\mathcal{C}_P$, or in the subtree of an entry in $\mathcal{C}_P$ (e.g., p'_5 is underneath N_3). Note that $\{p'_1, N_2, N_3, N_4\}$ is also a legal $\mathcal{C}_P$, even though it has a "redundant" N_2 that contains no skyline point in its subtree. Intuitively, $\mathcal{C}_P$ gives us enough information to find S_{sky} exactly, whenever necessary.

Definition 5 (O-safe Collection). *An* O-safe collection *is a set* $\mathcal{C}_O$ *of entries in the R-tree of* O *such that, if* $o \in O$ *is a reverse skyline point that has not been reported, then* o *must be either in* $\mathcal{C}_O$*, or in the subtree of an entry* $\mathcal{C}_O$*.* □

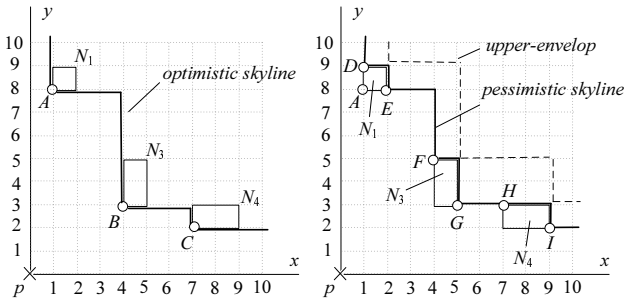
We stress that $\mathcal{C}_O$ depends on which reverse skyline points have been already output. For instance, as mentioned before, in Figure 4c the final reverse skyline contains o_1, o_2, and o_7. Assuming that none of them has been reported yet, a $\mathcal{C}_O$ can be $\{M_1, M_4\}$, or $\{M_1, M_2, o_7\}$ (as with $\mathcal{C}_P$, $\mathcal{C}_O$ may have redundant entries, e.g., M_2 here, which has no reverse skyline point in its subtree). If o_7 has been reported, then $\{M_1\}$ becomes a legal $\mathcal{C}_O$. Intuitively, $\mathcal{C}_O$ ensures that we can eventually discover *all* the reverse skyline points without any false miss.

Next, given a non-empty $\mathcal{C}_P$, we formulate an optimistic and a pessimistic estimate of the actual skyline S_{sky} of P'. To obtain an *optimistic skyline* S_{opt}, for each entry $e \in \mathcal{C}_P$, we take the *min-corner* of (the MBR of) e, which is the corner of e closest to the query p. Specially, if e is a point, then its min-corner is simply the point itself. S_{opt} is the skyline of the min-corners of all the entries in $\mathcal{C}_P$. To illustrate, consider the MBRs in Figure 5a, and assume $\mathcal{C}_P = \{N_1, N_3, N_4\}$. The min-corners of N_1, N_3, N_4 are A, B, C respectively, which constitute S_{opt}.

The construction of the *pessimistic skyline* S_{pes} is more complicated. For each entry $e \in \mathcal{C}_P$, we will prepare d *minmax-corners*[1] of e, where d is the dimensionality. Specifically, we first take the min-corner of e, which is adjacent to d boundaries of (the MBR of) e. Each boundary is essentially a $(d-1)$-dimensional rectangle. We use the point of this rectangle, which is the farthest to the query p, as a *minmax-corner* of e. Specially, if e is a point, then all its d minmax-corners degenerate into e itself. S_{pes} is the skyline of the minmax-corners of all entries in $\mathcal{C}_P$. For example, consider again $\mathcal{C}_P = \{N_1, N_3, N_4\}$. As mentioned before, the min-corner of N_1 is A, which is adjacent to two boundaries of N_1: edges AE and AD, respectively (Figure 5b). On boundary AE (AD), the farthest point to p is E (D); hence, the minmax-corners of N_1 are E and D. Similarly, the minmax-corners of N_3 (N_4) are F, G (H, I). Therefore, $S_{pes} = \{D, E, F, G, H, I\}$.

Similar to the ADA of S_{sky}, let the *anti-dominance area* (ADA) of S_{opt} (or S_{pes}) be the portion of the first quadrant of the query p that is not dominated by any point in S_{opt} (S_{pes}). Intuitively, the optimistic (pessimistic) skyline S_{opt} (S_{pes}) is below (above) the skyline S_{sky} of P'. More formally, *the ADA of* S_{opt} *is entirely covered by the ADA of* S_{sky}*, and the ADA of* S_{pes} *completely covers the ADA of* S_{sky}. This can be observed by comparing the ADA of

[1]The name stems from the well-known metric of *MINMAX distance* [28].



(a) Optimistic skyline S_{opt} (b) Pessimistic skyline S_{pes}

Figure 5: Skyline approximation with a P-safe collection $\mathcal{C}_P = \{N_1, N_3, N_4\}$

S_{opt} (S_{pes}) in Figure 5a (5b) with the shaded area in Figure 4a. It is worth mentioning that Papadias et al. [24] propose the similar concepts of *lower* and *upper envelops*. While the lower envelop is equivalent to the optimistic skyline, the pessimistic skyline is a tighter estimate than the upper envelop. This is evident in Figure 5b (an upper envelop is the skyline of the farthest corners (from p) of a set of MBRs).

Leveraging S_{opt} and S_{pes}, we are ready to clarify the proposed heuristics. The first one says that some nodes in the R-tree of O should be accessed early.

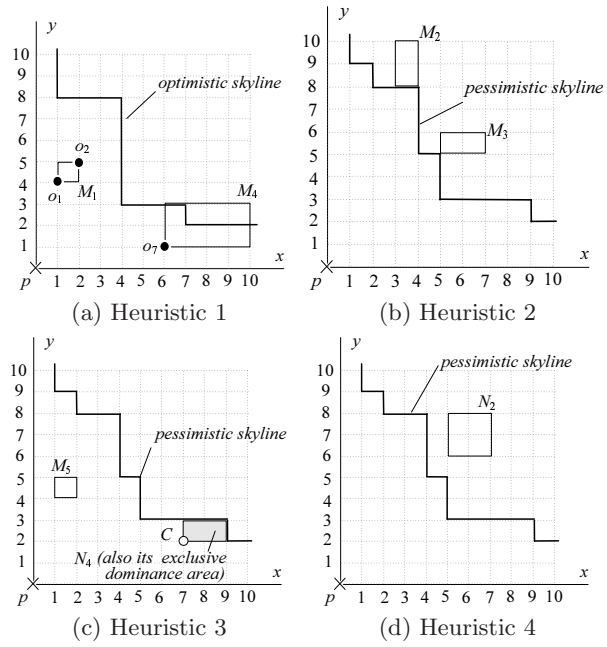
HEURISTIC 1. *Let M be a node in the R-tree of O. Access M immediately, if (the MBR of) M intersects the ADA of S_{opt}. Let o be a customer in O. If o is in the ADA of S_{opt}, then report o as a reverse skyline point.* □

To explain, Figure 6a demonstrates the optimistic skyline S_{opt} in Figure 5a, where the P-safe collection $\mathcal{C}_P = \{N_1, N_3, N_4\}$. Figure 6a also shows the MBRs M_1 and M_4 in the R-tree of O (from Figures 4c). As both M_1 and M_4 intersect the ADA of S_{opt}, they definitely intersect the ADA of the final S_{sky}, and thus, must be visited eventually to guarantee retrieval of all the reverse skyline points (recall our discussion in Section 4.1). There are two reasons why they should be accessed *immediately*. First, exploring them leads to the discovery of the reverse skyline points o_1, o_2, and o_7, which can be output directly (well before the algorithm terminates), since they are also in the ADA of S_{opt} (Heuristic 1). Second, the disappearance of M_1 and M_4 from the O-safe collection (Definition 5) actually enables pruning more nodes in the R-tree of P, as explained in Heuristic 3 later.

Unlike the above heuristic, which employs S_{opt} to decide nodes to be visited right away, the following heuristics deploy S_{pes} for pruning.

HEURISTIC 2. *Let M be a node in the R-tree of O. Prune M, if M is disjoint with the ADA of S_{pes}.* □

Figure 6b shows the pessimistic skyline S_{pes} in Figure 5b, and MBR M_2 in the R-tree of O (Figure 4c). Since M_2 does not intersect the ADA of S_{pes}, neither can it intersect that of S_{sky}. Hence, no customer in M_2 can be a reverse skyline point, and M_2 can be eliminated from further consideration. Similarly, the heuristic also prunes M_3.



(a) Heuristic 1 (b) Heuristic 2

(c) Heuristic 3 (d) Heuristic 4

Figure 6: Illustration of heuristics (P-safe collection $\mathcal{C}_P = \{N_1, N_3, N_4\}$)

The remaining heuristics concern pruning nodes from the R-tree of P'. Given S_{pes} and an MBR N from the tree, we define its *exclusive dominance area*, as the portion of the space Ω' that is dominated by the min-corner of N, but not by any point in S_{pes}. As an example, consider the S_{pes} in Figure 6c, and the MBR N_4 (a node from the R-tree of P' in Figure 4a). The shaded rectangle cornered at C is the exclusive dominance area of N_4, because any location in this area is dominated by the min-corner C of N_4, but not by any other point in S_{pes}. Utilizing this concept, we provide a rule that achieves strong pruning power (on the tree of P'), by leveraging information of the R-tree of O.

HEURISTIC 3. *Let $\mathcal{C}_O$ be an O-safe collection, and N be a node in the R-tree of P'. Prune N, if its exclusive dominance area (with respect to S_{pes}) does not intersect any entry in $\mathcal{C}_O$.* □

For instance, let O be the dot-dataset in Figure 4c. Assume that we have already reported the reverse skyline point o_7, and are keeping an O-safe collection $\mathcal{C}_O = \{M_5\}$ (see Figure 6c). Then, N_4, as shown in Figure 6c, can be eliminated, because its exclusive dominance area (the shaded regions) is disjoint with M_5. This is reasonable, because accessing N_4 can reduce the ADA of the pessimistic skyline S_{pes} at most by the MBR of N_4; the reduction is not useful, as no un-reported reverse skyline point can appear in the reduced area. Note that this phenomenon justifies Heuristic 1. Specifically, imagine that M_4 (the MBR containing o_7; see Figure 6a) has not been explored; thus, it must be in $\mathcal{C}_O$, in which case N_4, intersecting M_4, cannot be pruned.

HEURISTIC 4. *Let N be a node in the R-tree of P'. Prune N, if N is disjoint with the ADA of S_{pes}.* □

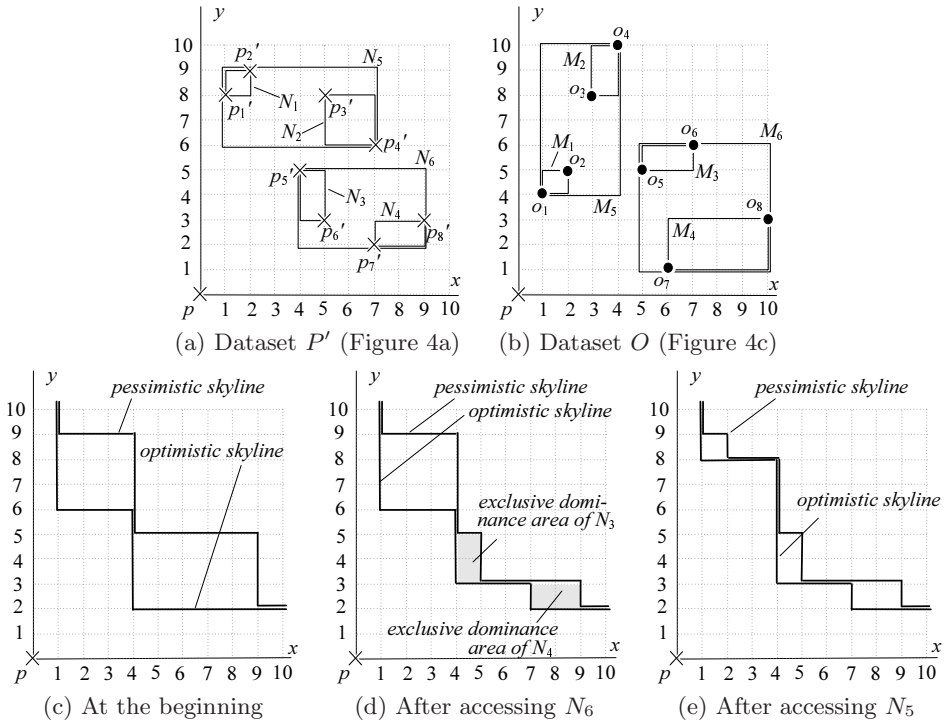


Figure 7: Illustration of *BRS*

By this rule, given the S_{pes} in Figure 6d, N_2 (an MBR in the tree of P'; see Figure 4a) can be pruned. Apparently, as N_2 does not intersect the ADA of S_{pes}, it must be disjoint with the ADA of S_{sky}. Hence, no point in N_2 can appear in the final S_{sky}.

We close this section by clarifying two fundamental operations. Given a skyline S and a rectangle N, the first operation checks whether N intersects the ADA of S, as is needed in Heuristics 1, 2, and 4. The answer is no, if and only if the min-corner of N is dominated by a point in S, which can be easily decided in $O(|S|)$ time. Given a skyline S, a point A, and a rectangle N, the second operation decides whether N intersects the exclusive dominance area of A (i.e., the portion of the data space dominated by A, but not by any point in S), as is required in Heuristic 3. This problem is settled by [37] in $O(|S|)$ time.

4.3 The Algorithm

In this section, we will put all the heuristics together into a complete algorithm *BRS*. Let us first provide a general description, before illustrating the algorithm with an example. *BRS* starts by initializing a P-safe collection $\mathcal{C}_P$ (O-safe collection $\mathcal{C}_O$) that includes all the entries in the root of the R-tree of P' (O). All the entries in $\mathcal{C}_P$ are marked as *live*. At any time, the algorithm maintains the optimistic and pessimistic skylines S_{opt} and S_{pes}, according to the current $\mathcal{C}_P$. The subsequent execution of *BRS* is in *iterations*, until $\mathcal{C}_O$ becomes empty. Specifically, each iteration performs four *tasks*:

(Task 1) Applying Heuristic 1, as long as any entry $e \in \mathcal{C}_O$ intersects the ADA of S_{opt}, *BRS* removes e, and (i) if e is a data point in O, reports it directly; (ii) otherwise, accesses its child node, and adds all the entries there to $\mathcal{C}_O$. When the first task is done, all entries in $\mathcal{C}_O$ are disjoint with the ADA of S_{opt}.

(Task 2) Remove as many entries from $\mathcal{C}_O$ as possible by Heuristic 2.

(Task 3) Based on Heuristics 3 and 4, mark all the prunable entries in $\mathcal{C}_P$ as *dead*. Whenever a dead entry is not contributing to S_{opt} and S_{pes}, it is discarded from $\mathcal{C}_P$. (In case a dead entry still defines part of S_{opt} or S_{pes}, it is kept in $\mathcal{C}_P$.)

(Task 4) If $\mathcal{C}_P$ has at least one live entry, *BRS* removes the live entry $e \in \mathcal{C}_P$ having the smallest mindist to the query product p (we use Manhattan distance), breaking ties arbitrarily. In case e is a leaf point, it is included in S_{sky} (the skyline of P'); otherwise, *BRS* fetches its child node, and places all the entries there in $\mathcal{C}_P$, which are marked as live.

Example. We illustrate *BRS* using the running example in Figure 4. Since the datasets P and O will be frequently referenced, we duplicate them, as well as the MBRs, in Figures 7a and 7b respectively. At the beginning, $\mathcal{C}_P = \{N_5, N_6\}$ and $\mathcal{C}_O = \{M_5, M_6\}$. Figure 7c demonstrates the optimistic skyline S_{opt} and pessimistic skyline S_{pes} decided by $\mathcal{C}_P$.

The algorithm starts the first iteration. In Task 1, since M_5 (see Figure 7b) intersects the ADA of S_{opt} (Figure 7c), M_5 is accessed (Heuristic 1), and its entries M_1, M_2 are added to $\mathcal{C}_O$. Similarly, as M_1 intersects the ADA of S_{opt}, it is also visited, fetching o_1, o_2, which are output as reverse skyline points immediately (Heuristic 1). Now, $\mathcal{C}_O = \{M_2, M_6\}$, and Task 1 continues. It explores M_6, then M_4, and outputs o_7, since they intersect or fall in the ADA of S_{opt} (Heuristic 1). At this point, Task 1 finishes with $\mathcal{C}_O = \{M_2, M_3, o_8\}$.

By Heuristic 2, Task 2 evicts o_8 and M_3 from $\mathcal{C}_O$, because they fall outside the ADA of S_{pes} (Figure 7c); $\mathcal{C}_O$ has only a single remaining entry $\{M_2\}$. Task 3 prunes nothing from $\mathcal{C}_P$, and thus, the current iteration enters Task 4. *BRS* accesses N_6, as its mindist 6 from the query p is smaller than the mindist 7 of the other entry N_5 in $\mathcal{C}_P$ (see Figure 7a). Replacing N_6 with N_3 and N_4 (found in N_6) in $\mathcal{C}_P$, the first iteration is completed with $\mathcal{C}_P = \{N_3, N_4, N_5\}$. Accordingly, the updated optimistic and pessimistic skylines S_{opt} and S_{pes} are as shown in Figure 7d.

In the second iteration, Task 1 has no effect, because M_2 (the only entry in $\mathcal{C}_O$) is disjoint with the ADA of S_{opt}. Task 2 does nothing, since M_2 still intersects the ADA of S_{pes}. Task 3 marks N_3 and N_4 dead, because their exclusive dominance regions (the shaded regions of Figure 7d) do not intersect M_2 (Heuristic 3). *BRS* permits N_3 and N_4 to stay in $\mathcal{C}_P$, as they still determine part of S_{opt} (in fact, also S_{pes}). Task 4 explores the only live entry N_5 in $\mathcal{C}_P$, which becomes $\{N_1, N_2, N_3^*, N_4^*\}$, where the asterisks indicate that N_3 and N_4 are dead. Figure 7e gives the current S_{opt} and S_{pes}.

The third iteration starts. Again, Task 1 has no effect, but Task 2 eliminates M_2 from $\mathcal{C}_O$, as it is outside the ADA of S_{pes} (Heuristic 2). Since $\mathcal{C}_O$ is empty, the algorithm terminates with $\{o_1, o_2, o_7\}$ as the final reverse skyline. □

So far our discussion concentrates on the first quadrant of the query p. To tackle all the quadrants, we execute 2^d threads of *BRS*, each dealing with a quadrant in a symmetric manner. These threads share a common memory buffer, to avoid reading the same node from the disk twice. There is a minor detail worth mentioning. In our presentation of *BRS*, we consider that each node N of the R-tree of P falls entirely in the first quadrant of p. When this is not true, N is handled as follows. If N is disjoint with the quadrant, it is simply ignored. In case N partially intersects the first quadrant, we take its intersection $\hat{N}$ with the quadrant, and transform $\hat{N}$ to an MBR $\hat{N}'$ on P' (by the midway conversion in Section 4.1). Then, *BRS* processes $\hat{N}'$ normally except that, in calculating the pessimistic skyline, $\hat{N}'$ is skipped (i.e., it contributes no minmax-corner).

The next lemma shows that *BRS* never entails higher I/O cost than *basic*:

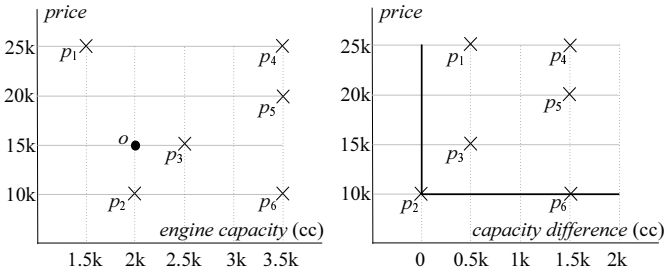
LEMMA 2. *If a node (from the tree of P or O) is visited by* BRS*, it is also accessed by* basic.

PROOF: By symmetry, it suffices to discuss the nodes, in the R-trees of P' (obtained from P by midway conversion) and O, that intersect a quadrant Ω_{quad} of p. First, we prove that if a node M from the R-tree of O is visited by *BRS*, it is also accessed by *basic*. In fact, due to the way Task 1 works, M must intersect the ADA of S_{opt}, at the time it is visited by *BRS*. Hence, M also intersects the ADA of S_{sky} (which always contains the ADA of S_{opt}), and therefore, is visited by *basic*.

Second, we show that any node (from the tree of P') visited by *BRS*, is also accessed by the P-step of *basic*, which is essentially the *BBS* algorithm. Assume that a node N (in the tree of P') is visited by *BRS*, but not by *BBS*, implying that a point $\hat{p} \in S_{sky}$ dominates N in Ω_{quad}. Consider the *accessing moment* when N is fetched by *BRS*. At this moment, by Definition 4, either $\hat{p}$ or one of its ancestors is in $\mathcal{C}_P$. If $\hat{p}$ is in $\mathcal{C}_P$, however, N should have been marked dead in Task 3, and cannot have been accessed in Task 4. In the sequel, we consider that $\mathcal{C}_P$ includes an ancestor e of $\hat{p}$.

e is dead at the accessing moment; otherwise, e would have replaced N as the node visited at Task 4, because it is closer to p than N. Further, e cannot have been marked dead by Heuristic 4, as it intersects the ADA of S_{sky} (and thus also S_{pes}). Therefore, e is marked as dead by Heuristic 3, at which moment its exclusive dominance region (EDR) intersects no entry in $\mathcal{C}_O$. Since both the EDR of e and $\mathcal{C}_O$ monotonically shrink, the EDR of e must still intersect nothing in $\mathcal{C}_O$ at the accessing moment. The EDR of e encloses the EDR of N at any time in *BRS*, because the min-corner of e dominates that of N. It follows that the EDR of N does not intersect anything in $\mathcal{C}_O$ either at the accessing moment. Hence, N should have been marked dead as well, contradicting the fact that it is visited. □

The opposite of the lemma, however, is not true, namely, *BRS* may avoid accessing nodes visited by *basic*. We have already seen evidence earlier: while *basic* explores all the nodes in the tree of Figure 7a except N_2 (explained in Section 4.1), *BRS* fetches only the root, N_5, and N_6 of that tree.



(a) The raw space (b) The transformed space

Figure 8: A dynamic skyline with bi- and uni-directional dimensions

5. EXTENSION

Sometimes bichromatic reverse skylines may return awkward results. To illustrate, let us replace the vertical axis of Figure 1a with *price*, but still use its horizontal axis *capacity*. Figure 8a presents the new dataset. As in Figure 1a, o denotes the profile of a customer, who here prefers a car with a 2000-cc engine at the price of 15k dollars. To offer recommendations, the dealer computes the dynamic skyline of o. If the computation follows the transformation in Figure 1, the resulting skyline is $\{p_2, p_3\}$, making the dealer believe that the two vehicles are incomparable, and both of them attract o. This, however, is unreasonable because p_2 is actually better than p_3 — p_2 more closely matches the capacity-requirement of o, and yet, is cheaper than p_3.

To fix this problem, the dealer converts the vehicles to a new space in Figure 8b. Similar to Figure 1b, the horizontal coordinate of a transformed p_i ($1 \leq i \leq 6$) equals the difference between the capacities of p_i and o. Unlike Figure 1b, however, the vertical coordinate of the converted p_i captures the original price of p_i *directly* (as opposed to its difference from o). The skyline in the transformed space contains a single vehicle p_2, which is indeed the best vehicle for o. The dynamic skyline of o should be $\{p_2\}$. Accordingly, o is a reverse skyline point of p_2.

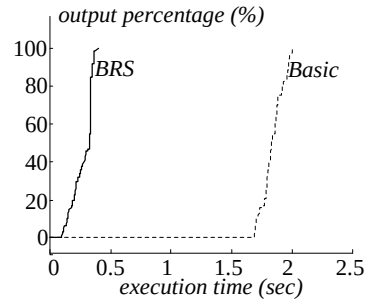
The different treatments on *price* and *capacity* (in computing dynamic skylines) are necessary because, unlike *capacity*, a customer always prefers a cheaper vehicle, rather than a vehicle whose price is closer to her/his preference. In fact, the price 15k is redundant in the customer profile o. We should simply minimize the price no matter what the profile says.

The above observation justifies the classification of dimensions into two categories: *bi-directional* and *uni-directional*. Specifically, a bi-directional attribute, such as *engine-capacity* and *seat-number*, does not have an obvious optimization direction. Hence, the dynamic skyline should minimize the absolute difference between a vehicle and a customer profile on the attribute. On the other hand, a uni-directional attribute, e.g., *price* and *mileage*, has a clear preferential direction. Thus, the dynamic skyline should optimize that direction as much as possible. The concept of bichromatic reverse skyline is applicable for any mixture of dimensions of each category. This is because, reverse skylines are well-defined, as long as the corresponding dynamic skylines can be properly formulated.

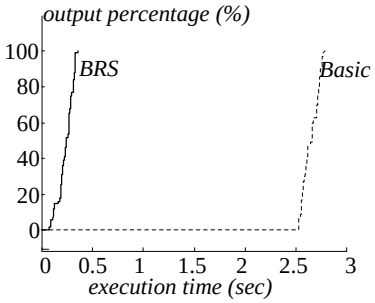
No previous work has addressed the difference between uni- and bi-directional dimensions before. In the sequel, we present a neat reduction that converts any bi-directional algorithm to retrieve the reverse skyline in a data space Ω with any mixture of uni- and bi-direction dimensions. Without loss of generality, assume that Ω has d_b bi-directional dimensions A_1, ..., A_{d_b}, and d_u uni-directional dimensions B_1, ..., B_{d_u}. We use Ω_b to denote the d_b-dimensional *bi-directional space* formed by A_1, ..., A_{d_b}, and similarly, Ω_u to denote the d_u-dimensional *uni-directional space* created by B_1, ..., B_{d_u}. Given a product (customer) dataset P (O) and a product $p \in P$, the objective is to find all the customers in the reverse skyline of p.

Our reduction first transforms O into an alternative dataset O^* as follows. For each customer $o \in O$, we add a point o^* to O^* such that $o^*[A_i] = o[A_i]$ for all $i \in [1, d_b]$, and $o^*[B_j] = 0$ for all $j \in [1, d_u]$, namely, o^* inherits all the bi-directional attributes of o, but its coordinate on each uni-directional dimension equals the best value 0. Then, we deal with a purely bi-directional problem: given a product set P and customer set O^*, retrieve the reverse skyline of p in Ω, *treating all dimensions (i.e., A_1, ..., A_{d_b}, B_1, ..., B_{d_u}) as bi-directional*. This reverse skyline is exactly the reverse skyline of p in the original problem.

The above reduction enables both *basic* and *BRS* (in Sections 3 and 4.3 respectively) for generic reverse skyline search, no matter how many dimensions in Ω are bi-/uni-directional. However, a special feature of O^* suggests a further optimization of *BRS*. Specifically, as all the points in O^* have coordinate 0 on the uni-directional dimensions B_1, ..., B_{d_u}, fewer threads are required in *BRS*. More precisely, recall that the query product p divides Ω into 2^d quadrants, each of which can be represented as a conjunction of $d = d_b + d_u$ clauses: $(A_1 \diamond p[A_1]) \wedge ... \wedge (A_{d_b} \diamond p[A_{d_b}]) \wedge (B_1 \diamond p[B_1]) \wedge ... \wedge (B_{d_u} \diamond p[B_{d_u}])$, where each $\diamond$ can independently be either $\leq$ or $\geq$ (e.g., $A_1 \leq p[A_1]$ means that every A_1-coordinate of the quadrant is at most $p[A_1]$). In fact, we only need to consider 2^{d_b} quadrants, where the $\diamond$ is fixed to $\leq$ on all the d_u uni-directional axes. Namely, those quadrants have the form $(A_1 \diamond p[A_1]) \wedge ... \wedge (A_{d_b} \diamond p[A_{d_b}]) \wedge (B_1 \leq p[B_1]) \wedge ... \wedge (B_{d_u} \leq p[B_{d_u}])$. This is correct, because none of the other quadrants can contain any point in O^*. Therefore, 2^{d_b} threads are sufficient in *BRS*.



(a) The query with the most reverse skyline points in Figure 9d



(b) The query with the most reverse skyline points in Figure 9e

Figure 10: Progressiveness comparison (cardinality = 100k, dimensionality = 2)

6. EXPERIMENTS

This section experimentally compares the proposed algorithm *BRS* against the only existing solution *basic* (discussed in Section 3). Our objective is to verify that *BRS* outperforms *basic* in both execution time and progressiveness.

The data space is normalized to have a unit range $[0, 1]$ on every dimension. We deploy datasets of three typical distributions in the skyline literature: *uniform*, *correlated*, and *anti-correlated*. In a uniform dataset, every point is randomly distributed in the data space. In a correlated dataset, if a point has a low value on a dimension, very likely it also has a small value on the other dimensions. Conversely, in an anti-correlated dataset, if a point has a low value on a dimension, it tends to have a large value on other dimensions. Generation of these data follows the description in [3]. Each dataset is indexed with an R*-tree [2] with 4k page size. All the experiments are executed on a machine running an Intel CPU at 2.13GHz with 1 Giga bytes memory.

The two underlying datasets P and O always have the same cardinality. The distributions of P and O can independently be uniform, correlated, and anti-correlated respectively, resulting in totally 9 combinations.

Each query point p is randomly sampled from the underlying P. We examine the efficiency of an algorithm by using it to answer a *workload* of 100 queries. Unless specifically stated, all the dimensions are bi-directional.

Efficiency. The first set of experiments compares the efficiency of *basic* and *BRS* under different data distributions. All the datasets used are two-dimensional and have cardinality 100k. Figure 9a shows the average workload cost of

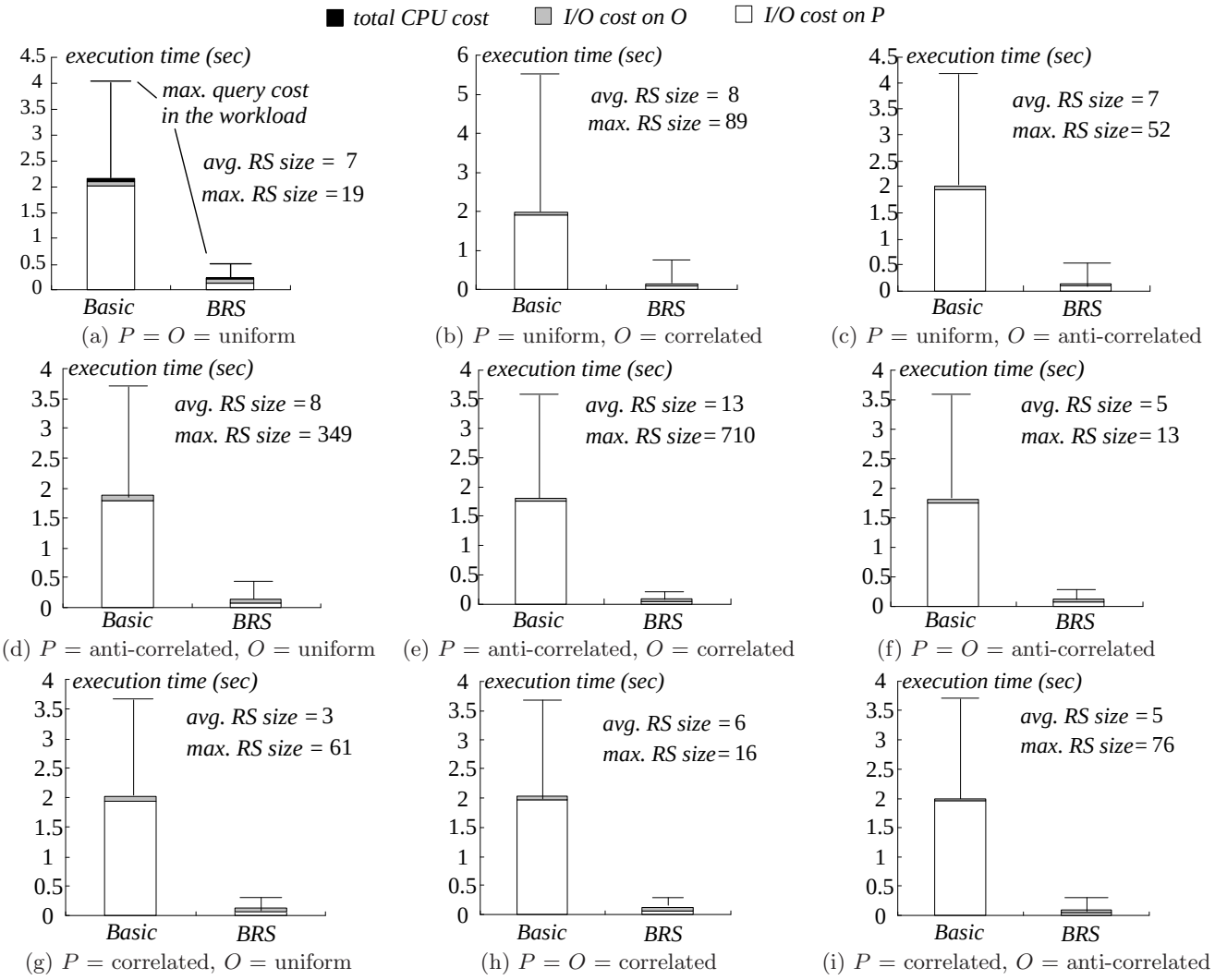


Figure 9: Efficiency comparison (cardinality 100k, dimensionality 2)

the two methods when the distributions of P and O are both uniform. Each average cost is further broken down into three parts, including the CPU time, the I/O overhead incurred in accessing P, and the I/O overhead on O, respectively. The bar above a column indicates the total cost of the most expensive query in the workload. Furthermore, the figure also provides the average (maximum) number 7 (19) of reverse skyline points per query.

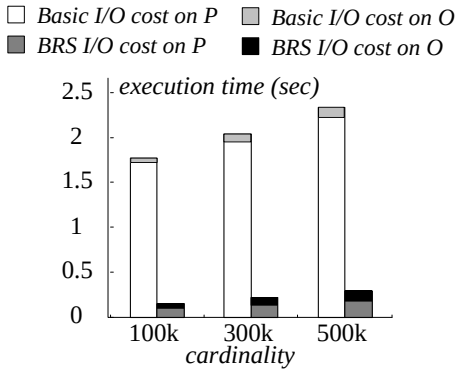
Evidently, *BRS* significantly outperforms *basic*. In particular, even the most costly query of *BRS* is much cheaper than the average overhead of *basic*. Due to the reason explained in Section 4.1, the two methods entail the same I/O cost in accessing the R-tree of O (i.e., the grey boxes of the two columns in Figure 9a have the same height). The advantage of *BRS* is reflected in its considerably smaller I/O overhead on the R-tree of P, confirming the effectiveness of our heuristics in Section 4.2. It is worth mentioning that both algorithms are heavily I/O-bounded. Since this is true in the rest experiments, we omit the CPU cost for better clarity.

Figures 9b-9i demonstrate the results of the experiment when the distributions of P and O take other combinations (e.g., P = uniform and O = correlated in Figure 9b). These results confirm the observations mentioned earlier.

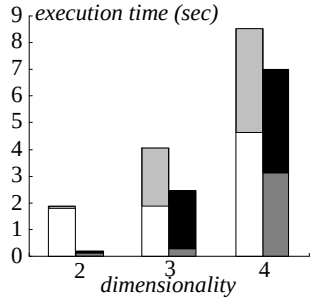
Observe that, at each row of Figure 9, *basic* incurs roughly same I/O cost on P (e.g., about 2 seconds in Figures 9a, 9b and 9c). This is expected because this cost depends solely on the distribution of P, which is identical in all three diagrams of each row. Furthermore, in each workload, the average reverse skyline size is much lower than the maximum size. This is due to the fact that some queries have no reverse skyline point. Note that such queries are crucial in practice because they reveal the bad products in P that do not attract any customer in O at all.

Progressiveness. We proceed to evaluate the progressive behavior of *basic* and *BRS* in outputting reverse skyline points. As progressiveness is important only for queries that return many results, we select the workloads in Figures 9d and 9e because they have the largest average reverse skyline sizes. From each workload, the query with the most sizable reverse skyline set is chosen as the representative.

Figure 10a (10b) demonstrates the progressiveness of the representative query in Figure 9d (9e), which returns 349

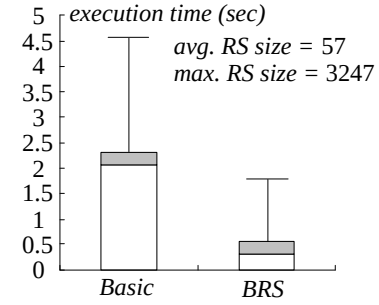


(a) Vs. cardinality (2D)

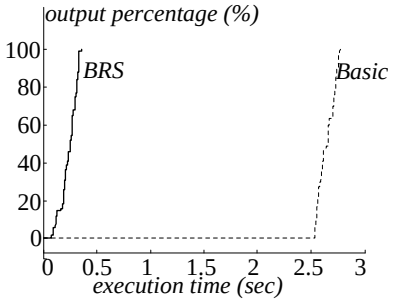


(b) Vs. dimensionality (100k cardinality)

Figure 11: Scalability comparison (P = anti-correlated, O = correlated)



(a) Efficiency



(b) Progressiveness of the query with the most reverse skyline points in (a)

Figure 12: Performance on a dataset with both uni- and bi-directional dimensions (P = anti-correlated, O = correlated)

(710) reverse skyline points. The figure plots the percentage of reverse skyline points reported as a function of the elapsed time. *BRS* achieves significantly better progressiveness. In particular, *BRS* starts producing results soon after beginning. In contrast, *basic* does not output anything until nearly the end of its execution. This is not surprising because *basic* cannot report any result until its P-step is completed — this is why the curve of *basic* stays at 0% for a long time. In each figure, the curve of *BRS* is shorter, because *BRS* incurs less total overhead than *basic* (see Figure 9).

Scalability. The next experiments inspect the scalability of *basic* and *BRS* with the dataset cardinality and dimensionality. We present only the results of the case where the distributions of P and O are anti-correlated and correlated, respectively. The other distribution combinations exhibit similar behavior, and hence, are omitted to avoid redundancy.

To study the influence of cardinality, we use 2D P and O, and vary their cardinalities from 100k to 500k. Figure 11a presents the average workload cost as a function of cardinality (each cost is broken into the I/O time on P and O, respectively). The overhead of both algorithms increases linearly with cardinality. *BRS* entails only a fraction of the cost of *basic* in all cases. The difference between the two algorithms is even more obvious for larger datasets.

To explore the effect with dimensionality, we fix the cardinalities of P and O to 100k, but vary their dimensionalities from 2 to 4. Figure 11b plots the average query cost as a function of dimensionality. *BRS* is the clear winner. Both algorithms are slower in high-dimensional space due to two reasons. First, the number of skyline points increases with dimensionality, forcing both algorithms to access more information. Second, the structure of R-trees degrades as dimensionality grows. It is worth noting that the skyline in dimensionality higher than 4 may not be meaningful because it would contain an excessive number of points.

Mixture of Bi- and Uni-directional Dimensions. Next, we compare *BRS* against *basic* when the data space has both bi- and uni-directional dimensions. For this purpose, we take the P and O in Figure 9e (the other distribution combinations yield similar results), but change them to 3D datasets by augmenting each point with a new coordinate, which is randomly generated in $[0, 1]$. The new dimension is treated as uni-directional, and the other dimensions bi-directional.

Figure 12a demonstrates the average workload cost of *basic* and *BRS*, together with the largest single-query overhead in the workload. Apparently, *BRS* significantly outperforms its competitor. Next, we take the query in the workload with the largest reverse skyline size (3247 points), and use it to compare the progressiveness of the two algorithms. The results are shown in Figure 12b, clearly demonstrating the superiority of *BRS*.

7. CONCLUSIONS

Despite its significant importance in practice, bichromatic reverse skyline search has not been well studied. The only existing algorithm, *basic*, is designed specifically for uncertain data, where the problem is more difficult, and the variety of available heuristics is limited. Therefore, although *basic* can also be used to support precise datasets, intuitively there should be room for enhancing its efficiency by leveraging the reduced problem complexity. Motivated by this, we develop the *BRS* algorithm, which is tailor-made for pre-

cise datasets, and contains several non-trivial heuristics to reduce the I/O cost considerably. It can be theoretically proved that *BRS* never incurs more I/Os than *basic*. Experiments show that *BRS* indeed outperforms *basic* by a factor up to an order of magnitude. Finally, we also tackle a general version of bichromatic reverse skyline retrieval, where the data space can have any mixture of uni- and bi-directional attributes.

Our work also initiates numerous promising directions for future work. Indeed, the concept of bichromatic reverse skyline can be applied in all the skyline-related contexts reviewed in Section 2. An interesting topic, for example, is to examine the effectiveness of this operator (in assessing product competitiveness) in high-dimensional spaces. A reverse direction is to explore the power of reverse skylines in different subspaces, and investigate why various subspaces can have different reverse skylines. Furthermore, it is equally exciting to study the behavior of reverse skylines on uncertain data [26], spatial databases [30], peer-to-peer networks [36], and so on.

Acknowledgements

This work was partially supported by CERG grants CUHK 1202/06, 4161/07, 4173/08, and 4182/06 from HKRGC.

REFERENCES

[1] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256–273, 2004.

[2] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

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

[4] C. Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD*, pages 203–214, 2005.

[5] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, pages 503–514, 2006.

[6] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.

[7] S. Chaudhuri, N. N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, page 64, 2006.

[8] B.-C. Chen, R. Ramakrishnan, and K. LeFevre. Privacy skyline: Privacy with multidimensional adversarial knowledge. In *VLDB*, pages 770–781, 2007.

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

[10] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou. Parallel distributed processing of constrained skyline queries by filtering. In *ICDE*, pages 546–555, 2008.

[11] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, pages 291–302, 2007.

[12] K. Deng, X. Zhou, and H. T. Shen. Multi-source skyline query processing in road networks. In *ICDE*, pages 796–805, 2007.

[13] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, pages 229–240, 2005.

[14] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline queries against mobile lightweight devices in manets. In *ICDE*, 2006.

[15] W. Jin, M. Ester, Z. Hu, and J. Han. The multi-relational skyline operator. In *ICDE*, pages 1276–1280, 2007.

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

[17] K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the skyline in z order. In *VLDB*, pages 279–290, 2007.

[18] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. Dada: a data cube for dominant relationship analysis. In *SIGMOD*, pages 659–670, 2006.

[19] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, pages 213–226, 2008.

[20] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, 2005.

[21] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.

[22] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, pages 267–278, 2007.

[23] M. D. Morse, J. M. Patel, and W. I. Grosky. Efficient continuous skyline computation. In *ICDE*, 2006.

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

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

[26] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

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

[28] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[29] N. Sarkas, G. Das, N. Koudas, and A. K. H. Tung. Categorical skylines for streaming data. In *SIGMOD*, pages 239–250, 2008.

[30] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, pages 751–762, 2006.

[31] M. A. Soliman, I. F. Ilyas, and N. Koudas. Finding skyline and top-k bargaining solutions. In *ICDE*, pages 1263–1267, 2007.

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

[33] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *ICDE*, 2006.

[34] A. Vlachou, C. Doulkeridis, and Y. Kotidis. Angle-based space partitioning for efficient parallel skyline computation. In *SIGMOD*, pages 227–238, 2008.

[35] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. Skypeer: Efficient subspace skyline computation over distributed data. In *ICDE*, pages 416–425, 2007.

[36] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu. Efficient skyline query processing on peer-to-peer networks. In *ICDE*, pages 1126–1135, 2007.

[37] P. Wu, D. Agrawal, Ö. Egecioglu, and A. E. Abbadi. Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation. In *ICDE*, pages 486–495, 2007.

[38] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*, pages 112–130, 2006.

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

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