

Flexible and Efficient Resolution of Skyline Query Size Constraints

Hua Lu, *Member, IEEE*, Christian S. Jensen, *Fellow, IEEE* Zhenjie Zhang *Student Member, IEEE*

Abstract—Given a set of multi-dimensional points, a skyline query returns the interesting points that are not dominated by other points. It has been observed that the actual cardinality (s) of a skyline query result may differ substantially from the desired result cardinality (k), which has prompted studies on how to reduce s for the case where $k < s$.

This paper goes further by addressing the general case where the relationship between k and s is not known beforehand. Due to their complexity, the existing pointwise ranking and set-wide maximization techniques are not well suited for this problem. Moreover, the former often incurs too many ties in its ranking, and the latter is inapplicable for $k > s$. Based on these observations, the paper proposes a new approach, called *skyline ordering*, that forms a skyline-based partitioning of a given data set, such that an order exists among the partitions. Then set-wide maximization techniques may be applied within each partition. Efficient algorithms are developed for skyline ordering and for resolving size constraints using the skyline order. The results of extensive experiments show that skyline ordering yields a flexible framework for the efficient and scalable resolution of arbitrary size constraints on skyline queries.

Index Terms—Skyline Queries, Query Processing, Database Management.

1 INTRODUCTION

Given a set of d -dimensional points, a skyline query [5] returns a subset of points that are *interesting* in that they are not dominated by other points. Point p_1 dominates p_2 if p_1 is better than p_2 in at least one dimension and no worse than p_2 in any other dimension. The skyline query is thus fundamental to multi-criteria decision making [5], [12], [18], [23], [26].

Consider the example shown in Figure 1. Each of the hotels has two attributes: room price and distance to the beach. For a tourist who prefers a low-cost hotel close to the beach, hotels A, B, C, and D are interesting—these are exactly the skyline of the hotels in the figure.

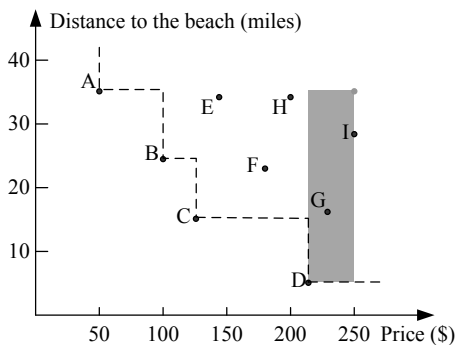


Fig. 1. Skyline of Hotels

- Hua Lu and Christian S. Jensen are with the Department of Computer Science, Aalborg University, Denmark. The work was conducted in part when Christian S. Jensen was a Visiting Scientist at Google Inc. E-mail: {luhua, csj}@cs.aau.dk
- Zhenjie Zhang is with School of Computing, National University of Singapore, Singapore. E-mail: zhenjie@comp.nus.edu.sg

Although skyline queries are effective in identifying interesting points in multi-dimensional data sets, they also have a well-known weakness—their result cardinalities may vary. The result cardinality can be very large, especially when the dimensionality is high or the data is anti-correlated.

To contend with large skyline results, four categories of approaches have been proposed. In the first category, namely *pointwise ranking*, all d -dimensional points are totally ordered according to some specific scoring function or mapping function, and only the top points are returned [7], [23]. Such approaches face the difficulty of reflecting the multiple criteria in the scoring functions. In the second category, namely *subspace reference*, all subspaces of the full d -dimensional space are investigated, and those points preferred in subspaces are favored in the result [6], [7], [30]. Such approaches incur high computation costs in the traversal of all subspaces. In the third category of approaches, namely *set-wide maximization*, a subset of the skyline is deliberately selected such that a collective quantity based objective, e.g., the number of points dominated by those in that subset, is maximized [21]. The problem has NP-hard computational complexity, so only approximate results can be computed efficiently. In the fourth category, namely *approximate selection*, points are compared approximately with a predefined threshold such that more points are identified as being dominated [17]. The approach in this category is not able to fully control the result cardinality.

It is meaningful for an approach to belong to different categories. For example, the subspace skyline frequency method [7] belongs to the first two categories. Section 2.2 offers a detailed review of previous approaches.

The above approaches focus on reducing large skylines; however, small skylines can also be a problem.

Consider again the classical example in Figure 1. The tourist now specifies a parameter k —the number of interesting hotels to be returned in the query result. A conventional skyline query usually, if not always, fails to return precisely k hotels. If k is 3, the skyline is too big. On the other hand, a large k value can offer practical convenience. Suppose a large number of hotel rooms are needed to accommodate all participants to a popular event to take place. To make sure to get enough rooms, the event organizer may specify k to be 5 or even larger, if all hotels in the skyline (namely A, B, C and D) together provide fewer vacant rooms. In such cases where skylines may be too small, the use of large k values is well motivated.

Another real-life example comes from online shopping. Many online shopping sites, such as eBay, allow users to specify multiple product dimensions (e.g., price, quality, guarantee, etc.) as criteria for good bargains. The resulting user requests are well captured by skyline queries against the background product database. Specifying only a few dimensions in a query tends to result in a skyline set of low cardinality; a user may find that such a set does not offer enough alternatives and may desire a larger set. In such cases, it makes practical sense to extend the small skyline to a larger size as expected by the user.

The skyline cardinality s of an arbitrary d -dimensional data set can be considerably smaller than a user-specified number k , and there is no good, straightforward way to increase s to k . We conclude that an approach is needed that supports skyline querying for arbitrary k .

This paper studies *size constrained skyline queries*, which takes an arbitrary size constraint k as a parameter and retrieves k desirable points from a d -dimensional data set. We assume that when a user issues such a query with a specific k , the original skyline of the data set is unknown. Therefore it is also unknown whether k is larger than, equal to, or smaller than the skyline size.

The conventional relevant techniques, pointwise ranking and set-wide maximization, are not well suited for arbitrary size constraints. Pointwise ranking approaches sometimes prefer non-skyline points to skyline points. Also, such approaches usually prefer points with similar features, since similar points are likely to get similar ranking scores, which limits the variety in query results. With set-wide maximization it is possible to return results with larger variety, but set-wide maximization is not applicable to arbitrary query size constraints—cases where $k > s$ cannot be handled. Set-wide maximization is also computationally expensive due to its counting and set-wide optimization characteristics.

Motivated by these shortcomings, we propose a new approach, called *skyline ordering*, that supports arbitrary size constraint over the conventional skyline query. Skyline ordering introduces a skyline-based partitioning of a given data set, it provides an ordering among the partitions, and it reserves room for the use of various set-wide maximization techniques within partitions.

Skyline ordering aims to overcome the limitations of pointwise ranking and set-wide maximization by combining them into a uniform framework. In particular, given the partitions in skyline order, the following hold: (1) no point can dominate any other point in the same partition or in a previous partition; (2) any point in a partition, except in the first partition, must be dominated by some point(s) in the previous partition.

When answering a size constrained skyline query with skyline ordering, we start from the first partition and continue to output partitions until at least k points have been output; the last partition is pruned to return exact k points as the result. This way, arbitrary size constraints on skyline queries can be computed efficiently within a flexible framework built upon skyline ordering.

Our approach differs from skyband-based ranking[27]. The latter first retrieves from the database a K -skyband [24], a set of K points dominated by at most $K - 1$ other ones, that has the smallest $K \geq k$. Then it returns the k best points from the K -skyband. In contrast, the partitioning in our approach neither counts nor minimizes the numbers of dominators. When selecting k points as the result, our approach may involve several consecutive partitions rather than a single K -skyband.

The paper makes the following contributions. First, its proposal supports arbitrary size constraints on skyline queries, which facilitates both reducing and expanding an original skyline. Second, it includes a comprehensive review of previous approaches that modify the cardinality of a skyline, it analyzes their application to arbitrary skyline size constraints, and it proposes a simple yet efficient estimation-based method for selecting representative skyline points from a large skyline. Third, it proposes the skyline ordering concept, together with algorithms for computing skyline orders. Fourth, it defines size constrained skyline queries based on skyline order and develops various query processing algorithms. Fifth, the paper offers results of empirical evaluations of skyline order computation and size constrained skyline query processing.

The rest of this paper is organized as follows. Section 2 gives a general problem definition of size constrained skyline queries and briefly reviews related work. Section 3 discusses different approaches for tuning skyline cardinality. Section 4 covers the skyline order definition and computation. Section 5 applies skyline ordering to size constrained skyline queries. Section 6 presents the empirical studies. Section 7 concludes the paper.

2 PRELIMINARIES

2.1 Problem Definition

Given a d -dimensional data set P with cardinality N , each data point p represents a choice of interest to a user. Without loss of generality, we assume that the user interests cover all d dimensions and that smaller values are preferred in each dimension. For two points p and q

in P , we use $p \prec q$ to represent that p dominates q under the conventional skyline definition [5].

We may define a *size constrained skyline query* $Q_k^{scs}(P)$ informally as a subset S of P consisting of k points that are *good* in terms of user interest. In this informal definition, the query parameter k explicitly specifies the cardinality of the result to be returned. But what it means for a k -point subset of P to be *good*, and in what sense one subset of points is *better* than another, remain to be formalized. Put differently, we need criteria to determine which skyline points to discard or which non-skyline points to include when necessary. By applying such precise criteria to the result point selection, we are able to give formal definitions. These issues are covered in Sections 3 and 5.

Table 1 lists the notation used in this paper.

TABLE 1
Table of Notations

Notation	Description
P	Argument data set
N	Cardinality of P
d	Dimensionality of P
\mathcal{R}^d	d -dimensional space
S_P	Skyline of P
s	Size of S_P
$Q_k^{scs}(P)$	A k -size constrained skyline query on P
\mathcal{S}_P	Skyline order of P
S_i	i -th skyline order subset of S_P
n	Skyline order length
$D_P^{\prec}(p)$	All points in P that are dominated by p
$D_P^{\prec}(S)$	All points in $P \setminus S$ dominated by point(s) in S

2.2 Related Work

2.2.1 Skyline Algorithms

Two categories of skyline algorithms exist. Algorithms in the first category do not rely on indexes on the data set. The theoretical algorithms for maximal vector computation [1], [3], [19] fall into this category. Borzanyi et al. [5] introduce the skyline query into databases, defining *Block Nested Loop* (BNL) and *Divide-and-Conquer* (D&C) algorithms. Chomicki et al. [10] propose a variant of BNL called the *Sort-Filter-Skyline* (SFS) algorithm. Godfrey et al. [12] provide a comprehensive analysis of these non-index-based algorithms and propose a hybrid method with improvements. Bartolini et al. [2] propose a pre-sorting based algorithm that is able to stop dominance tests early. Zhang et al. [31] propose a dynamic indexing tree for skyline points (not for the data set), which helps reduce CPU costs in sort-based algorithms [10], [12], [2].

Algorithms in the second category require specific indexes. Tan et al. [26] propose two progressive algorithms: *Bitmap* and *Index*. The former represents points by means of bit vectors, while the latter utilizes data transformation and B⁺-tree indexing. Kossmann et al. [18] propose a *Nearest Neighbor* (NN) method that identifies skyline points by recursively invoking R*-tree based depth-first

NN search over different data portions. Papadias et al. [23] propose a *Branch-and-Bound Skyline* (BBS) method based on the best-first nearest neighbor algorithm [13].

2.2.2 Dominance Based Skyline Query Derivatives

Several skyline query derivatives based on dominance have been proposed. All approaches known to the authors are listed in Table 2. For each approach, we consider its methodological nature, its applicability to the case where the number of points expected (k) is less than the skyline size (s) and the opposite case ($k > s$), and its relevant algorithm. For the nature of an approach, we consider four types: *pointwise ranking*, *subspace reference*, *set-wide maximization*, and *approximate selection*.

Papadias et al. [23] propose the top- k dominating query that retrieves points that dominate the largest number of points. Yiu and Mamoulis [29] design efficient processing algorithms for this problem. The top- k dominating query belongs to the pointwise ranking category.

By utilizing the frequency of points' membership in subspace skylines, Chan et al. [7] formalize a top- k ranking problem that gives priority to points that appear more frequently in subspace skylines. This skyline frequency approach is able to return a fixed number of points for both cases. Zhang et al. [30] propose the concept of strong skyline points, which appear frequently in small-sized subspace skylines in high-dimensional spaces. This method does not return a fixed number of points in the case of $k < s$, and it is not applicable to the case of $k > s$. Chan et al. [6] propose a K -dominant¹ skyline for high dimensional space. The strict dominance covering all dimensions is relaxed to only K dimensions in any subspace. A K -dominant skyline does not return a fixed number of points in the case of $k < s$, and it is not applicable to the case of $k > s$. All the three approaches just covered use subspaces for result determination, while the skyline frequency approach can also be regarded as a total pointwise ranking.

Lin et al. [21] propose the k most representative skyline points problem (top- k RSP for short), which selects a portion of the points from the traditional skyline that maximizes the total number of dominated points. However, the results can turn out to not be representative. Based on this observation, Tao et al. [25] propose techniques that aim to minimize the distance between a non-representative skyline point and its nearest representative one in the selection. These two approaches are instances of set-wide maximization.

Koltun and Papadimitriou [17] introduce approximately dominating representatives, which produces a smaller, but not fixed-size, so-called ϵ -*ADR* skyline. In particular, before a point is compared with others, it is first enlarged by ϵ in all dimensions (larger values are assumed to be preferred). This approach is not applicable to the case of $k > s$. Xia et al. [28] define

1. This K carries a different meaning than does k in our problem definition.

TABLE 2
Previous Approaches to Dominance Based Skyline Query Derivatives

Approach Nature	Approach Name	$k < s$	$k > s$	Remarks
<i>pointwise ranking</i>	Top- k dominating query [23]	+	+	
<i>Subspace reference</i>	<i>Skyline frequency</i> [7]	+	+	Also a pointwise ranking.
	Strong skyline point [30]	—	×	
	K -dominant skyline [6]	—	×	
<i>Set-wide maximization</i>	Top- k RSP [21], [25]	+	×	
<i>Approximate selection</i>	ε -ADR dominance [17]	—	×	
	ε -skyline [28]	—	—	
	Thick skyline [16]	×	—	

+: Result of exact k points; —: Result of uncontrolled size; ×: Inapplicable.

a slightly different, so-called ε -skyline that allows user-specified weights on each dimension. By varying ε and the weights, it is possible to both increase and decrease the size of an ε -skyline. However, this arrangement offers only indirect and rough control. The skyline does not accept an explicit result size constraint k as input. Thus, the ε -skyline does not afford the user the kind of fine-grained control offered by this paper’s proposal.

Rather than evaluating single data points, Jin et al. [16] extend the conventional skyline, which they term a thin skyline, to a so-called thick skyline, by including points in the proximity of the conventional skyline points. This approach is only applicable to the case of $k > s$, and it cannot return a fixed number of points. Both of the above approaches perform approximate selections, as they employ approximate measures when evaluating point candidates.

3 SKYLINE SIZE TUNING

3.1 Pointwise Ranking

To support size constraints on skyline queries, we need criteria for point candidates that select satisfactory ones and rule out undesirable ones. In this section, we discuss options to resolve the query size constraint by ranking all points with a dominance-based scoring function.

To enable pointwise ranking of all points in P , a mapping function $f : P \rightarrow C$, is needed that maps each point p in P to a value $f(p)$ in a totally ordered domain C . As a result, all points in P can be ranked in terms of their $f(p)$ values.

The pointwise ranking approach essentially converts the multiple criteria optimization that underlies a size constrained skyline query to a top- k ranking problem. It is capable of fully controlling the result size according to arbitrary k from 0 to $|P|$, as C is totally ordered. However, this approach needs a careful design of the mapping function f , which is expected to be dominance based, and invariant to dimension scaling and shifting.

One option is to take dominating capabilities into account when defining a mapping function f . A straightforward way is to count for a point p the number of points it dominates [23]. As a result, the mapping function is $f_{DN}(p) = |D_P^<(p)|$, where $D_P^<(p) = \{q | q \in P \wedge p \prec q\}$ is the set of points from P that are dominated by p .

This definition is simple, but it sometimes prefers non-skyline points over skyline points. Referring to Figure 1, we have $f_{DN}(A) = 0$ because point A dominates nobody, $f_{DN}(F) = 2$ because F dominates H and I. Following the same line of reasoning, we have $f_{DN}(B) = 3$, $f_{DN}(C) = 5$, and $f_{DN}(D) = 2$. If a size constrained skyline query based on f_{DN} asks for $k = 4$ points, points C, B, D, and F are returned, but the skyline point A is suppressed. However, since F is dominated by C, F is unlikely to be interesting to the user. In this sense, the result actually offers fewer options compared to the result that includes A instead.

Another way to rank all points in P is to consider for each point p its frequency of appearance in the subspace skylines. The mapping function is defined as $f_{SSF}(p) = |\mathcal{R}' \in 2^{\mathcal{R}^d} \mid p^{\mathcal{R}'} \in S_{P^{\mathcal{R}'}}|$. Here $p^{\mathcal{R}'}$ is the projection of p on the subspace \mathcal{R}' , and function $P^{\mathcal{R}'}$ is the projection of all points in P onto \mathcal{R}' . Intuitively, $f_{SSF}(p)$ counts the number of subspaces on which p locates on the skyline of the data set P .

However, this method requires a traversal of all subspaces of \mathcal{R}^d , computing the skyline of P ’s projection on each subspace, and counting subspace frequency for each point in P . An approximate counting algorithm exists [7] that has an upper bound time complexity of $\mathcal{O}(2d^2 \ln(2/\delta)/\epsilon^2)$, when the approximate result is within an error of ϵ with a confidence level $\geq 1 - \delta$.

3.2 Set-Wide Maximization

Set-wide maximization is also an alternative for evaluating size constrained skyline queries. Instead of considering each single point separately, this method considers a set of points collectively to maximize a target value.

3.2.1 Count Based Maximization

The top- k representative skyline points problem [21] is limited to the case where $k < |S_P|$. This definition can be extended by modifying the outcome for the case where $k \geq |S_P|$. A direct extension is given as follows.

Definition 3.1: (Extended Top- k Representative Skyline Points Query) The extended top- k representative skyline points query $Q_k^{ersp}(P)$ retrieves a subset S of k points from a d -dimensional point set P that maximizes $|D_P^<(S)|$,

where $D_P^{\prec}(S)$ is the set of points from $P \setminus S$ dominated by some point in S .

For the extended definition above, we have the following lemma that explicitly states the relationship between the usual skyline and the result of query $Q_k^{ersp}(P)$.

Lemma 3.1: Given the skyline S_P of a d -dimensional point set P , and the result S_{ersp} of an extended top- k representative skyline points query $Q_k^{ersp}(P)$, the following properties hold:

- 1) $S_P \subset S_{ersp}$, if $k > |S_P|$;
- 2) $S_{ersp} \subset S_P$, if $k < |S_P|$.

Proof. 1. As $|S_{ersp}| = k > |S_P|$, there must exist at least one non-skyline point p s.t. $p \in S_{ersp}$ but $p \notin S_P$. Suppose there exists a skyline point s s.t. $s \in S_P$ but $s \notin S_{ersp}$. If s dominates p , $(S_{ersp} \setminus \{p\}) \cup \{s\}$ is a better result than S_{ersp} , in terms of the number of points dominated. If s does not dominate p , there must exist another skyline point s' that dominates p . If $s' \notin S_{ersp}$, $(S_{ersp} \setminus \{p\}) \cup \{s'\}$ is a better result than S_{ersp} ; otherwise, $(S_{ersp} \setminus \{p\}) \cup \{s\}$ is better since $|D_P^{\prec}((S_{ersp} \setminus \{p\}) \cup \{s\})| \geq |D_P^{\prec}(S_{ersp})| + 1 > |D_P^{\prec}(S_{ersp})|$. Each case leads to a contradiction, which proves that $S_P \subset S_{ersp}$ if $k > |S_P|$. 2. Suppose there is a $p \in S_{ersp}$ but $p \notin S_P$. There must be an $s \in S_P$ that dominates p . If $s \notin S_{ersp}$, $(S_{ersp} \setminus \{p\}) \cup \{s\}$ is a better result than S_{ersp} , since $|D_P^{\prec}((S_{ersp} \setminus \{p\}) \cup \{s\})| \geq |D_P^{\prec}(S_{ersp})| + 1 > |D_P^{\prec}(S_{ersp})|$. If $s \in S_{ersp}$, for any skyline point $s' \notin S_{ersp}$ (such a s' must exist as $|S_{ersp}| < |S_P|$), $(S_{ersp} \setminus \{p\}) \cup \{s'\}$ is a better result than S_{ersp} since $|D_P^{\prec}((S_{ersp} \setminus \{p\}) \cup \{s'\})| \geq |D_P^{\prec}(S_{ersp})| + 1 > |D_P^{\prec}(S_{ersp})|$. Both cases lead to a contradiction, which proves that $S_{ersp} \subset S_P$ if $k < |S_P|$. ■

Lemma 3.1 indicates how to answer an extended top- k representative skyline points query: its result is contained in the skyline if less points are expected, or it contains the skyline if more points are expected. However, the lemma also indicates that the direct extension given in Definition 3.1 is problematic when k is larger than the original skyline size s . For that case, $D_P^{\prec}(S)$ actually can never exceed $D_P^{\prec}(S_P)$ because every point in $P \setminus S_{ersp}$ must be dominated by some skyline point in S_P . It is shown by Lemma 3.1 that S_P is always contained in the query result S_{ersp} , but it makes no difference which non-skyline points are included into S_{ersp} , as none of them increases $|D_P^{\prec}(S_{ersp})|$. Thus, it is attractive to modify Definition 3.1 so that it becomes more meaningful for an arbitrary k . This will be discussed in Section 4.

3.2.2 Estimation Based Maximization

It is expensive to count exactly how many points are dominated by a point or a set of points [21]. In this section, we propose another result criterion based on dominating capability estimation. Such estimations take into account the dominating region [14] of each point and can be used for queries with $k < s$.

The dominating region of a point p is a hyper-rectangle, whose main diagonal is the line segment

from p to the maximum corner of P . As any point in this dominating region is definitely dominated by p , its volume, termed as $VDR(p)$, is used to estimate p 's capability of dominating other points.

To illustrate, consider hotel D in Figure 1. Its dominating region is a rectangle as there are only two dimensions. For all hotels, \$250 is the maximum price and 35 miles is the longest distance to the beach. These two maximum values together correspond to the (virtual) maximum corner that together with D defines the main diagonal of D's dominating region. Hotels G and I are within that region and are thus dominated by hotel D.

For a set of points, we consider their overall dominating capability which is indicated by the union of all their dominating regions. By applying the Inclusion-Exclusion principle, we can compute the volume of such a union of all points in a set $S = \{s_1, s_2, \dots, s_k\}$, with U_l denoting the maximum value on dimension l , as follows.

$$VDR(S) = \sum_{i=1}^k VDR(s_i) + \sum_{j=2}^k ((-1)^{j-1} \sum_{1 \leq i_1 < \dots < i_j \leq k} \prod_{l=1}^d (U_l - \max(s_{i_1}.a_l, \dots, s_{i_j}.a_l))) \quad (1)$$

Formula 1 only refers to k points in the set S , without any other global information like the number of dominated points and the subspace skyline frequency. However, an optimization based on Formula 1, i.e., selection of a subset S from P with maximum $VDR(S)$, is still not cheap when k is very large, whose time complexity is $O(\binom{2e|P|}{k})^k$ [11]. Specific heuristics can be used to approximately compute the $VDR(S)$.

One straightforward option is to select k points with the maximum sum of all VDR values. A greedy option maximizes the sum of point distances at each step when a new point is included. As the straightforward method usually obtains approximations with better precision [11], it is used in the experimental studies.

3.3 Discussion

The existing approaches exhibit limitations when attempting to use them for resolving arbitrary size constraints on skyline queries.

The use of pointwise ranking approaches causes three difficulties. First, a total ranking tends to deconstruct the essence of multi-criteria optimization when it converts a complex problem into a simple one. As pointed out in Section 3.1, if we count dominating points only, non-skyline points may be preferred over skyline points. This also happens in subspace skyline frequency based ranking, where a non-skyline point can have an advantage in more subspaces than has a skyline point.

Second, a total ranking can incur too many ties, which invalidates the ranking. For example in Figure 1, we cannot differentiate D and F (or E and G) by counting dominating points. For an arbitrary data set, such ties are very likely to exist. Ties also occur in the subspace

skyline frequency based ranking, especially when the dimensionality is not large enough.

Third, a total ranking is time-consuming especially when the data set is large. Hence, complex index structures [29] or approximate algorithms [7] are employed.

Next, the count based set-wide maximization approach is not applicable to arbitrary size constraints on skyline queries because it is unable to handle the case where more points than the skyline cardinality are expected. Moreover, the count based set-wide maximization approach is computationally very expensive due to its counting and set-wide optimization characteristics.

It makes sense to apply pointwise ranking approaches and set-wide maximization approaches in combination. The former is focused on judging whether any individual point is good enough to be included in the result. The latter is concerned with maximizing the collective advantage of a number of result points. The two approaches thus tackle the same problem from opposite directions: bottom-up and top-down. An interesting question thus arises: Can we find some novel way in-between the two to overcome the aforementioned obstacles in supporting arbitrary size constrained skyline queries? Our answer is positive. In the next two sections, we cover our framework based on the concept of *skyline order* which ranks points partially and in batches, rather than fully, one point by one point.

4 SKYLINE ORDER

4.1 Definitions and Properties

Definition 4.1: (Skyline Order) The *skyline order* of a set of d -dimensional points P is a sequence $S = \langle S_1, S_2, \dots, S_n \rangle$ defined as follows:

- 1) S_1 is the skyline of P , i.e., S_P ;
- 2) $\forall i, 1 < i \leq n$, S_i is the skyline of $P \setminus \bigcup_{j=1}^{i-1} S_j$.
- 3) $\bigcup_{i=1}^n S_i = P$

We call each S_i a *skyline subset* in the skyline order or a *skyline order subset*, and we call n the *skyline order length*, i.e., the number of subsets in S . The definitions iteratively removes the current skyline starting with P until all points in P belong to a skyline subset, resulting in n skyline subsets are obtained. By construction, the points in S_i are incomparable and $S_i \cap S_j = \emptyset$ for $i \neq j$.

For convenience, we define *skyline order index* for any point p in P as follows.

Definition 4.2: (Skyline Order Index) Given the skyline order S_1, S_2, \dots, S_n of a d -dimensional point set P , the *skyline order index* of a point $p \in P$ is i if $p \in S_i$.

The skyline order has some interesting properties that can be inferred directly from the definition.

Lemma 4.1: Given the skyline order S_1, S_2, \dots, S_n of a d -dimensional point set P , the following properties hold:

- 1) $\forall i > 1 \forall p_1 \in S_i \exists p_2 \in S_{i-1} (p_2 \prec p_1)$.
- 2) $\forall i > 1 \forall p_1 \in S_i \forall j < i \exists p_2 \in S_j (p_2 \prec p_1)$.
- 3) $\forall p_1 \in S_i \forall j > i \nexists p_2 \in S_j (p_2 \prec p_1)$.

An example of skyline order is shown in Figure 2. Here, $S_1 = \{A, B, C, D\}$ is also the skyline of all hotels.

Next, $S_2 = \{E, F, G\}$, the skyline of subset $\{E, F, G, H, I\}$, and $S_3 = \{H, I\}$. Further, the skyline order index is 1 for any point in $\{A, B, C, D\}$, 2 for any point in $\{E, F, G\}$, and 3 for any point in $\{H, I\}$, as indicated by the corresponding numeric subscripts in the figure.

Skyline order has practical applications. Referring to Figure 2, assume that many attendees are to be accommodated by a conference. A skyline query returns the best hotels A, B, C, and D. They together, however, may provide fewer vacancies than needed. To provide enough rooms, hotels E, F and G can be easily included in the result. The skyline order is thus a systematic approach to provide results with enough vacancies.

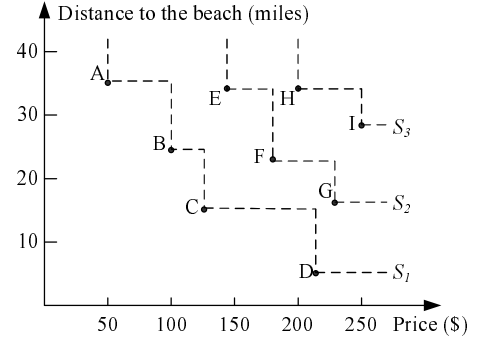


Fig. 2. Skyline Order

A notion of ranking through iterated preference is proposed in [9], which is focused on preference modeling without specific skyline order definitions or algorithms. In-place algorithms for computing layers of maxima in [4] are focused on 2-dimensional data sets only. This paper's setting poses no in-place requirement, and our algorithms are not limited to two dimensions. The idea of layered skyline has also been used in the literature as part of an indexing method for top- k queries [15]. In this paper, we focus on computing the skyline layers efficiently and on applying them in size-constrained skyline queries, which has not been studied in the literature.

Skyline order differs from the Onion technique [8]. First, we partition a data set based on the skyline concept, whereas the Onion technique is based on the convex hull definition. Second, the skyline order supports size constrained skyline queries well, as we will see in the rest of this paper, whereas the Onion technique is focused on indexing for linear optimization queries. Third, the skyline order supports *local queries*, size constrained skyline queries in our setting, without any specific hierarchical structures as in the Onion technique.

We proceed to discuss how to compute skyline order efficiently. The maintenance of skyline order in the face of point updates is addressed elsewhere [22].

4.2 Skyline Order Computation

We proceed to introduce skyline order algorithms that do not rely on indexes. In Section 6.1.3, we adapt an existing index based skyline algorithm for skyline order.

4.2.1 Basic Algorithm

The basic algorithm scans the input data set P , as described in Algorithm 1. For each point p in P , it checks p against each subset in the current skyline order and puts it into a specific subset or creates a new subset for it. The algorithm maintains all subsets of the current skyline order. While determining the membership for p , some points in the skyline subset being checked currently may also change their membership, thus producing a new subset in the skyline order.

Specifically, the skyline order \mathcal{S} is initialized to be empty (line 1). Each point p in P is compared to each point q in each S_i sequentially from S_1 to the current S_n (lines 2–16). If p is dominated by q , the algorithm breaks from the loop on the current S_i and skips to the next subset in the skyline order (lines 6–7). If p dominates q , q will be moved into a temporary list S_{tmp} that contains all those points that come from the current S_i and are dominated by p (lines 9–10). If p is found to qualify as a skyline point for the current S_i (line 11), the necessary actions are to be taken to ensure the skyline order is updated correctly, followed by breaking from the loop on \mathcal{S} (line 18). If S_i is empty, which indicates that p dominates all points previously in S_i , $\{p\}$ will be used as the new S_i and S_{tmp} will be inserted into \mathcal{S} after the new S_i (lines 12–13). Otherwise, the procedure AdjustSkyOrd will be called to adjust the subsets after S_i in \mathcal{S} (lines 15–16) in case S_{tmp} is not empty, and p will be added into S_i (line 17). If p does not belong to any S_i in \mathcal{S} , $\{p\}$ will be appended to \mathcal{S} (lines 19–20).

Algorithm 1 SkylineOrderScan (data set P)

```

1:  $\mathcal{S} = \langle \rangle$ 
2: for each point  $p$  in  $P$  do
3:   for each  $S_i$  from  $S_1$  to  $S_n$  in  $\mathcal{S}$  do
4:      $isSky = \text{TRUE}$ ;  $S_{tmp} = \emptyset$ 
5:     for each point  $q$  in  $S_i$  do
6:       if  $q \prec p$  then
7:          $isSky = \text{FALSE}$ ; break
8:       else
9:         if  $p \prec q$  then
10:          move  $q$  from  $S_i$  to  $S_{tmp}$ 
11:     if  $isSky$  then
12:       if  $S_i = \emptyset$  then
13:          $S_i = \{p\}$ ; insert  $S_{tmp}$  into  $\mathcal{S}$  after  $S_i$ 
14:       else
15:         if  $S_{tmp} \neq \emptyset$  then
16:           AdjustSkyOrd( $S_{tmp}$ ,  $\mathcal{S}$ ,  $i+1$ )
17:         add  $p$  to  $S_i$ 
18:       break
19:   if  $p$  does not belong to any  $S_i$  then
20:     append  $\{p\}$  to  $\mathcal{S}$ 
21: return  $\mathcal{S}$ 

```

The pseudo code of AdjustSkyOrd is given in Algorithm 2. It takes three inputs: the temporary list S_{tmp} containing points excluded from the current skyline subset in the scanning, the current Skyline order \mathcal{S} , and the next subset index in the scanning. It loops on each subset S_i in \mathcal{S} starting from S_{next} . Each point p in S_i is moved to a temporary list $temp$ if it is not

dominated by any point in S_{tmp} (lines 3–5). If S_i has not been changed, indicated by an empty temporary list $temp$, S_{tmp} is added to \mathcal{S} immediately before S_i , and the algorithm returns (lines 6–7). A non-empty $temp$ is merged into S_{tmp} (line 8). If S_i becomes empty after all its points have been checked, S_{tmp} replaces it in \mathcal{S} , and the algorithm returns (lines 9–10). Otherwise, S_i and S_{tmp} are swapped, and the loop continues to the next subset in \mathcal{S} (line 11). If S_{tmp} is not empty when the loop is over, it is appended to \mathcal{S} (lines 12–13).

Taking Figure 2 as an example, let the processing order of all hotels be I, E, F, A, D, H, G, C, B. Table 3 lists the resulting skyline order after each hotel is processed. The * symbol indicates that the current p triggers a call of the AdjustSkyOrd algorithm.

Algorithm 2 AdjustSkyOrd(Temporary list S_{tmp} , Current Skyline order \mathcal{S} , Next subset index $next$)

```

1: for each subset  $S_i$  from  $S_{next}$  to  $S_n$  in  $\mathcal{S}$  do
2:    $temp = \emptyset$ 
3:   for each point  $p$  in  $S_i$  do
4:     if  $\nexists q \in S_{tmp}$  s.t.  $q \prec p$  then
5:       move  $p$  from  $S_i$  to  $temp$ 
6:   if  $temp == \emptyset$  then
7:     add  $S_{tmp}$  to  $\mathcal{S}$  immediately before  $S_i$ ; return
8:   merge  $temp$  to  $S_{tmp}$ 
9:   if  $S_i == \emptyset$  then
10:    replace  $S_i$  in  $\mathcal{S}$  with  $S_{tmp}$ ; return
11:   swap  $S_i$  and  $S_{tmp}$ 
12: if  $S_{tmp} \neq \emptyset$  then
13:   append  $S_{tmp}$  to  $\mathcal{S}$ 

```

When E is processed, it is merged with I to form the only skyline order subset because they are incomparable to each other. As F dominates I, this causes $S_{tmp} = \{I\}$ that becomes the new tail in the skyline order after the AdjustSkyOrd algorithm is called. When G is processed, I is again moved out to the new tail.

Finally, when C is processed, the situation is more complex. The $S_{tmp} = \{E, F\}$ is the input to the AdjustSkyOrd algorithm, in which S_{tmp} is compared with the skyline order subsets $S_2 = \{G, H\}$ and $S_3 = \{I\}$ in turn. In the first iteration of the for-loop, G is merged with $\{E, F\}$, as G is dominated by neither E nor F. Then $S_{tmp} = \{E, F, G\}$ is swapped with the reduced $S_2 = \{H\}$. In the second iteration, $S_{tmp} = \{H\}$ is expanded to become $\{H, I\}$ and replaces $S_3 = \{I\}$ as the new S_3 .

4.2.2 Improvements

The basic algorithm calls the procedure AdjustSkyOrd when a new point p is found to dominate some old points that form the temporary list S_{tmp} passed to AdjustSkyOrd. We can eliminate the cost of this by pre-sorting the data set P . Assuming that smaller values are preferred in the skyline computation, we pre-sort P in the non-descending order of the sum of a point's dimension values. As a result, a point p is not able to dominate any point q that has been processed before it [10]. The relevant improved pseudo code is described

TABLE 3
Example Steps of Algorithm 1

Current p	Resulting Skyline Order
I	$\langle \{I\} \rangle$
E	$\langle \{E, I\} \rangle$
F*	$\langle \{E, F\}, \{I\} \rangle$
A	$\langle \{A, E, F\}, \{I\} \rangle$
D	$\langle \{A, D, E, F\}, \{I\} \rangle$
H	$\langle \{A, D, E, F\}, \{H, I\} \rangle$
G*	$\langle \{A, D, E, F\}, \{G, H\}, \{I\} \rangle$
C*	$\langle \{A, C, D\}, \{E, F, G\}, \{H, I\} \rangle$
B	$\langle \{A, B, C, D\}, \{E, F, G\}, \{H, I\} \rangle$

in Algorithm 3. Here, we have no list S_{tmp} , and neither do we call the procedure AdjustSkyOrd.

A further improvement is to carry out a binary search for the loop on all existing subsets in the skyline order, instead of the sequential scan in line 3 in Algorithm 3. Essentially, the loop on all skyline order subsets finds the subset S_x that satisfies: (a) all points in S_x are incomparable to p ; (b) x is the minimum index among all such subsets satisfying (a). A binary search starts with $low = 1$ and $high = n$, the skyline order length. At each loop step, we check the dominance relationship between the current point p and the subset S_{mid} where $mid = \lfloor low + high \rfloor / 2$.

Algorithm 3 SkylineOrderPreSort (sorted data set P)

```

1:  $S = \langle \rangle$ 
2: for each point  $p$  in  $P$  do
3:   for each  $S_i$  from  $S_1$  to  $S_n$  in  $S$  do
4:      $isSky = \text{TRUE}$ 
5:     for each point  $q$  in  $S_i$  do
6:       if  $q \prec p$  then
7:          $isSky = \text{FALSE}$ ; break
8:     if  $isSky$  then
9:       add  $p$  to  $S_i$ ; break
10:  if  $p$  does not belong to any  $S_i$  then
11:    append  $\{p\}$  to  $S$ 
12: return  $S$ 

```

Due to P being sorted, only two possibilities exist: p is dominated by some point(s) in S_{mid} or p is incomparable to all points in S_{mid} . For the former, we set $low = mid + 1$ and continue the binary search. For the latter, we look one subset backwards, checking the relationship between p and the subset S_{mid-1} . If p is dominated by some point(s) in S_{mid-1} , S_{mid} is exactly what we need and the binary search ends. Otherwise, i.e., points in S_{mid-1} are also incomparable to p , we set $high = mid - 1$ and continue the binary search. When the binary search terminates, either the expected subset S_x is found and p is inserted into S_x , or the tail of the skyline order is reached and p will constitute a new singleton tail. Note that p cannot be a new singleton head as it cannot dominate any point before it when P is sorted as described above.

We note that the backward check above is necessary to ensure the correctness of the binary search based algorithm. The binary search here is different from an

exact match binary search, which returns as soon as the expected value is found. The incomparable relationship here is not equivalent to an exact match. Rather, we need to find the minimum (instead of any) index x such that any point in the subset S_x is incomparable to p .

Referring to Figure 2, let the current skyline order be $\langle \{C\}, \{F\}, \{I\} \rangle$ and B be the p being processed. For simplicity, we ignore the other points. Note that B should be inserted into $\{C\}$ to maintain a correct skyline order, as $\{C\}$ is the first skyline order subset and C is comparable with B . However, if we do not employ a backward check, B will be inserted into $\{F\}$ instead, since we have $low = 1$ and $high = 3$, which yields $mid = 2$.

The binary search idea can also be used in Algorithm 1. However, it does not lower the possibility that a new point p dominates an old one as the input data set is not sorted. Hence, the frequent and time-consuming call of AdjustSkyOrd is not avoided at all.

4.2.3 Analyses

We briefly analyze the proposed skyline order computation algorithms, by regarding the dominance comparisons between two points as the characteristic operation. For the sake of presentation simplicity, we do not count the presorting time cost when analyzing the SkyOrdPreSort algorithm and its variant with binary search.

For the SkylineOrderScan algorithm, the best case is that each point p from P forms a new first skyline order subset in S , without invoking the procedure AdjustSkyOrd. In this case, only one dominance comparison is needed, involving p and the single point in the current S_1 in S . As a result, the total number of dominance comparisons, processing the whole data set P , is $N - 1$.

The worst case, however, involves cascading adjustments caused by a call of AdjustSkyOrd. Suppose we are processing the j 'th point p from P , and the cardinalities of S_{tmp} and S_{i+1} to S_n are c_0, c_{i+1}, \dots, c_n , respectively, before AdjustSkyOrd is called in line 16 of Algorithm 1. Note that $c_0 + \sum_{k=i+1}^n c_k \leq j - 2$ as we are processing the j 'th point p from P and S_i cannot be empty if AdjustSkyOrd is called (lines 12–16 in Algorithm 1).

A total cascading adjustment chain in AdjustSkyOrd (Algorithm 2) maintains all but one point in every current skyline subset S_i . This is because if $temp$ is empty, there will be no more cascading adjustment (lines 6–7 in Algorithm 2). Those points are swapped into S_{tmp} (line 11 in Algorithm 2) and then compared with all points in S_{i+1} . This happens until S_n is reached. Considering that all points in the input S_{tmp} are compared with the initial S_{next} , the total number of dominance comparisons is $c_0 \cdot c_{i+1} + (c_{i+1} - 1) \cdot c_{i+2} + \dots + (c_{n-1} - 1) \cdot c_n = c_0 \cdot c_{i+1} + \sum_{k=i+1}^{n-1} c_k \cdot c_{k+1} - \sum_{k=i+2}^n c_k \leq (c_0 + \sum_{k=i+1}^n c_k)^2 / 2 = (j - 2)^2 / 2$. In the worst case, processing each j 'th ($j \geq 2$) point p from P incurs this cost. As a result, the number of dominance comparisons contributed by AdjustSkyOrd is bounded by $\sum_{j=2}^N (j - 2)^2 / 2 = (\sum_{k=1}^{N-2} k^2) / 2 = (N - 2) \cdot (N - 1) \cdot (2N - 3) / 12$,

yielding a worst case complexity of $\mathcal{O}(N^3)$. This is also the upper bound of the worst case cost of the SkylineOrderScan algorithm, as it cannot incur more dominance comparisons without calling AdjustSkyOrd.

For the SkyOrdPreSort algorithm without binary search, the best case is the same as that of SkylineOrderScan, i.e., $N - 1$ dominance comparisons. Its worst case happens if each point p from P is compared with every point in every skyline subset from the first to the last in S . The worst case leads to a total of $N(N - 1)/2$ dominance comparisons.

For the SkyOrdPreSort variant with binary search, the best case is that every point from P forms a singleton skyline order subset, and all points already in S are compared with the newly encountered i 'th point p from P in a binary search fashion. This requires a total of $\sum_{i=1}^N \log_2 i = \log_2(N!) = \mathcal{O}(N \log N)$ dominance comparisons. The worst case is that all points from P form only one skyline order subset, which causes each point p from P to be compared with all points encountered before it. As a result, the total number of dominance comparisons in the worst case is $N(N - 1)/2$.

A summary of the cost analyses is given in Table 4.

TABLE 4
Summary on Algorithm Complexity

Algorithm	Best Case	Worst Case
SkylineOrderScan	$\mathcal{O}(N)$	$\mathcal{O}(N^3)$
AdjustSkyOrd	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$
SkylineOrderPreSort	$\mathcal{O}(N \log N)$	$\mathcal{O}(N^2)$

5 PROCESSING SIZE CONSTRAINED SKYLINE QUERIES WITH SKYLINE ORDER

We next show how skyline order can be used to process size constrained skyline queries. We first give a query definition, and then present algorithms for the query.

5.1 Definition

Based on skyline order, we are able to define a new size constrained skyline query that combines pointwise ranking and set-wide maximization to return interesting points according to both approaches.

Definition 5.1: (Skyline Order Based Size Constrained Skyline Queries) Let a set P of d -dimensional points with skyline order $S = \langle S_1, S_2, \dots, S_n \rangle$ be given. The skyline order based size constrained skyline query $Q_k^{sosc}(P)$ retrieves the set S_{sosc} defined as follows:

$$S_{sosc} = \left(\bigcup_{i=1}^l S_i \right) \cup S'_{l+1}$$

where l is defined such that $\sum_{i=1}^l |S_i| \leq k < \sum_{i=1}^{l+1} |S_i|$,

and $S'_{l+1} \subset S_{l+1}$ such that $|S_{sosc}| = k$.

Here, S'_{l+1} is selected from S_{l+1} using some set-wide maximization approach presented in Section 3.2. The query returns consecutive skyline order subsets from the original skyline until k points have been returned.

5.2 Algorithms

We present two algorithms that process a size constrained skyline query using the concept of skyline order. One takes advantage of a pre-computed skyline order, whereas the other does not.

5.2.1 Algorithm With Pre-Computed Skyline Order

If the skyline order S of the data set P has been computed before a size constrained skyline query $Q_k^{sosc}(P)$ is issued, S can be used to facilitate the query processing. Following Definition 5.1, our algorithm uses a simple loop on all pre-computed skyline order subsets while maintaining a count of the points seen. The relevant pseudo code is described in Algorithm 4. First, the result set S is initialized to be empty, and a count variable cnt is set to k (line 1). Then each subset S_i in S is checked sequentially. If the cardinality of S_i equals the current count in cnt , S_i is merged into S , and the loop stops (lines 3–4). If the cardinality of S_i is smaller than cnt , S_i is merged into S and the loop continues with an updated cnt (lines 5–6). Otherwise, $rSKY$ is called to select the last cnt points from S_i , and the loop stops (line 8).

Algorithm 4 SCSQuerySkyOrdPre (Skyline order S , number of points to retrieve k)

```

1:  $S = \emptyset$ ;  $cnt = k$ 
2: for each subset  $S_i$  from  $S_1$  to  $S_n$  in  $S$  do
3:   if  $|S_i| == cnt$  then
4:      $S = S \cup S_i$ ; break
5:   else if  $|S_i| < cnt$  then
6:      $S = S \cup S_i$ ;  $cnt = cnt - |S_i|$ 
7:   else
8:      $S = S \cup rSKY(S_i, cnt)$ ; break
9: return  $S$ 

```

The set-wide maximization approaches discussed in Section 3.2 can be used for implementing $rSKY$. Such approaches will be experimentally compared in Section 6.2.

As a remark, skyline orders are pre-computed (and maintained) with the same reasoning as are materialized views [20]. They can help efficiently solve arbitrary size constraints posed by future skyline queries.

5.2.2 Algorithm Without Pre-Computed Skyline Order

If no precomputed skyline order S of the data set P is available, query $Q_k^{sosc}(P)$ can be processed as described in Algorithm 5, which accepts a sorted data set P as described in Section 4.2.2 and accomplishes the query processing in two phases.

In the first phase, it follows the same logic as the improved skyline order computation in Algorithm 3. But only a partial skyline order S is now maintained. As soon as there are enough points in S , the extra subsets

Algorithm 5 SCSQuerySkyOrd (sorted data set P , number of points to retrieve k)

```

1:  $S = \langle \rangle$ 
2: for each point  $p$  in  $P$  do
3:   for each  $S_i$  from  $S_1$  to  $S_n$  in  $S$  do
4:      $isSky = \text{TRUE}$ 
5:     for each point  $q$  in  $S_i$  do
6:       if  $q \prec p$  then
7:          $isSky = \text{FALSE}$ ; break
8:     if  $isSky$  then
9:       add  $p$  to  $S_i$ ; break
10:    else
11:      if  $\sum_{1 \leq j \leq i} |S_j| \geq k$  then
12:        remove all subsets after  $S_i$  in  $S$ ; break
13:  if ( $isSky$  AND  $\sum_{S_i \in S} |S_i| < k$ ) then
14:    append  $\{p\}$  to  $S$ 
15: return SCSQuerySkyOrdPre( $S, k$ )

```

on the tail are removed from S , any point p belonging to those subsets are also ignored (lines 10–12). A new singleton tail is created only when S still has less than k points in total (lines 13–14). By keeping a partial skyline order only, considerable computation cost is saved. In the second phase, SCSQuerySkyOrdPre (Algorithm 4) is called to pick up k points from the partial skyline order S (line 15).

Following the same line of reasoning as covered in Section 4.2.2, the sequential scan on all skyline order subsets (line 3 in Algorithm 5) can also be replaced by a binary search. We will experimentally study this improvement in Section 6.

6 EXPERIMENTAL STUDIES

In this section, we evaluate skyline order algorithms and size constrained skyline query algorithms.

We generate both independent and anti-correlated synthetic data sets. Anti-correlated data sets are generated according to a method introduced in previous work [5]. The value domain for each dimension is $[0, 1]$. All parameters and their settings are listed in Table 5, with $1K = 1024$. All code is written in Java and run on a Windows XP PC with a 2.8GHz Intel Pentium D CPU and 1GB RAM.

TABLE 5
Parameters Used in Experiments

Parameter	Setting
Data set distribution	Indep. (IN), Anti-Correl. (AC)
Data set cardinality	100K, 200K, ..., 1000K
Data set dimensionality	2, 3, 4, 5, 10, 15, 20
k	50, 100, 150, ..., 500

6.1 Experiments on Skyline Order

A more comprehensive experimental study on skyline order can be found elsewhere [22].

6.1.1 Skyline Order Distribution

We first fix the data set dimensionality to 2 and 5, and vary the cardinality from 100K to 1,000K. The resulting skyline order lengths, i.e., the number of subsets in a skyline order, are listed in Table 6. The sizes of the first two skyline order subsets are given in parentheses. We can see that for both data distributions, larger cardinality leads to longer skyline order. However, an anti-correlated data set has shorter skyline order than its independent counterpart. This is because an anti-correlated data set has more skyline points than an independent one, when they are of the same cardinality.

TABLE 6
Skyline Order Length vs. Cardinality

Card.	Anti-Corre.	Indep.
100K, 2D	210 (52, 82)	617 (8, 18)
5D	18 (15493, 29209)	23 (1003, 2684)
200K, 2D	297 (46, 80)	886 (9, 17)
5D	19 (21143, 45826)	27 (1108, 3308)
300K, 2D	363 (48, 107)	1096 (15, 19)
5D	21 (26037, 59842)	28 (1413, 4347)
400K, 2D	418 (47, 88)	1257 (14, 16)
5D	23 (29200, 70356)	30 (1561, 4691)
500K, 2D	467 (53, 81)	1412 (10, 14)
5D	23 (31589, 79498)	32 (1512, 4649)
600K, 2D	518 (51, 73)	1546 (9, 15)
5D	24 (34680, 89413)	32 (1646, 5164)
700K, 2D	556 (45, 79)	1668 (14, 22)
5D	26 (36377, 94982)	34 (1886, 5853)
800K, 2D	593 (64, 110)	1778 (15, 22)
5D	26 (38394, 102850)	35 (1876, 6300)
900K, 2D	633 (61, 88)	1909 (9, 15)
5D	27 (39767, 108147)	37 (1727, 5888)
1000K, 2D	671 (67, 98)	2008 (13, 24)
5D	27 (41991, 114960)	38 (1956, 6010)

Next we fix the data set cardinality to 100K and vary the dimensionality from 3 to 5, 10, 15, and 20. The resulting skyline order length are listed in Table 7. For both data distributions, the dimensionality has an apparent impact on the skyline order length: as the dimensionality increases, the skyline order get significantly lower. This is attributed to the “curse of dimensionality” that also affects skyline queries: the more dimensions a data set has, the higher the percentage of its points enter its skyline. Further, the difference between anti-correlated data sets and independent data sets is also apparent here.

TABLE 7
Skyline Order Length vs. Dimensionality

Dim. \ Distrib.	Anti-Corre.	Indep.
2D	210 (52, 82)	617 (8, 18)
3D	66 (653, 1420)	105 (85, 155)
4D	29 (4432, 10414)	41 (338, 868)
5D	18 (15493, 29209)	23 (1003, 2684)
10D	5 (84393, 14531)	6 (26207, 46478)
15D	3 (96972, 5417)	3 (77920, 24095)
20D	2 (101438, 962)	2 (98826, 3574)

6.1.2 Skyline Order Computation Cost

We compare three skyline order computation approaches: the basic scan approach (Algorithm 1), the improved approach with pre-sorting (Algorithm 3), and the one with binary search in addition to pre-sorting. We measure the total computation time cost. As we are focused on the efficiency of these approaches themselves, the time for pre-sorting is excluded from the results.

The effects of data set cardinality are reported in Figure 3. For both distributions, we fix the dimensionality at 2. It is seen that as cardinality increases, the basic scan approach deteriorates dramatically, the binary search approach only slowly incurs higher cost, and the pre-sorting approach remains in-between. The pre-sorting approach performs better than the basic scan approach because the pre-sorting avoids the expensive cascading adjustment (the calling of `AdjustSkyOrd` in Algorithm 2), which happens when a current point p dominates some points scanned before it.

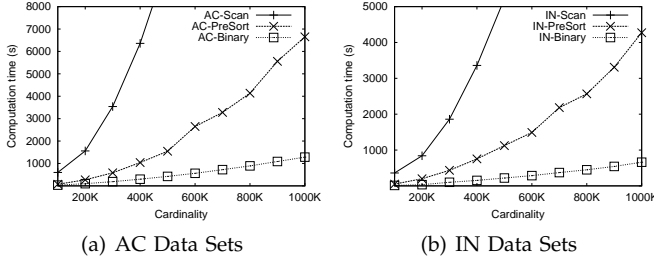


Fig. 3. Computation Cost vs. Cardinality

The binary search approach improves the performance further, as it does not sequentially scan all skyline order subsets. In addition, the performance improvement between the basic scan approach and the pre-sorting approach is much more significant than the one between the latter and the binary search approach. This difference shows that `AdjustSkyOrd` is the most time-consuming part of the basic scan approach. This also indicates that applying the binary search idea directly to the basic scan approach is not attractive, as it has no chance to outperform the pre-sorting approach.

The effects of data dimensionality are reported in Figure 4. Here the cardinality is fixed at 100K. We have two important observations. First, the binary search approach is no longer the best when the dimensionality exceeds 2; rather, it becomes the worst when the dimensionality is higher than 10. This is because the skyline order length of a given data set shrinks as the dimensionality increases (refer to Table 7). Shorter skyline orders disfavor binary search, which consequently performs poorly with considerable extra costs for worst-case inputs.

Second, as the dimensionality increases (higher than 10 for anti-correlated data sets and 15 for independent data sets), the basic scan approach becomes comparable to the pre-sort approach. For dimensionality 10 to 20, the very short skyline order lengths (seen from Table 7)

provide very limited room for optimizations that would otherwise benefit the pre-sort approach.

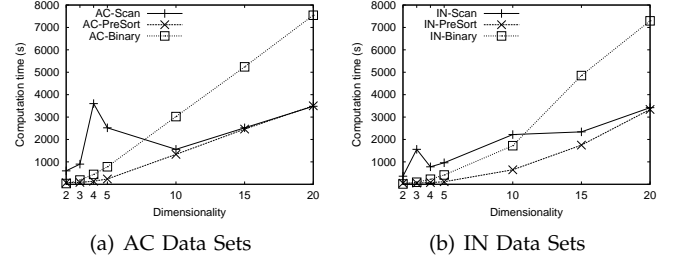


Fig. 4. Computation Cost vs. Dimensionality

6.1.3 Adapting to Disk-Resident Data

So far we have assumed that both input data and skyline order subsets are kept in the main memory. If the data is too large to fit in memory, skyline order computation algorithms require appropriate adaptations to work properly. We adapt the binary search approach since it is the most efficient in most cases.

In particular, all data points are loaded into memory in different batches. For each batch, a binary search is executed. A skyline order subset is initially created in memory, and it may be output to the disk later to get free memory blocks. When a subset needed for comparison is not in memory, it is loaded from disk. Small skyline order subsets are allowed to share a memory block, whereas a large one may use multiple blocks with the last one accepting insertions of points. A very large subset that requires more memory than available is loaded in batches for each point batch. When a dirty memory block is to be eliminated, modified data will be output to disk to the corresponding skyline order subset(s). When all points are processed, all dirty memory blocks are output.

We also adapt the IO-optimal BBS algorithm [23] to compute skyline orders as follows. In addition to the main min-heap for all R-tree entries containing skyline points, we use an extra min-heap to contain all the entries that are pruned. When the skyline (also the first subset of the skyline order) is computed for the input data set, the extra heap is upgraded to be the main heap, and the BBS algorithm is executed to obtain the second skyline order subset, with the old main heap being the extra heap for pruned entries. This upgrading and processing is repeated until all data points are processed, indicated by an empty extra min-heap.

We use an LRU buffer whose size is 1% of that of the input data set. For the adapted binary search approach, one buffer block is used for loading the input data, and the others are used for skyline order subsets. For the adapted BBS algorithm, half of the buffer blocks are used for the R-tree and half are used for the skyline order subsets. The R-tree page size is set to 4K. The IO costs are reported in Figure 5. The inferiority of the adapted

BBS approach, especially when the data set is large, is attributed to the fact that all pruned R-tree entries in the extra heap are accessed again subsequently, which results in ineffective pruning.

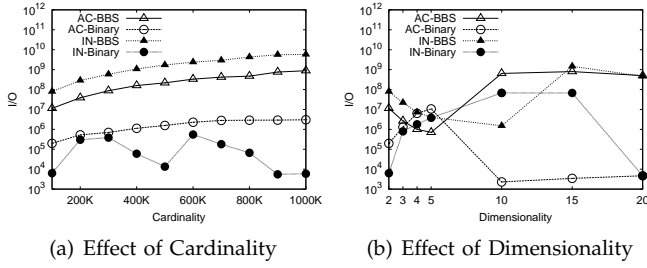


Fig. 5. IO Costs of Skyline Order Computation

6.2 Experiments on Queries

We proceed to compare different approaches that can be used for answering size constrained skyline queries. From the ranking based approaches, we choose the subspace skyline frequency approach [7] (SSF) as a representative. The algorithm parameters in SSF are set as: $\epsilon = 0.2$ and $\delta = 0.05$. Readers are referred to [7] for detailed explanations on these parameters.

We also compare different skyline orders based query processing approaches: the algorithm with pre-computed skyline order (Algorithm 4 in Section 5.2.1), the algorithm without pre-computed skyline order (Algorithm 5 in Section 5.2.2), and the binary search algorithm (an improvement of Algorithm 5 in Section 5.2.2).

We combine each of the algorithms with different skyline reduction algorithms to return exact number of result points. We compare our volume of dominating region (VDR) estimation based approach (Section 3.2.2) with the top- k representative skyline points approach [21]. For the former, we use the heuristic that maximizes the sum of VDR values of candidate points. For the latter, we implement a bitmap based greedy algorithm, where each candidate point has a long bitmap indicating all points dominated by it. Note that the greedy algorithm does not return the optimal result for the top- k representative problem, which actually is NP-hard for data sets with dimensionality 3 or higher [21].

All approaches, apart from SSF, and their abbreviations to be used in the performance result graphs are listed in Table 8. We proceed to compare the response times of these approaches.

6.2.1 Effect of Cardinality

In this batch of experiments, we fix the data set dimensionality at 2 and k at 100, and we study the effects of the cardinality on the different approaches. We vary the cardinality from 100K to 1,000K; the resulting query computation times are reported in Figure 6. For clarity, we only plot a portion of the results here. The omitted results exhibit similar trends.

TABLE 8
Description of Approaches

SCS query algorithm	Set-Wide Maximization	
	Count (C [21])	Estimation (E)
Pre-computation based (P)	PCM	PEM
No pre-computation (O)	OCM	OEM
Binary search (B)	BCM	BEM

E: Estimation of VDR (Sec. 3.2.2)

P: Algorithm SCSQuerySkyOrdPre (Sec. 5.2.1)

O: Algorithm SCSQuerySkyOrd (Sec. 5.2.2)

B: Improvement on O (Sec. 5.2.2)

Several observations are noteworthy for both data distributions. First, pre-computation based approaches usually outperform their corresponding counterparts without pre-computation: PCM is better than OCM/BCM; PEM is better than OEM/BEM. This shows that pre-computed skyline orders save query processing time.

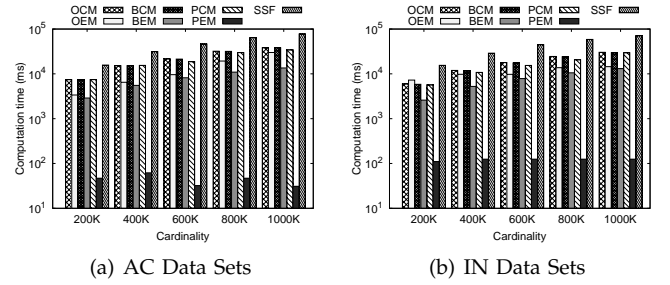


Fig. 6. Query Cost vs. Cardinality

Second, when reducing skyline size, the VDR estimation based approach is better than the count of dominated points based approach: OEM is better than OCM, BEM is better than BCM, and PEM is better than PCM. The estimation based approach is much faster because it does not explore the global dominance relationship to count, but makes estimates based on the given candidate point only. Because of the dual merits of pre-computation and estimation, the PEM approach has a significant advantage over its competitors.

Third, the total ranking approach SSF is the worst among all approaches, and it degrades noticeably as the cardinality increases. The total pointwise ranking of SSF is very time-consuming, and larger cardinalities inevitably worsen the situation.

Fourth, PEM performs better on anti-correlated data sets than on independent ones. For an anti-correlated data set, its starting skyline order subsets contain more points than those of an independent data set. This difference makes PEM explore fewer skyline order subsets than the former. However, PCM does not have this advantage. The benefit of pre-computation of PCM is counteracted by its time-consuming count based skyline reduction approach.

6.2.2 Effect of Dimensionality

Next, we fix the data cardinality at 100K and k at 100, and then vary the dimensionality from 2 to 5. The results are reported in Figure 7.

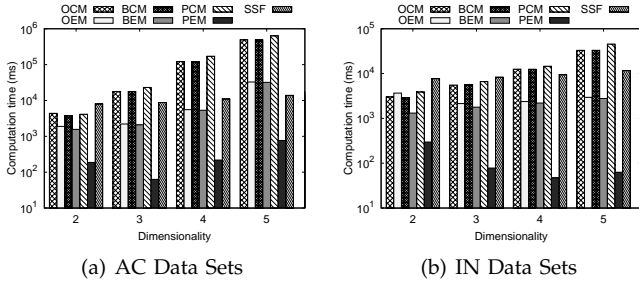


Fig. 7. Query Cost vs. Dimensionality

As the dimensionality grows, most approaches deteriorate markedly, especially OCM, BCM, and PCM. The count based reduction incorporated by these approaches requires more time to check for dominance when dimensionality increases. In contrast, the estimation based approaches OEM and BEM are more steady. The cost of PEM even decreases in some cases: from 2D to 3D on anti-correlated data sets; from 2D to 4D on independent data sets. These surprising improvements are attributed to skyline order subset size variations. Referring to Table 7, PEM needs to explore 2 subsets for 2D data set, but only one for other data sets. This helps reduce query processing time on one hand. On the other hand, 4D and 5D data sets contain much more points than k in their first skyline order subsets, meaning that considerable time is needed to reduce points. As a result, the overall costs on 4D and 5D data sets increase. Similar reasoning applies to independent data sets.

6.2.3 Effect of k

In the next experiment, we fix the data cardinality at 1,000K and the dimensionality at 2, and we then vary k from 50 to 500. Figure 8 reports the relevant results.

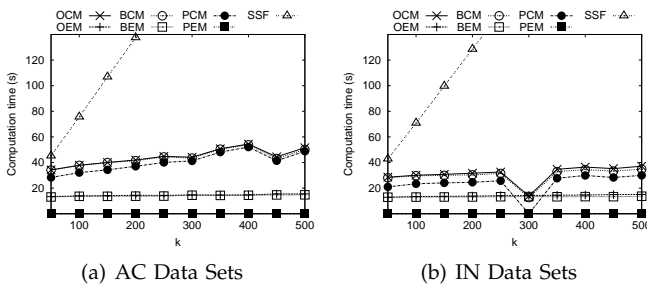


Fig. 8. Query Cost vs. k

It is again seen that pre-computation is beneficial and that VDR estimation is better than the count based approach. For the approaches incorporating count based reduction (the *CM approaches), the costs usually increase for larger k . As k increases, more skyline order

subsets are explored. For those subsets, the numbers of candidate points increase as the subset indexes increase. Hence, the *CM approaches slow down because their greedy heuristic reduction gets larger inputs. In contrast, the approaches with estimation based reduction (the *EM approaches) are considerably more steady because VDR estimation is computationally quite fast even for larger numbers of candidate points.

It is seen that the *CM approaches exhibit “drops”: at $k = 450$ for the anti-correlated data set, and at $k = 300$ for the independent data set. The independent data set of 1,000K points has exactly 300 points in its first eight skyline order subsets. As a result, for the case $k = 300$ no skyline reduction occurs for any of the *CM and the *EM approaches. Therefore, the cost of any *CM approach is exactly the same as that of its *EM counterpart. The anti-correlated data set of 1,000K points has a total of 448 points in its first four skyline order subsets. As a result, for $k = 450$ any of the *CM approaches need to choose only 2 additional points from the fifth skyline order subset with 169 points, which is quite easy for the greedy heuristic employed by the *CM approaches.

6.2.4 Performance on Disk-Resident Data

To investigate the query performance on disk-resident data, we adapt the algorithms in a similar way as in Section 6.1.3.

We also adapt the IO-optimal BBS algorithm [23] to process queries. In particular, skyline order subsets are computed on-the-fly one by one using the adapted BBS version with two min-heaps, as detailed in Section 6.1.3. After a skyline order subset is computed, the BBS processing stops if all current subsets contain k or more points in total. For the latter, the VDR estimation covered in 3.2.2 is used to reduce the result to exactly k points. The R-tree page size is set to 4K for the BBS algorithm.

We use an LRU buffer with a size of 1% of the data size. For all algorithms, one buffer block is used to contain the query result; others are used for holding skyline order subsets or R-tree nodes. The IO costs are reported in Figures 9 to 11. The settings are given in the captions. Our PEM approach always incurs the lowest IO cost, followed by the PCM approach in most cases.

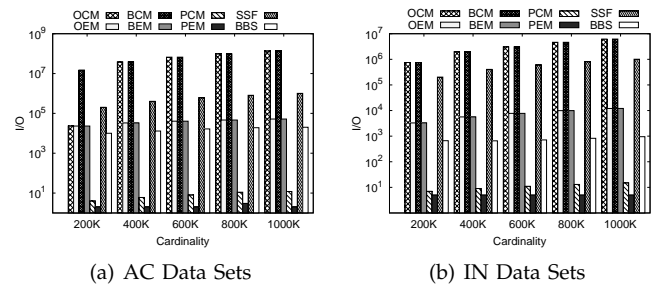


Fig. 9. IO Cost vs. Cardinality ($d = 5, k = 100$)

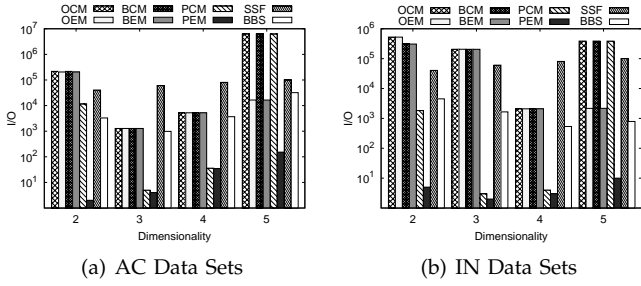


Fig. 10. IO Cost vs. Dimensionality ($|P| = 100K$, $k = 100$)

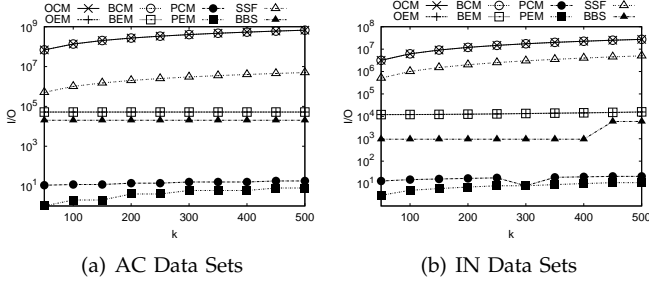


Fig. 11. IO Cost vs. k ($|P| = 1000K$, $d = 5$)

6.2.5 Interpretation of Query Results

Note that all the *EM approaches produce the same set of points given the same input; so do all the *CM approaches. Therefore, we compare the results of PEM and PCM with those alternatives.

As mentioned in Section 3.3, pointwise ranking is prone to ties and thus to selecting points with similar features, while results with diverse points are usually more desirable to users issuing skyline queries. Therefore, we evaluate the diversity of the query results computed by the different approaches based on the variance among the points returned from synthetic data sets. Let R be the set of points returned as the query result. Its variance $V(R)$ is defined as the average Euclidean distance between points in R and the mean of R , i.e., $V(R) = \frac{1}{k} \sum_{p \in R} \|p - \bar{R}\|^2$, where k is $|R|$.

We fix the synthetic data cardinality at 1,000K and the dimensionality at 5. The results obtained are shown in Figure 12. For the anti-correlated data set, all approaches are quite close to each other in terms of result variance. However, for the independent data set, our skyline order concept achieves better result variance due to its combined advantages; SSF fails to do so due to its pointwise ranking nature.

We proceed to compare the query results obtained from the NBA data set² that has 19,112 17-attribute records and generally follows a correlated data distribution. Following Yiu and Mamoulis [29], we query on the four most important attributes: games played, points, rebounds, and assists. All records are normalized to the unit space $[0, 1]^4$. The query results for $k=5$ are

2. NBA Statistics v2.0: <http://databasebasketball.com>.

listed in Table 9. PEM returns impressive result by identifying the peak seasons of the legend player Wilt Chamberlain; PCM returns players with larger variance. SSF fails to identify star players due to too many ties in its ranking. The pointwise ranking based top- k may prefer non-skyline points over skyline points. E.g., Kevin Garnett (2002) in its result is actually dominated by Wilt Chamberlain (1967) [29]. In contrast, PEM and PCM do not exhibit such anomalies.

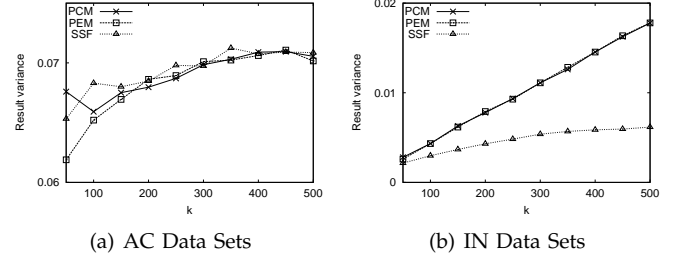


Fig. 12. Result Variance vs. k

7 CONCLUSION

Skyline queries are able to find interesting data points in multi-dimensional data sets. However, most previous work has been focused on how to reduce skyline result cardinalities. In this paper, we aim to resolve arbitrary size constraints on skyline queries, without any a priori assumptions about the relationship between the expected number of points and the skyline cardinality.

The paper describes how typical existing techniques are not well equipped to contend with arbitrary size constraints. Then the concept of skyline order is proposed where a data set is partitioned using the skyline dominance relationship. The skyline ordering provides an order among different partitions, but still reserves room for different local optimizations within each partition. Efficient algorithms are developed for computing skyline orders and resolving arbitrary size constraints using skyline order. Extensive empirical studies show that skyline order presents a flexible framework for efficient and scalable resolution of arbitrary size constraints on skyline queries.

ACKNOWLEDGMENTS

Christian S. Jensen is an Adjunct Professor at University of Agder, Norway.

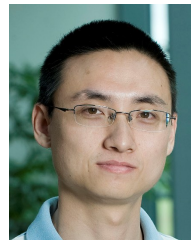
REFERENCES

- [1] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In *Proc. SODA*, pp. 179–187, 1990.
- [2] I. Bartolini, P. Ciaccia, and M. Patella. Efficient sort-based skyline evaluation. *TODS*, 33(4):1–49, 2008.
- [3] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *JACM*, 25(4):536–543, 1978.

TABLE 9
Comparison of Query Results

k	PCM	PEM	Top- k	SSF
1	Wilt Chamberlain (1967)	Wilt Chamberlain (1965)	Wilt Chamberlain (1967)	John Abramovic (1946)
2	Julius Erving (1974)	Wilt Chamberlain (1963)	Billy Cunningham (1972)	Chet Aubuchon (1946)
3	Julius Erving (1971)	Oscar Robertson (1961)	Kevin Garnett (2002)	Norm Baker (1946)
4	Oscar Robertson (1961)	Wilt Chamberlain (1966)	Julius Erving (1974)	Herschel Baltimore (1946)
5	Isiah Thomas (1983)	Wilt Chamberlain (1967)	Kareem Abdul-Jabbar (1975)	John Barr (1946)

- [4] H. Blunck and J. Vahrenhold. In-place algorithms for computing (layers of) maxima. In *Proc. SWAT*, pp. 363–374, 2006.
- [5] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. ICDE*, pp. 421–430, 2001.
- [6] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. Finding k -dominant skylines in high dimensional space. In *Proc. SIGMOD*, pp. 503–514, 2006.
- [7] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. On high dimensional skylines. In *Proc. EDBT*, pp. 478–495, 2006.
- [8] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The onion technique: Indexing for linear optimization queries. In *Proc. SIGMOD*, pp. 391–402, 2000.
- [9] J. Chomicki. Preference formulas in relational queries. *TODS*, 28(4):427–466, 2003.
- [10] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proc. ICDE*, pp. 717–719, 2003.
- [11] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou. Parallel Distributed Processing of Constrained Skyline Queries by Filtering. In *Proc. ICDE*, pp. 546–555, 2008.
- [12] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *Proc. VLDB*, pp. 229–240, 2005.
- [13] G. Hjaltason and H. Samet. Distance browsing in spatial database. *ACM TODS*, 24(2):265–318, 1999.
- [14] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline queries against mobile lightweight devices in MANETs. In *Proc. ICDE*, page 66, 2006.
- [15] W. Jin, M. Ester, and J. Han. Efficient processing of ranked queries with sweeping selection. In *Proc. PKDD*, pp. 527–535, 2005.
- [16] W. Jin, J. Han, and M. Ester. Mining thick skylines over large databases. In *Proc. PKDD*, pp. 255–266, 2004.
- [17] V. Koltun and C. H. Papadimitriou. Approximately dominating representatives. In *Proc. ICDT*, pp. 204–214, 2005.
- [18] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proc. VLDB*, pp. 275–286, 2002.
- [19] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *JACM*, 22(4):469–476, 1975.
- [20] P. Larson, H. Z. Yang. Computing Queries from Derived Relations. In *Proc. VLDB*, pp. 259–269, 1985.
- [21] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *Proc. ICDE*, pp. 86–95, 2007.
- [22] H. Lu, C. S. Jensen, and Z. Zhang. Skyline ordering: A flexible framework for efficient resolution of size constraints on skyline queries. *DB Technical Report TR-27*, Department of Computer Science, Aalborg University, 28 pages, 2010.
- [23] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. SIGMOD*, pp. 467–478, 2003.
- [24] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM TODS*, 30(1):41–82, 2005.
- [25] Y. Tao, L. Ding, X. Lin and J. Pei. Distance-based representative skyline. In *Proc. ICDE*, pp. 892–903, 2009.
- [26] K. L. Tan, P. K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *Proc. VLDB*, pp. 301–310, 2001.
- [27] A. Vlachou, C. Doukeridis, K. Nøravåg, and M. Vazirgiannis. Skyline-based Peer-to-Peer Top- k Query Processing. In *ICDE*, pp. 1421–1423, 2008.
- [28] T. Xia, D. Zhang, and Y. Tao. On skylining with flexible dominance relation. In *Proc. ICDE*, pp. 1397–1399, 2008.
- [29] M. L. Yiu and N. Mamoulis. Efficient processing of top- k dominating queries on multi-dimensional data. In *Proc. VLDB*, pp. 483–494, 2007.
- [30] Z. Zhang, X. Guo, H. Lu, A. K. Tung, and N. Wang. Discovering strong skyline points in high dimensional spaces. In *Proc. CIKM*, pp. 247–248, 2005.
- [31] Shiming Zhang, N. Mamoulis, and D. W. Cheung. Scalable skyline computation using object-based space partitioning. In *Proc. SIGMOD*, pp. 483–494, 2009.



Hua Lu received the BSc and MSc degrees from Peking University, China, in 1998 and 2001, respectively and the PhD degree in computer science from National University of Singapore in 2007. He is currently an assistant professor in the Department of Computer Science, Aalborg University, Denmark. His research interests include skyline queries, spatio-temporal databases, geographic information systems, and mobile computing. He is a member of the IEEE.



Christian S. Jensen, Ph.D., Dr.Techn., is a Professor of Computer Science at Aalborg University, Denmark, where he directs Center for Data-Intensive Systems. From September 2008 to August 2009, he was on sabbatical at Google Inc., Mountain View. He has previously been on several sabbaticals at University of Arizona. His research concerns data management and spans semantics, modeling, indexing, and query and update processing. During the past decade, his focus has been on spatio-temporal data management. He is an IEEE Fellow, a member of the Danish Academy of Technical Sciences and the EDBT Endowment, and a trustee emeritus of the VLDB Endowment. In 2001 and 2002, he received Ib Henriksen's Research Award for his research in mainly temporal data management and Telenor's Nordic Research Award for his research in mobile services and data management. He is vice president of ACM SIGMOD. He is an editor-in-chief of The VLDB Journal and has served on the editorial boards of ACM TODS, IEEE TKDE, and the IEEE Data Engineering Bulletin. He was PC chair or co-chair for STDM 1999, SSTD 2001, EDBT 2002, VLDB 2005, MobiDE 2006, MDM 2007, TIME 2008, and DMSN 2008. He will be PC chair of ACM SIGSPATIAL GIS 2011.



Zhenjie Zhang received his B.S. from Department of Computer Science and Engineering, Fudan University in 2004. He is currently a Ph.D. student and Research Assistant in the School of Computing, National University of Singapore. He was visiting student of Hong Kong University of Science and Technology in 2008 and visiting scholar of AT&T Shannon Lab in 2009. His research interests cover a variety of different topics including clustering analysis, skyline query, non-metric indexing and game theory. He has served as Program Committee members in VLDB 2010 and KDD 2010.