Int J Software Informatics, Volume 6, Issue 4 (2012), pp. 475–493
International Journal of Software and Informatics, ISSN 1673-7288

E-mail: ijsi@iscas.ac.cn
http://www.ijsi.org
Tel: +86-10-62661040

Skyline: Stacking Optimal Solutions in Exact and Uncertain Worlds

Wenjie Zhang[1], Muhammad Aamir Cheema[1], Ying Zhang[1], and Xuemin Lin[1,2]

[1](School of Computer Science and Engineering, University of New South Wales, Australia)

[2](School of Software Engineering, East China Normal University, Shanghai 200241, China)

Email: {zhangw,macheema,yingz,lxue}@cse.usw.edu.au

Abstract In many applications involving multiple criteria optimal decision making, users may often want to make a personal trade-off among all optimal solutions for selecting one object that best fits their personal needs. As a key feature, skyline in a multi-dimensional space provides a minimal set of candidates for such purposes by removing every object that is not preferred by *any* (monotonic) utility/scoring function; that is, the skyline removes all objects not preferred by any user no matter how their preferences vary. Due to its importance, the problem of skyline computation and its variants have been extensively studied in the database literature. In this paper, we provide a comprehensive survey of skyline computation techniques. Specifically, we first introduce the skyline computation algorithms on traditional (exact) data where each object corresponds to a point in a multi-dimensional space. Then, we discuss the skyline models and efficient algorithms to handle uncertain data which is inherent in many important applications. Finally, we briefly describe a few variants of the skyline (e.g., *skycube*, *k-skyband* and *reverse skyline*) in this paper.
Key words: query processing; skyline; uncertain data

Zhang WJ, Aamir Cheema M, Zhang Y, Lin XM. Skyline: stacking optimal solutions in exact and uncertain worlds. *Int J Software Informatics*, **Vol.6, No.4 (2012): 475–493. http://www.ijsi.org/1673-7288/6/i141.htm**

1 Introduction

In a d-dimensional space R^d, the *skyline* is defined over given preferences of coordinate values on each dimension (i.e., either smaller or larger coordinate values are preferred). Without loss of generality, *in the rest of the paper, we assume that smaller coordinate values are preferred on each dimension and all points are in* R^d_+ *(i.e., co-ordinate values are non-negative).* Given two points x and y in R^d_+, x *dominates* y if x is not greater than y on any dimension and is smaller than y on at least one dimension. More precisely, assuming $x[i]$ denotes the i-th co-ordinate value of a point x, we say x *dominates* y if $x[i] \leqslant y[i]$ for *every* dimension $i \in [1, d]$ and there exists at least one dimension $j \in [1, d]$ such that $x[j] < y[j]$.

Assume that the users prefer smaller values on each dimension and want to select a top object (point) from a set D of objects (points). Clearly, a point in D

This work is sponsored by ARC DP120104168 and ARC DE120102144 (Wenjie Zhang); ARC DP110104880 and UNSW ECR grant PS27476 (Ying Zhang); ARC DP0987557, ARC DP110102937, ARC DP120104168, and NSFC61021004 (Xuemin Lin).
Corresponding author: Xuemin Lin, Email: lxue@cse.unsw.edu.au
Received 2012-02-01; Accepted 2012-08-05.

that dominates every other point in D would be the top object. However, D may not contain a point that dominates every other point. Therefore, scoring functions are often used to rank the points in D. A *decreasing* scoring function f has the property that it returns higher scores for the points that have smaller co-ordinate values, i.e., $f(x) \geqslant f(y)$ if x dominates y. Hence, a decreasing scoring function may be used to capture the user's preference of smaller values on each dimension and the object with the highest score may be returned as the top object.

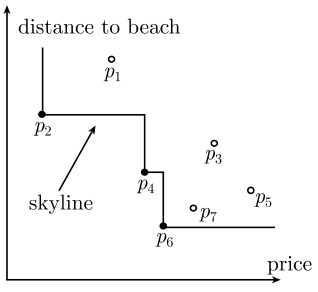
Clearly, the score of a point computed by a scoring function may be easily affected by outlier coordinate values of the point on some dimensions. Consequently, the users may not be content with the optimal solution based on a single scoring function. Moreover, due to the lack of domain knowledge or incomparable units on different dimensions (e.g., dollars vs meters), the users may not be able/willing to define a single scoring function that best meets their personal needs. A feasible solution to this problem is to return the users a set of points S where $S \subseteq D$ and has the following property: for each point $y \in D - S$, there is a point $x \in S$ such that $f(x) \geqslant f(y)$ for *every* decreasing scoring function f. In other words, the top object is always a point in S no matter which decreasing scoring function is used to rank the objects. The users may make personal trade-off to select one object from S which best meets their personal needs. We call such an S a *superior set* of D.

The *skyline* of D consists of the points in D which are not dominated by any other points in D. The skyline S of D provides the superior set of D with the minimum size for users to make their personal trade-offs. Such a personal trade-off is very effective especially when the number of skyline points is small.

Figure 1 demonstrates a classical motivating example for the skyline computation. Assume that a tourist is looking for a hotel that is cheap and is close to a beach. Figure 1 shows a list of hotels, each of which has two numerical attributes, *price* and *dist* (distance to the beach). Unless a single scoring function is used, a hotel that best meets her needs cannot be determined. Nevertheless, we can safely exclude p_1 because the hotel p_2 dominates the hotel p_1 and hence $f(p_2) \geqslant f(p_1)$ for *every* decreasing function f. In other words, the hotel p_2 is always preferable to p_1 regardless of the scoring function used. With similar rationale, we can also exclude p_3, p_5 and p_7 because they are all dominated by p_6. Hence, the skyline in Fig. 1 is $\{p_2, p_4, p_6\}$. These skyline hotels are returned to the user and she may choose a hotel that best meets her needs by making a personal trade-off between the price and the closeness

id	*dist*(km)	*price*($)
p_1	4.0	150
p_2	3.0	110
p_3	2.5	240
p_4	2.0	180
p_5	1.7	270
p_6	1.0	195
p_7	1.2	210

(a) Hotels



(b) Skyline

Figure 1. Skyline of hotels

to the beach. It can be guaranteed that her most preferable hotel is not missed if the skyline is returned.

A straightforward approach for the skyline computation is to check the dominance relationship between each pair of objects in the dataset and to remove the objects which are dominated by any other object. Clearly, this is computationally expensive especially when the number of objects is large. Due to the importance of the skyline, in the past few years, many efficient skyline computation algorithms are proposed and we briefly describe these techniques in this paper.

In many applications, uncertainty is inherent due to various factors such as data randomness and incompleteness, limitation of equipment, and delay or loss in data transfer. In such cases, the exact values of an object may be unknown and such an object is called an uncertain object. An uncertain object may be described either continuously or discretely. In *continuous* cases, an uncertain object U may be described by a probability density function (PDF) f_U such that $\int_{u \in U} f_U(u)du = 1$; Nevertheless, in many applications PDFs are not always available. Hence, a *discrete* case representation is used where an uncertain object U is represented by a set of *instances* such that each instance $u \in U$ has a probability $P(u)$ to appear. The *discrete* case representation has the property that $0 < P(u) \leq 1$ and $\sum_{u \in U} P(u) = 1$. Unlike the traditional skyline computation in which the dominance relationship between two objects are clearly defined, it is a challenge to capture the dominance relationship between two uncertain objects since there are multiple instances in each uncertain object. We provide an overview of various approaches that compute skyline on uncertain data.

Some important variations of skyline operators are also introduced in this paper. Figure 2 summaries the breakthroughs in skyline computation over both certain and uncertain data, as well as skylines variations.

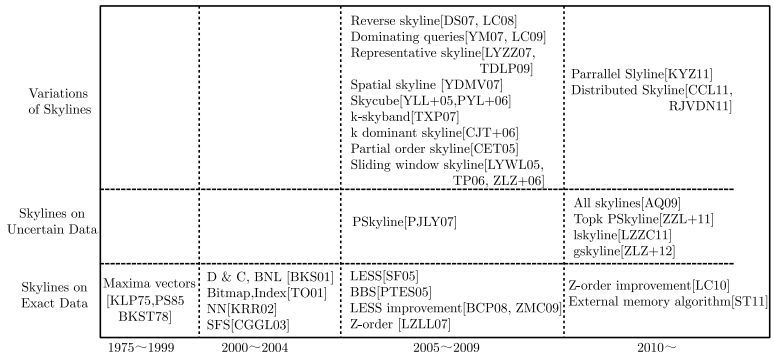


Figure 2. Research breakthroughs in skyline processing

This survey is organized as follows. In Section 2 and Section 3, we introduce some of the most notable skyline computation algorithms for exact data. Specifically, the algorithms that compute skyline over non-indexed data are presented in Section 2 followed by the skyline computation algorithms for indexed data in Section 3. In Section 4, we present an overview of different skyline models for uncertain data. We also describe the computation algorithm for each model. In Section 5, we introduce a few variants of the skyline such as *skycube*, *k-skyband* and *reverse skyline* etc.

2 Skyline Computation Over Non-Indexed Exact Data

The problem of skyline computation has been extensively studied ever since it was introduced in 1975 by Kung *et al.*[11]. They show that the time complexity of the skyline computation is $O(N\ log\ N)$ for $d = 2$ and $d = 3$ where d is the total number of dimensions and N is the total number of points. For $d \geqslant 4$, the time complexity is bounded by $O(N\ log^{d-2}\ N)$. Bentley *et al.* showed[5] that the average size of skyline is $O(log^{d-1}\ N)$ assuming that the values on each dimension are independent to the values on the other dimensions.

Realizing the need of efficient skyline computation for massive datasets, the database research community has also given significant attention to skyline computation since 2001[4]. In this section, we introduce a set of efficient skyline computation algorithms (for exact data) that assume that the data is not indexed by any data structure. Specifically, we survey *Divide and Conquer*[4], *Block Nested Loops*[4], *Sort First Skyline*[8] and *Linear Elimination Sort for Skyline*[29] algorithms.

2.1 Divide and conquer algorithm (D&C)

The D&C approach[4] recursively partitions the whole dataset until each partition fits in the main memory. For each partition, a skyline (called local skyline) is computed using a main-memory algorithm, e.g., Refs. [11, 25]. The final skyline is obtained by merging the local skylines.

Assume that we want to compute the skyline on the dataset shown in Fig. 3. First, D&C calculates the median of the data points on dimension x and divides the dataset into two parts, S_1 and S_2 (see Fig. 3(a)). Assuming that both S_1 and S_2 fit in main-memory, the skylines of S_1 and S_2 are computed. This step of D&C is called *divide step*. The local skylines of S_1 and S_2 are shown in Fig. 3(a).

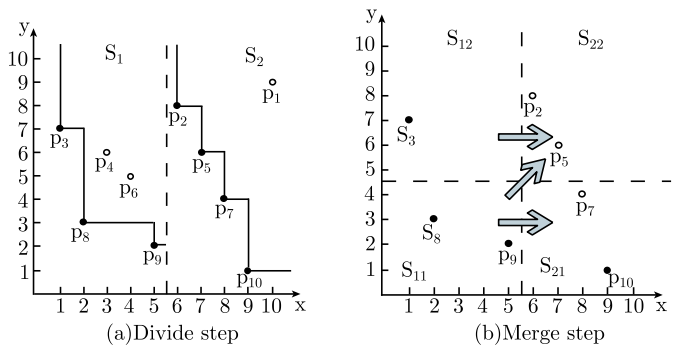


Figure 3. An example of DC algorithm

In *merge step*, D&C merges the local skylines by eliminating the data points of the local skyline of S_2 which are dominated by the local skyline points of S_1. To efficiently eliminate such points, the local skyline points of S_1 and S_2 are further partitioned by using the median of the local skyline points of S_1 on dimension y. These partitions are shown in Fig. 3(b). Note that every data point in S_{22} is dominated by every point in S_{11}. Hence, all the points in S_{22} (i.e., p_2 and p_5) can be eliminated. Furthermore, none of the points in S_{21} is dominated by any point in S_{12} because each point in S_{21} has a smaller value in y dimension. Hence, none of the points in S_{12} can

eliminate a point of S_{21}. Consequently, D&C only needs to check whether a point in S_{11} dominates a point in S_{12}. It is found that p_7 is dominated by p_9. Hence, p_7 is removed. So, the overall skyline is $\{p_3, p_8, p_9, p_{10}\}$.

Analysis of D&C. The partitioning phase of D&C requires to read and write the dataset at least once, thus incurring significant IO overhead. To improve the performance of D&C when limited main memory is available, M-way DC algorithm is proposed in Ref. [4]. We omit the details due to the space limitations.

2.2 Block nested loops algorithm (BNL)

A straightforward way to compute the skyline is to compare each data point p in the dataset D with all the other data points in D. A point p is reported as a skyline point if it is not dominated by any other point. Based on this concept, BNL algorithm[4] maintains a *window* of candidate skyline points in the main memory and evaluates the data points in the dataset one by one. During evaluation, p is compared against all the candidate points in the window. There may be the following three cases.

Case 1: p is dominated by a candidate point in the window. If so, p is discarded immediately without further comparison with other points.

Case 2: p dominates some candidate points in the window. In this case, the dominated candidates points are removed from the window and p is inserted into the window as a new candidate point.

Case 3: p is neither dominated by nor dominates any candidate point in the window. In this case, p is inserted into the window as a new candidate point.

After all the data points are evaluated, the candidate points in the window are reported as the skyline points. If the window grows larger than the main memory, BNL adopts a temporary file to store the candidate points that satisfy Case 3. After the whole dataset is scanned, all the candidate points in the window which were evaluated before the creation of the temporary file are output as the skyline points. Then, BNL evaluates all data points in the temporary file in the same way again until no temporary file is created.

Analysis of BNL. The advantage of BNL is that it can be used for any dimensionality without indexing or sorting. However, it relies on main memory to store the candidate skylines and a small memory may lead to numerous iterations.

2.3 Sort first skyline (SFS)

SFS[8] is a variation of BNL. In order to improve BNL, SFS introduces the *entropy* value $E(p)$ for each data point $p = (p[1], p[2], \cdots, p[d])$

$$E(p) = \sum_{i=1}^{d} \ln(p'[i] + 1)$$

where $p'[i]$ is the normalized value of $p[i]$ and d is the dimensionality. Obviously, given two data points p and q, p cannot dominate q if $E(p)$ is greater than or equal to $E(q)$. Based on the this observation, SFS first sorts all the data points in non-decreasing

order of their entropy values. After that, SFS processes the sorted dataset in the same way as BNL.

Analysis of SFS. Compared with BNL, SFS possesses the *progressive* nature which means skylines could be output without accessing the whole dataset. This is because the pre-sorting ensures that a point p can be dominated by only the points that are visited before p. This also alleviates BNL's problem of reliance on main memory. Nevertheless, SFS has to scan the entire data file to return a complete skyline.

2.4 Linear elimination sort for skyline (LESS)

LESS[29] improves SFS by integrating external merge sort procedure tightly into skyline computation. Similarly to SFS, LESS first sorts the dataset according to entropy values of all data points and then computes the skyline in the same way as BNL. In order to eliminate the data points efficiently, LESS makes the following two major changes during external merge sorting procedure.

1. It maintains an *elimination-filter* (EF) window in pass 0 of the external merge sort procedure to eliminate some non-skyline data points; and
2. It combines the final pass of the external merge sort procedure with the skyline examination procedure of BNL algorithm.

Analysis of LESS. Compared with SFS, LESS consistently performs better because (1) the dataset to be processed after pass 0 in LESS is smaller than that of SFS; this may also require SFS to do more passes in order to complete the sort; and (2) LESS saves at least one pass since it combines the final merge pass with the skyline examination procedure. It has been proven[29] that the average-case running time of LESS is $O(dn)$ where d and n are dimensionality and cardinality of the dataset, respectively.

In addition, Ref. [3] computes the skyline based on sorting and further improves the performance of Ref. [8]. Reference [42] applies the sort based paradigm to conduct skyline computation on high dimensions.

3 Skyline Computation Over Indexed Exact Data

In many real applications, the size of dataset might be massive and it is desirable to develop the indexing based techniques such that the performance of the skyline computation can be significantly improved in terms of CPU and I/O costs by taking advantage of the pre-computed indexes. In this section, we introduce the algorithms that utilize the indexes such as *Bitmap*[33], *Index*[33], *Nearest Neighbor Skyline*[13], *Branch and Bound Skyline*[26] and Z-order[21] algorithms.

3.1 Bitmap

Bitmap algorithm[33] encodes each data point with a *bitmap* according to the rank of its value on each dimension. The bitmaps enable the algorithm to efficiently determine whether a data point is a skyline point or not by bitwise operations (i.e., AND). Consider a data point $p = (p[1], p[2], \cdots, p[d])$, where d is the dimensionality. Each coordinate $p[i]$ $(1 \leqslant i \leqslant d)$ is converted into m_i-bit vector, where m_i is the number of distinct values on the i-th dimension, in which the $(m_i - rank(p[i]) + 1)$

most significant bits are 1 and others are 0. After conversion, every data point is mapped to an m-bit vector, where $m = \sum_{i=1}^{d} m_i$.

As an example, assume that there are 10 points in the 2-dimensional space as shown in Table 1. Since rank of $p_{10}[1]$ is 9 in the first dimension, it is converted to vector 1100000000. Similarly, all the bits of the corresponding vector to $p_{10}[2]$ are 1 because $p_{10}[2]$ is the smallest value in the second dimension (i.e., $rank(p_{10}[2]) = 1$).

Table 1 An example of Bitmap algorithm

Id	Coordinates	Bit Vectors
p_1	$(10, 9)$	$(1000000000, 100000000)$
p_2	$(6, 8)$	$(1111100000, 110000000)$
p_3	$(1, 7)$	$(1111111111, 11000000)$
p_4	$(3, 6)$	$(1111111100, 111100000)$
p_5	$(7, 6)$	$(1111000000, 111100000)$
p_6	$(4, 5)$	$(1111111000, 111110000)$
p_7	$(8, 4)$	$(1110000000, 111111000)$
p_8	$(2, 3)$	$(1111111110, 111111100)$
p_9	$(5, 2)$	$(1111110000, 111111110)$
p_{10}	$(9, 1)$	$(1100000000, 111111111)$

After converting all data points to bitmaps, every data point can be efficiently determined whether it belongs to the skyline by calling bitwise operations on the bitmaps. Specifically, given a data point $p = (p[1], p[2], \cdots, p[d])$, Bitmap algorithm first generates d bit vectors $b_1, b_2, \cdots, b_d$, where b_i $(1 \leqslant i \leqslant d)$ is juxtaposing the corresponding $rank(p[i])$ bits of every data point. The 1's in the result of b_1 & b_2 & ... & b_d indicate the data points which dominate p. Obviously, if there is only single 1 in the result, the considered data point is a skyline point.

Continuing the above example to check whether p_7 is a skyline point. For p_7, the corresponding bit vectors b_1 and b_2 are 0111111110, and 0000001111, respectively. Then, the result of b_1 & b_2 is 0000001110 which indicates p_7 is dominated by p_8 and p_9. As a result, p_7 is not a skyline point. On the other hand, for data point p_8, the result of b_1 & b_2 is 0010000100 & 0000000111 = 0000000100, which has only single 1. Therefore, p_8 belongs to the skyline. To obtain the entire skyline, Bitmap algorithm repeats the same examination for every data point in the dataset.

Analysis of Bitmap. The computation of the entire skyline using Bitmap is expensive because for each inspected point, the bitmaps of all points must be retrieved in order to obtain the juxtapositions. Furthermore, the space consumption may be prohibitive if the number of distinct values (i.e., m) is large.

3.2 Index algorithm

Given a set of d-dimensional data points, Index algorithm[33] organizes the data set into d B^+-tree indices. A data point $p = (p[1], p[2], \cdots, p[d])$ is assigned to the i-th $(1 \leqslant i \leqslant d)$ B^+-tree index if and only if $p[i]$ is the minimum coordinate among all coordinates of p. The key of each B^+-tree index is the minimum coordinate (denoted by $minValue$) of each data point. The data points in the same B^+-tree index which have same $minValue$ are maintained in a batch.

To compute skyline, the maximum value of all the coordinates of the current skyline points is maintained, which is denoted as $maxValue$. The algorithm iteratively examines each B^+-tree index and processes the batch which has the smallest $minValue$. $minValue$ of this batch is first compared with $maxValue$. Obviously, if $maxValue$ is smaller than or equal to $minValue$, then there exists a skyline point in the current skyline which dominates the data points in this batch as well all other unprocessed data points in this B^+-tree index. Therefore, the batch and all other unprocessed data points in the same B^+-tree index can be safely discarded. Otherwise, within the batch processed, a local skyline is computed first. Then, the points of this local skyline are compared with the points maintained in the current skyline. If one of these points is dominated by any current skyline point, it is discarded. Otherwise, it is inserted into the skyline as a new skyline point. Once a new skyline point is found, $maxValue$ is updated. Index algorithm returns the skyline result after the batches of all B^+-tree indices are processed.

Consider the example of dataset shown in Table 2. All the batches in the 2 B^+-tree indices are listed in Table 2 in the increasing order of $minValue$. Initially, Index algorithm processes the batches with $minValue = 1$ one by one. Data points p_3 and p_{10} are added into the skyline as they are not dominated by the current skyline points. After that, $maxValue$ is updated to 9. Similarly, p_8 and p_9 are added into skyline after their batches are processed. $maxValue$ is not changed after these two new skyline points are found. Then, p_4 is examined and discarded as it is dominated by some current skyline point (i.e., p_8). Index algorithm continues to examine the remaining batches in the increasing order of its $minValue$. When p_1 is processed, since its $minValue = 9$ is equal to $maxValue$, p_1 is discarded immediately without being evaluated with the current skyline points. After all batches are processed, Index algorithm outputs the skyline $\{p_3, p_{10}, p_8, p_9\}$.

Table 2 An example of Index algorithm

Index 1		**Index** 2	
$minValue$	**batch**	$minValue$	**batch**
1	$p_3(1,7)$	1	$p_{10}(9,1)$
2	$p_8(2,3)$	2	$p_9(5,2)$
3	$p_4(3,6)$	4	$p_7(8,4)$
4	$p_6(4,5)$	6	$p_5(7,6)$
6	$p_2(6,8)$	9	$p_1(10,9)$

Analysis of Index Algorithm. Index algorithm is progressive in nature, i.e., it can return skyline points as they are found. However, it cannot output skyline points according to a user-defined order. Furthermore, as indicated in Ref. [13], Index algorithm cannot support skylines retrieval on arbitrary subset of dimensions since the B^+-tree that a point belongs to may change with a different subset of dimensions.

3.3 Nearest neighbor based algorithm (NN)

NN[13] is based on the following fundamental observation.

Observation: *Given a dataset and a monotonic distance function f (e.g., Euclidean distance), the nearest neighbor of the origin is a skyline point.*

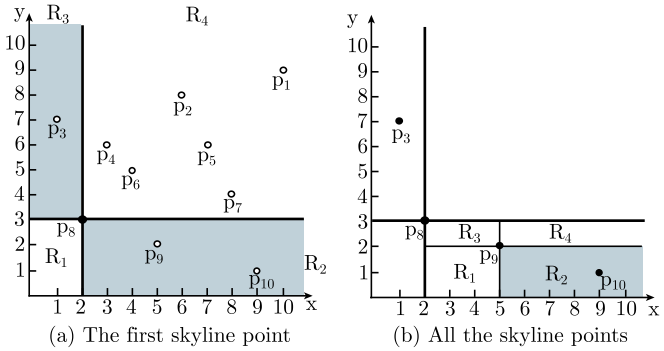
This observation can be easily proven by contradiction. Based on the above observation, to compute the skyline for the given dataset, NN first finds the nearest neighbor p of the origin. Then, the algorithm partitions the data space into 3 parts with respect to p (see Fig. 6(a) where p_8 is the nearest neighbor of the origin).

Part 1: The hyper rectangle with the origin as lower-left corner and p as upper-right corner (i.e., R_1 shown in Fig. 4(a)). According to the above observation, this part is empty as no data point dominates p.

Part 2: The hyper rectangle with p as lower-left corner and the upper-right corner of the data space as upper-right corner (i.e., R_4). Obviously, all data points in this part are dominated by p. Therefore, these data points can be discarded safely.

Part 3: The other regions (i.e., R_2 and R_3). The property of those regions is that their local skyline points belong to the global skyline as p does not dominate any data point in those regions. As a result, NN is recursively applied to these regions till the whole data space is evaluated.

As shown in Fig. 4(a), after the nearest neighbor of the origin (p_8) is found, the data space is partitioned into 4 parts, R_1, R_2, R_3, and R_4. As stated above, NN only needs to consider regions R_2 and R_3. For each region, NN recursively finds the nearest neighbor of its lower-left corner and partitions it into sub-regions. For the partition R_3, since there is only one data point p_3, it is returned as a skyline point and the search process in this region is terminated. Similarly, p_9 is found as the nearest neighbor of the lower-left corner of R_2. Hence, R_2 is further partitioned into four regions (as shown in Fig. 4(b)). After the last data point p_{10} is found as the nearest neighbor of its sub-region, NN terminates and outputs all the nearest neighbors found as the skyline.



(a) The first skyline point (b) All the skyline points

Figure 4. An example of NN algorithm

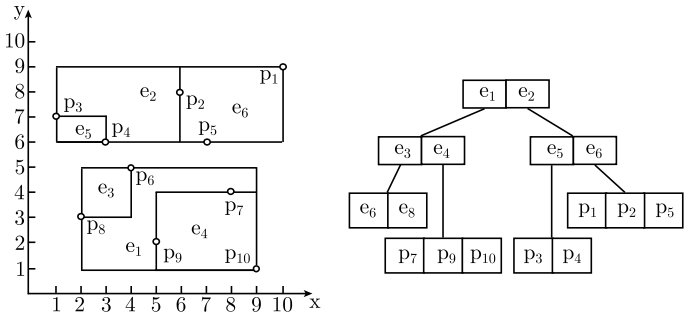
In general, for multiple dimensional space ($d > 2$), the regions in part 3 may overlap. Elimination methods are also proposed in NN[13] to eliminate such duplicates.

Analysis of NN. The performance of NN is significantly impacted due to the expensive eliminations of duplicates when dimensionality is higher than 2.

3.4 Branch and bound skyline algorithm (BBS)

Like NN algorithm, BBS[26] is also based on nearest neighbor search. The dataset is indexed by an R-tree. To find the skyline, BBS traverses the R-tree in a *best-first* manner, i.e., it always evaluates and expands the node that is closest to the origin among all un-visited nodes. To do that, BBS employs a heap in which the key of each entry (i.e., R-tree node or data point) is its minimum distance to the origin. Here, the minimum distance of an R-tree node to the origin is the summation of the coordinates of its lower-left corner. Initially, all child entries of the root node of the R-tree are inserted into the heap. In each iteration, the top entry e is removed from the heap and examined against the skyline computed so far. If e is dominated by a current skyline point, e is discarded. Otherwise, e is either expanded or output as a skyline point depending on whether it is a node or a data point. More specifically, if e is an R-tree node, it is expanded by inserting all its child entries which are not dominated by any current skyline point into the heap. If e is a data point, it is output as a new skyline point. BBS terminates when the heap becomes empty.

Consider the dataset shown in Fig. 5(a). Its corresponding R-tree is illustrated in Fig. 5(b). To compute skyline, as listed in Table 3, BBS first inserts entries e_1 and e_2 into the heap. Then, e_1, which is closer to origin than e_2, is expanded to entries e_3 and e_4. The next top entry is e_3 and its data points p_8 and p_6 are inserted into the heap. Next, p_8 is processed. As it is not dominated by any current skyline point (the current skyline is empty), it is output as a new skyline point. Similarly, entry e_4 is expanded and its data points (p_7, p_9 and p_{10}) are inserted into the heap.



(a) Dataset (b) R-tree

Figure 5. An example dataset and its R-tree

Table 3 An example of BBS algorithm

Action	Heap	Skyline
initializing	$\langle e_1, 3\rangle$, $\langle e_2, 7\rangle$	$\varnothing$
expand e_1	$\langle e_3, 5\rangle$, $\langle e_4, 6\rangle$, $\langle e_2, 7\rangle$	$\varnothing$
expand e_3	$\langle \mathbf{p_8}, \mathbf{5}\rangle$, $\langle e_4, 6\rangle$, $\langle e_2, 7\rangle$, $\langle p_6, 9\rangle$	p_8
expand e_4	$\langle \mathbf{p_9}, \mathbf{7}\rangle$, $\langle e_2, 7\rangle$, $\langle p_6, 9\rangle$, $\langle p_{10}, 10\rangle$, $\langle p_7, 12\rangle$	p_8, p_9
expand e_2	$\langle e_5, 7\rangle$, $\langle p_6, 9\rangle$, $\langle p_{10}, 10\rangle$, $\langle p_7, 12\rangle$	p_8, p_9
expand e_5	$\langle \mathbf{p_3}, \mathbf{8}\rangle$, $\langle p_6, 9\rangle$, $\langle p_{10}, 10\rangle$, $\langle p_7, 12\rangle$	p_8, p_9, p_3
examine p_6, p_{10}, p_7	$\varnothing$	p_8, p_9, p_3, p_{10}

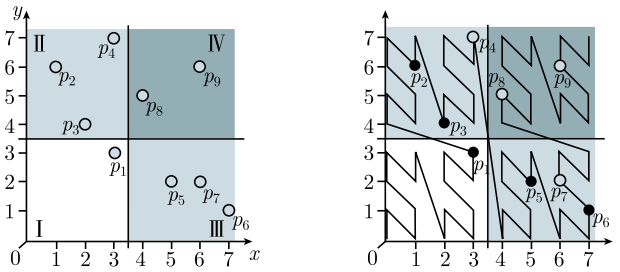
When e_2 is expanded, its child entry e_6 is found being dominated by one skyline point

(i.e., p_8) and it is discarded. For the same reason, when e_5 is processed, the data point p_3 is inserted into the heap whereas p_4 is discarded. BBS continues to the process in this way till all entries are processed and the skyline $\{p_8, p_9, p_3, p_{10}\}$ is returned.

Analysis of BBS. BBS is progressive and IO optimal – the number of nodes of R-tree accessed is minimized.

3.5 Z-Order based algorithm (Z-order)

Reference [21] considers both query efficiency and update efficiency and proposes an index structure called ZBtree based on Z-order curve. As shown in Fig. 6(a), all points in Region IV are dominated by points in Region I. This means we can eliminate Region IV if Region I is non-empty. Points in Region II and III do not dominate each other but could be dominated by the points in Region I. Thus, a natural access order should be I, II, III, IV (or I, III, II, IV), which forms a Z-order curve. Similar philosophy applies to the points inside each subregion recursively until there is only one point in the subregion. The proposed ZB-tree is a modification of B^+-tree to index these Z-order curves. To retrieve skyline objects based on ZB-tree, block based dominance tests are conducted to avoid dominating tests of unpromising regions (e.g., Region IV in Fig. 6) or unnecessary dominance checks among different regions (e.g., Regions II and III in Fig. 6). The update operations such as insertion and deletion are shown to be more effective than BBS[26]. Reference [17] further improves Ref. [21] by combining ZB-tree with a nested coding scheme and also supports skylines on partial orders.



(a) 2D Data Space (b) Z-order Curve

Figure 6. Example of Z-order curve

In addition, a very recent work[31] provides an external memory algorithm for computing skyline of a d-dimensional dataset. The algorithm improves the I/O complexity from $O((N/B)log_2^{(d-2)}(N/M))$ to $O((N/B)log_{M/B}^{(d-2)}(N/M))$ for $d \geq 3$, where N is the cardinality of the dataset, M is the capacity of main memory, and B is the size of a disk block.

4 Skyline Computation Over Uncertain Data

As mentioned in the introduction, it is non-trivial to capture the dominance relationship between two uncertain objects because there are multiple instances in each uncertain object. In this section, we introduce four probabilistic skyline models and their corresponding skyline computation algorithm.

4.1 Threshold based probabilistic skylines (PSkyline)

The problem of skyline computation over uncertain data is firstly studied in Ref. [24] following the possible world semantics. As shown in Fig. 7, each uncertain object may have multiple instances where each instance represents a possibility of the uncertain object. If the uncertainty of an uncertain object U is modeled continuously, the probability for U to be a skyline object is:

$$Pr(U) = \int_{u \in U} f(u) \prod_{\forall V \neq U} (1 - \int_{v \prec u} f'(v)dv)du \tag{4.1}$$

Here $\prod_{V \neq U}(1 - \int_{v \prec u} f'(v)dv)$ is the probability that the point $u \in U$ is not dominated by any other uncertain object. f and f' denotes the PDF of U and V, respectively. In a discrete case, the skyline probability of U is:

$$Pr(U) = \sum_{u \in U} (P(u) \times \prod_{\forall V \neq U} (1 - \sum_{v \in V, v \prec u} P(v))) \tag{4.2}$$

$\prod_{V \neq U}(1 - \sum_{v \in V, v \prec u} p(v))$ is the probability that $u \in U$ is not dominated by any other object. Recall that $P(u)$ denotes the appearance probability of instance u.

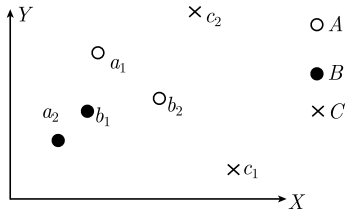


Figure 7. Uncertain objects

Given a set of uncertain objects $\mathcal{U}$, PSkyline aims to retrieve all objects from $\mathcal{U}$ with skyline probabilities not smaller than a given probability threshold γ. Bounding-pruning-refining iteration is deployed to achieve efficiency. Two algorithms, bottom-up and top-down, are developed. The bottom-up algorithm computes $Pr(U)$ from instance level. After calculating skyline probabilities of some selected instances, these values are used to prune other instances and objects. Top-down algorithm, on the other hand, partitions instances of one uncertain object into several groups and apply pruning techniques in the group and object level.

4.2 Computing all skyline probabilities

While Ref. [24] solves the case of probabilistic skyline computation with a pre-given threshold, Ref. [2] studies the problem of computing skyline probabilities for every object in the uncertain database. The time complexity to compute all skyline probabilities is sub-quadratic by using space partitioning and dominance counting algorithms. In Ref. [2], the assumption of equal instances' probabilities is removed and total occurrence probability of instances from the same uncertain object could be smaller than 1.

Suppose there are m uncertain objects denoted as $O_1, \cdots, O_m$. For each object O_i, let S_i denote the set of n_i instances of O_i. Use S to denote the set of all instances from m objects and n to denote $|S|$, namely n is the total number of instances from m uncertain objects. For each instance $p \in S$, $Pr(p)$ denotes its probability. Suppose p is an instance from object O_j. Then, the skyline probability of p is as follows.

$$Pr_{sky}(p) = Pr(p) \times \prod_{i=1, i \neq j}^{m} (1 - \sum_{p' \in D_{S,i}(p)} Pr(p'))$$

Here $D_{S,i}(p)$ denotes the set of instances of object O_i in S that dominate p and is called the dominant set of p. Let $\beta(p)$ denote $\prod_{i=1, i \neq j}^{m} \left(1 - \sum_{p' \in D_{S,i}(p)} Pr(p')\right)$. The skyline probability of an uncertain object O_i is simply the sum of skyline probabilities of its instances, i.e.,

$$Pr_{sky}(O_i) = \sum_{p \in O_i} Pr_{sky}(p)$$

As shown in Fig. 8, all instances in S forms a grid with n^2 instances. For an instance p, $\beta(p)$ could be computed using the grid by accumulating the probability of instances dominating p along each dimension. The weighted dominance counting problem is for a given set S of n weighted points, to compute for each point p of S the sum of the weights of all points in S that dominate p[25]. The grid method and the weighted dominance counting techniques are combined together to yield a sub-quadratic time complexity algorithm to compute skyline probabilities for each uncertain object.

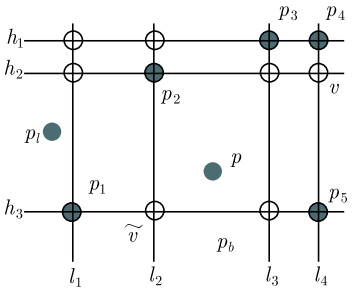


Figure 8. Grids

4.3 Top-k probabilistic skyline operator

Instead of requiring a probability threshold, the top-k probabilistic skyline operator outputs k uncertain objects from the data set with the highest skyline probabilities. Both discrete and continuous cases are tackled in Ref. [43].

Given a set of uncertain objects $\mathcal{U} = \{U_1, \cdots, U_n\}$ such that each U_i has a PDF f_{U_i} defined on U_i. The possible world semantic can be extended to cover the continuous case as follows. A *possible world* $W = \{u_1, u_2, \cdots, u_n\}$ is a point in the space $\Omega = \prod_{i=1}^{n} U_i$ such that $\int_{W \in \Omega} \prod_{i=1}^{n} f_{U_i}(u_i) du_1 du_2 \cdots du_n = 1$. $SKY(W)$ is

defined as the objects with a point in the skyline of W. The skyline probability of U is

$$P_{sky}(U) = \int_{U \in SKY(W), W \in \Omega} \prod_{i=1}^{n} f_{U_i}(u_i) du_1 du_2 \cdots du_n. \tag{4.3}$$

This can be rewritten as:

$$P_{sky}(U) = \int_{u \in U} f_U(u) \prod_{V \neq U} (1 - \int_{v \prec u, v \in V} f_V(v) dv) du. \tag{4.4}$$

The framework to retrieve top-k objects with the highest skyline probabilities follows the seeding and refinement paradigm. BBS algorithm[26] is also adopted in the framework. For the continuous case, a randomized algorithm is proposed with accuracy guarantee.

4.4 Stochastic skylines

The research in ranking uncertain objects against multiple criteria has a long history in economics, finance, and mathematics; see Refs. [12, 18, 30] for example. The *expected utility principle* is the most popular model[12,18] to select the optimal uncertain object against multiple criteria. In the light of the expected utility principle, given a function f, an uncertain object U with the maximum expected utility is the optimal solution; that is, select U to maximize $E[f(U)]$ for a given utility function f.

Recall that skyline provides the minimal set of candidates by removing the points not preferred by any decreasing scoring functions. While the skyline probabilities can effectively capture the possible dominance relationships among uncertain objects, skyline probabilities do not provide the candidature of optimal solutions regarding the expected utility principle. This motivates the study of modeling skyline operator over uncertain objects using stochastic orders. Two types of stochastic skyline operators are proposed so far[22,41], where the stochastic skyline operator based on lower orthant order (lskyline) provides the minimum candidate set of optimal solutions to all multiplicative monotonic scoring functions and the stochastic skyline operator based on usual order (gskyline) provides the minimum candidate set of optimal solutions to all monotonic scoring functions.

Stochastic orders have been widely used in many real-life applications[12,18,30], including economics, finance, and multi-criteria statistic decision making to facilitate the expected utility principle. Generally, given a family $\mathcal{F}$ of utility (scoring) functions from all users, an uncertain object (random variable) U *stochastically dominates* V regarding $\mathcal{F}$, denoted by $U \prec_{\mathcal{F}} V$ if and only if $E[f(U)] \geqslant E[f(V)]$ for each $f \in \mathcal{F}$ (see Ref. [12]); that is, all users prefer U to V according to the expected utility principle. Given a set $\mathcal{U}$ of uncertain objects, the *stochastic order based skyline* regarding $\mathcal{F}$ is the subset of $\mathcal{U}$ such that each object U in the stochastic order based skyline is not stochastically dominated by any other object in $\mathcal{U}$ regarding $\mathcal{F}$. Based on the above definition of stochastic orders, it is immediate that the stochastic order based skylines regarding $\mathcal{F}$ provide the minimum set of candidates to the optimal solutions (maximum values) for all functions in $\mathcal{F}$ by removing the objects not preferred by any function in $\mathcal{F}$.

In Fig. 9, assume that A has 3 instances a_1, a_2, and a_3 with the occurrence

probabilities $\frac{1}{2}$, $\frac{1}{7}$, $\frac{5}{14}$, respectively, assuming the game performance a_1 occurs 7 times, a_2 occurs 2 times, and a_3 occurs 5 times. Suppose that B has 2 instances b_1 and b_2 with the occurrence probabilities $\frac{1}{2}$ and $\frac{1}{2}$, respectively; and C has 3 instances c_1, c_2, and c_3 with the occurrence probabilities $\frac{1}{8}$, $\frac{1}{8}$, and $\frac{3}{4}$ respectively. Immediately, the expected utility of A regarding the 3 instances regarding a utility function f is $E[f(A)] = \frac{1}{2}f(a_1) + \frac{1}{7}f(a_2) + \frac{5}{14}f(a_3) = \frac{1}{2}f(1,4) + \frac{1}{7}f(2,1) + \frac{5}{14}f(3,2)$. Similarly, we can calculate $E[f(B)]$ and $E[f(C)]$. It can be immediately verified that for any nonnegative decreasing function f, $E[f(A)] \geqslant E[f(B)]$; that is, A is always preferred to (better than) B. Consequently, B should be *excluded* as a candidate for users to make a personal trade-off. However, it can be verified that the skyline probabilities of A, B, and C are $\frac{51}{56}$, $\frac{7}{16}$, and $\frac{3}{14}$ respectively. Clearly, B is always preferred to C based on the skyline probability values; that is, impossible to exclude the object B without excluding the object C based on skyline probabilities.

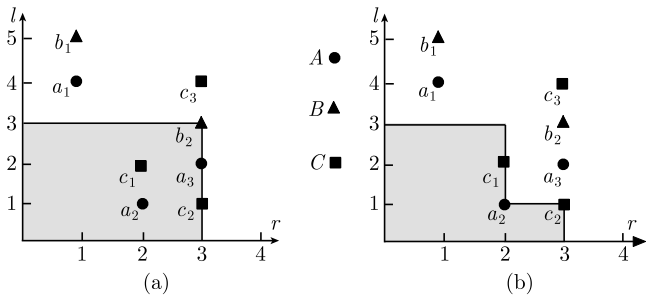


Figure 9. Motivating example for stochastic skylines

Lower Orthant Order based Skyline (lskyline).

For a point $x \in R_+^d$, the *probability mass* $U.cdf(x)$ of U is the sum of the probabilities of the instances in $R((0,\cdots,0),x)$ where $(0,0,\cdots,0)$ is the origin of R_+^d; that is, $U.cdf(x) = \sum_{u \preceq x, u \in U} p_u$. In Fig. 12(a), $A.cdf(b_2) = \frac{1}{2}$, $B.cdf(b_2) = \frac{1}{2}$ and $C.cdf(b_2) = \frac{1}{4}$.

Below we give the definition of lower orthant order and lskyline.

[Lower Orthant Order.] Given two uncertain objects U and V, U *stochastically dominates* V regarding the lower orthant order, denoted by $U \prec_{lo} V$, if $U.cdf(x) \geqslant V.cdf(x)$ for each point $x \in R_+^d$ and $\exists y \in R_+^d$ such that $U.cdf(y) > V.cdf(y)$.

[lskyline.] Given a set of uncertain objects $\mathcal{U}$, $U \in \mathcal{U}$ is a lskyline object if there is no object $V \in \mathcal{U}$ such that $V \prec_{lo} U$. The set of all lskyline objects of $\mathcal{U}$ is the lskyline of $\mathcal{U}$.

In Ref. [22], it is proved that the problem of determining the lower orthant order between two objects is NP-complete regarding the dimensionality. A novel, efficient,

partition based algorithm is developed to verify if an uncertain object is stochastically dominated by another object based on the lower orthant order. The algorithm runs in polynomial time if the dimensionality is fixed. To retrieve lskyline from a given uncertain data set $\mathcal{U}$, Ref. [22] adopts the BBS paradigm Ref. [26]. Pruning techniques are also proposed based on distance and statistic information.

Usual Order based Skyline (gskyline).

Given a set $S \subseteq R_+^d$, $U.cdf(S)$ denotes the probability mass of U restricted to S; that is, $U.cdf(S) = \sum_{u \in S \cap U} p_u$. Regarding the shaded area S_1 in Fig. 9(a), $A.cdf(S_1) = \frac{1}{2}$, $B.cdf(S_1) = \frac{1}{2}$, and $C.cdf(S_1) = \frac{1}{4}$, while regarding the shaded area S_2 in Fig. 9(b), $A.cdf(S_2) = \frac{1}{7}$, $B.cdf(S_2) = 0$, and $C.cdf(S_2) = \frac{1}{4}$.

Next, we give the definition of lower set, usual order and gskyline.

[Lower Set.] A set S of points in R_+^d is a *lower set* if for each pair of points $x \preceq y$ in R_+^d, $y \in S$ implies $x \in S$.

The shaded areas in Figs. 9(a) and (b) are lower sets, respectively.

[Usual Order.] Given two uncertain objects U and V, U *stochastically dominates* an object V regarding the usual order, denoted by $U \prec_{uo} V$, if $U.cdf(S) \geqslant V.cdf(S)$ for each lower set $S \subseteq R_+^d$ and $U.cdf(S') > V.cdf(S')$ for a lower set $S' \subseteq R_+^d$.

Regarding the depicted lower set S_1 (shaded area) in Fig. 9(a), since $A.cdf(S_1) = \frac{1}{2}$ and $C.cdf(S_1) = \frac{1}{4}$, $C.cdf(S_1) < A.cdf(S_1)$; thus, $C \not\prec_{uo} A$. Regarding the depicted lower set S_2 (shaded area) in Fig. 9(b), since $A.cdf(S_2) = \frac{1}{7}$ and $C.cdf(S_2) = \frac{1}{4}$, $A.cdf(S_2) < C.cdf(S_2)$; thus, $A \not\prec_{uo} C$.

[gskyline.] In a set $\mathcal{U}$ of uncertain objects, $U \in \mathcal{U}$ is a gskyline object if there is no object $V \in \mathcal{U}$ such that $V \prec_{uo} U$. The set of all gskyline objects of $\mathcal{U}$ is the gskyline of $\mathcal{U}$.

In Ref. [41], it is shown that the problem of testing the usual order can be solved in polynomial time though it involves more complex geometric forms in the definition than that in the definition of the lower orthant order. The problem of testing usual order is converted to the max-flow problem, and then novel techniques are proposed to effectively reduce the size of auxiliary networks to significantly speed up the testing. To retrieve gskyline from a set of uncertain objects $\mathcal{U}$, the framework is similar to Ref. [22] except that the pruning techniques are more sophisticated.

5 Variations of Skyline

We briefly introduce the variations of the skyline operator in this section.

Reverse skyline. Given a set of multidimensional objects D and a query object q, the dynamic skyline corresponds to the skyline of a transformed data space where point q becomes the origin and all points of D are represented by their distance vector to q. The *reverse skyline* query retrieves the objects whose dynamic skyline contains the query object q. Reverse skylines are first studied in Ref. [10]. Reverse skylines on uncertain data are studied in Ref. [15] following the possible world semantics.

Top-k dominating queries. *Top-k dominating* queries combine the advantages of top-k and skyline queries[38]. Instead of retrieving the objects which are not dominated by other objects, a top-k dominating query returns k objects with the highest dominating abilities. The dominating ability of an object p is measured by the number of points dominated by p. This problem is first tackled in Ref. [38] and the probabilistic top-k dominating queries are studied in Refs. [16, 40].

Top-k representative skyline. The size of skylines could be large and difficult to analyze in some cases. *Top-k representative skyline* identifies k objects that best capture the full skyline information. Reference [20] first studies representative skylines with the aim that the returned k skyline objects together cover as many objects from the database as possible. References [1, 32] tackles the problem from the aspect of distances and distributions, respectively.

Spatial skyline. Spatial preference queries are often used to suggest the objects based on their spatial proximity to the facilities in many user recommendation systems[36]. The *spatial skyline* provides the minimal candidate set of the optimal solutions for any monotonic distance based spatial preference query[23].

Skycube. Suppose that the skyline operator is defined on d dimensions. Different users may have different preferences and would like to select the skylines based on different combinations of the d dimensions. *Skycubes*, or *subspace skylines*, address this problem by retrieving skylines of all possible non-empty subsets of a given set of d dimensions[27,37].

k-skyband. Recall that the skyline provides minimum candidate set for the optimal solution of all monotonic ranking functions, namely, top-1 result for all monotonic ranking functions. The *k-skyband* operator, on the other hand, retrieves the candidate set of top-k results for all monotonic ranking functions. Reference [35] studies k-skyband queries on both full spaces and subspaces.

k dominant skyline. In high dimensional space, the chances that a point dominates another point become small. This leads to a large size of the skyline and may fail to provide interesting insights to users. *k dominant skyline* query relaxes the definition of skylines in that one point p dominates another point q if p is better than or equal to q in at least k dimensions ($k \leq d$ where d is the total number of dimensions) and is better than q in at least one of these k dimensions[9].

Skyline over partially ordered domains. In many applications, the attributes may be partially ordered. The traditional skyline techniques are all based on the assumptions that attributes are all totally ordered. Reference [7] is the first to study skylines over partially ordered domains.

Skyline over sliding windows. In many important applications, the data arrives continuously in a streaming fashion. Usually users are interested in the most recent information with a window size N. This is also called the sliding window model. Skyline computation over sliding windows are studied in Refs. [19, 34]. Specifically, Ref. [19] supports skyline computation of any recent n elements as long as $n \leq N$. Skyline computation over uncertain data streams is studied in Ref. [39].

Parallel Skyline. Since skylines are computationally expensive, Ref. [14] propose techniques to compute skyline using parallel processing.

Distributed Skyline. In various applications, data is stored and processed in a distributed way. The distribution of content and lack of global knowledge pose

challenges for skyline processing. Skyline and constrained skyline queries processing are studied in Refs. [6, 28].

6 Conclusions and Future Work

The skyline operator is a useful tool in multi-criteria decision making since it possesses many desirable features such as it provides a minimal candidate set to the optimal solutions of all monotonic ranking functions. Furthermore, the skyline operator is not influenced by value ranges on different dimensions and requires no pre-given ranking functions. We summarize the techniques for skyline computation over both exact and uncertain data, as well as variations of skylines. Some future directions on skylines processing include skylines in the cloud environment and over complex unstructured data (e.g., graphs).

References

[1] Aarma AD, Lall A, Nanongkai D, Lipton RJ, Xu J. Representative skylines using threshold-based preference distributions. ICDE. 2011.
[2] Atallah MJ, Qi Y. Computing all skyline probabilities for uncertain data. PODS. 2009.
[3] Bartolini I, Ciaccia P, Patella M. Efficient sort-based skyline evaluation. TODS. 2008.
[4] Borzsonyi S, Kossmann D, Stocker K. The skyline operator. ICDE 2001. 2001.
[5] Bentley JL, Kung HT, Schkolnick M, Thompson CD. On the average number of maxima in a set of vectors and applications. J. ACM, 1978.
[6] Chen L, Cui B, Lu H. Constrained skyline query processing against distributed data sites. TKDE. 2011.
[7] Chan CY, Eng PK, Tan KL. Stratified computation of skylines with partially-ordered domains. SIGMOD. 2005.
[8] Chomicki J, Godfrey P, Gryz J, Liang D. Skyline with pre-sorting. ICDE. 2003.
[9] Chan CY, Jagadish HV, Tan KL, Tung AKH, Zhang Z. Finding k-dominant skylines in high dimensional space. SIGMOD. 2006.
[10] Dellis E, Seeger B. Efficient computation of reverse skyline queries. VLDB. 2007.
[11] Kung HT, Luccio F, Preparata FP. On finding the maxima of a set of vectors. In *JACM*, 1975.
[12] Kijima M, Ohnishi M. Stochastic orders and their applications in financial optimization. Mathematical Methods of Operations Research, 1999, 50(2).
[13] Kossmann D, Ramsak F, Rost S. Shooting stars in the sky: an online algorithm for skyline queries. VLDB. 2002.
[14] Kohler H, Yang J, Zhou X. Efficient parallel skyline processing using hyperplane projections. SIGMOD. 2011.
[15] Lian X, Chen L. Monochromatic and bichromatic reverse skyline search over uncertain databases. SIGMOD. 2008.
[16] Lian X, Chen L. Top-k dominating queries in uncertain databases. EDBT. 2009.
[17] Liu B, Chan CY. Zinc: Efficient indexing for skyline computation. VLDB. 2010.
[18] Levy H. Stochastic dominance and exptected utility: survey and analysis. Management Science, 1992, 38(4).
[19] Lin X, Yuan Y, Wang W, Lu H. Stabbing the sky: Efficient skyline computation over sliding windows. ICDE. 2005.
[20] Lin X, Yuan Y, Zhang Q, Zhang Y. Selecting stars: The k most representative skyline operator. ICDE. 2007.
[21] Lee KCK, Zheng B, Li H, Lee WC. Approaching the skyline in z order. VLDB. 2007.
[22] Lin X, Zhang Y, Zhang W, Cheema MA. Stochastic skyline operator. ICDE. 2011.
[23] Lin Q, Zhang Y, Zhang W, Li A. General spatial skyline operator. DASFAA. 2012.
[24] Pei J, Jiang B, Lin X, Yuan Y. Probabilistic skylines on uncertain data. VLDB. 2007.
[25] Preparata FP, Shamos MI. Computational geometry: An introduction. Springer-Verlag. 1985.

[26] Papadias D, Tao Y, Fu G, Seeger B. Progressive skyline computation in database systems. TODS. 2005.
[27] Pei J, Yuan Y, Lin X, Jin W, Ester M, Liu Q, Wang W, Tao Y, Yu JX, Zhang Q. Towards multidimensional subspace skyline analysis. TODS. 2006.
[28] Rocha-Junior JB, Vlachou A, Doulkeridis C, Norvag K. Efficient execution plans for distributed skyline query processing. EDBT. 2011.
[29] Ryan Shipley PG, Gryz J. Maximal vector computation in large data sets. VLDB. 2005.
[30] Shaked M, Shanthikumar JG. Stochastic Orders and Their Applications. Academic Press, 2007.
[31] Sheng C, Tao Y. On finding skylines in external memory. PODS. 2011.
[32] Tao Y, Ding L, Lin X, Pei J. Distance-based representative skyline. ICDE. 2009.
[33] Tan K, Ooi B. Efficient progressive skyline computation. VLDB. 2001.
[34] Tao Y, Papadias D. Maintaining sliding window skylines on data streams. TKDE. 2006.
[35] Tao Y, Xiao X, Pei J. Efficient skyline and top-k retrieval in subspaces. TKDE. 2007.
[36] Yiu ML, Dai X, Mamoulis N, Vaitis M. Top-k spatial preference queries. ICDE. 2007.
[37] Yuan Y, Lin X, Liu Q, Wang W, Yu JX, Zhang Q. Efficient computation of the skyline cube. VLDB. 2005.
[38] Yiu ML, Mamoulis N. Efficient processing of top-k dominating queries on multi-dimensional data. VLDB. 2007.
[39] Zhang W, Lin X, Zhang Y, Wang W, Yu JX. Probabilistic skyline operator over sliding windows. ICDE. 2009.
[40] Zhang W, Lin X, Zhang Y, Pei J, Wang W. Threshold-based probabilistic top-k dominating queries. VLDB J.. 2010.
[41] Zhang W, Lin X, Zhang Y, Cheema MA, Zhang Q. Stochastic skylines. TODS. 2012.
[42] Zhang S, Mamoulis N, Cheung DW. Scalable skyline computation using object based spation partitioning. SIGMOD. 2009.
[43] Zhang Y, Zhang W, Lin X, Jiang B, Pei J. Ranking uncertain sky: the probabilistic top-k skyline operator. Information Systems. 2011.