Applied GIS

ISSN: 1832-5505
Vol-4 Issue-04 Oct 2016

Questions About the Horizon: A Review

R Gaidhani[1], Avinash Potdar[2], Avinash C Taskar[3]
Department of Computer Engineering, Sandip Institute of Engineering and Management Abhar

Abstract**—** During the two past decades, skyline queries were used in several multi-criteria decision support applications.

Given a dominance relationship in a dataset, a skyline query returns the objects that cannot be dominated by any other objects. Skyline queries were studied extensively in multidimensional spaces, in subspaces, in metric spaces, in dynamic spaces, in streaming environments, and in time-series data. Several algorithms were proposed for skyline query processing, such as window- based, progressive, distributed, geometric-based, index-based, dividend-conquer, and dynamic programming algorithms.

Moreover, several variations were proposed to solve application-specific problems like *k*-dominant skylines, top-*k* dominating queries, spatial skyline queries, and others. As the number of objects that are returned in a skyline query may become large, there is also an extensive study for the cardinality of skyline queries. This extensive research depicts the importance of skyline queries and their variations in modern applications.

INTRODUCTION

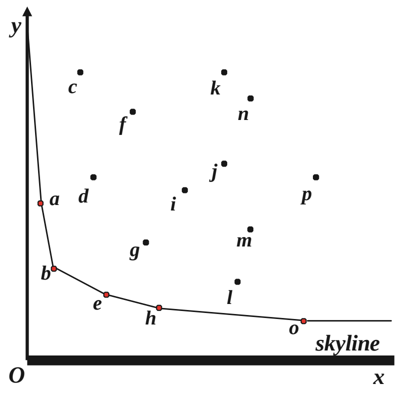
In the database world, skyline queries have been a hot topic for decades. For many multi-criteria decision making uses, the skyline computation has become indispensable. Many algorithms were suggested and intensively researched.

Definition

A skyline query finds the items in a dataset that are not dominated by any other objects, given a dominance relationship. To be considered superior to another item in a multidimensional dataset, it must be at least as excellent in all dimensions and superior in at least one. Similar to the well-known maximum vector issue, the specification of skyline queries in multidimensional datasets is the same [3, 22]. All data were supposed to be stored in memory in these first publications, which framed skyline computing as an algorithmic challenge. But these days, we have to deal with massive datasets that live in secondary memory. Both index-based and non-index-based techniques are presented for processing skyline queries with the data stored on disk(s).

Example

A typical example of a skyline query is when the data objects are two-dimensional points in the Euclidean plane, and the preference for each dimension is the minimum. Figure 1 depicts such an example for 16 points with coordinates: $a(1,12)$, $b(2,7)$, $c(4,22)$, $d(5,14)$, $e(6,5)$, $f(8,19)$, $g(9,9)$, $h(10,4)$, $i(12,13)$, $j(15,15)$, $k(15,22)$, $l(16,6)$, $m(17,10)$,



$n(17,20)$, $o(21,3)$, $p(22,14)$. The skyline query returns the objects $\{a,b,e,h,o\}$.

NON-INDEX-BASED ALGORITHMS

Block-Nested-Loop (BNL)

A naïve approach for calculating a skyline query would need a nested-loop across the whole dataset, comparing each item to itself. Due to its inefficient quadratic complexity O(N2), this approach is not practical for large datasets.

In a similar vein, the Block-Nested-Loop method [4] employs the same notion, although with a window (a memory block with constrained size) that can only store a certain number of data items. whether object p is a potential contender, we check

Applied GIS

ISSN: 1832-5505

Vol-4 Issue-04 Oct 2016

to see whether any other objects in the window dominate it. In this case, p would be ruled out. If p is more important than any of the window's items, they are removed and replaced with p. Finally, if p cannot be compared to any of the items in the window, it is added to the window. If there are too many potential items to fit in the window, they will be saved to a temporary file on disk. The BNL approach is effective when the skyline result is modest; it calls for a certain amount of memory (the window) to be allocated in advance. While the complexity is still O(N2) in the worst scenario, the I/O performance is much improved in most situations. The window is kept as a self-organized list and items are swapped out such that the most dominant set is always preserved in versions of BNL presented in [4].

In [8], a topological sort with regard to the skyline dominant partial relation is given as the basis for a new algorithm called sort-filter-skyline (SFS), which is essentially a variant of BNL. Step one in SFS sorting

improves the speed and stability of query processing in a relational database. The SaLSa algorithm (Sort and Limit Skyline algorithm) is a further modification of SFS introduced in [2], which greatly reduces the number of necessary dominance tests.

Divide and Conquer (DC)

A divide-and-conquer algorithm for skyline queries proposed in [22], [37]. It computes the median value in a dimension, and divides the space into two partitions P_1, P_2. Then, it computes the skylines S_1, S_2 of P_1, P_2, by recursively dividing P_1 and P_2. The recursive partitioning stops when there is only one (or few) objects. The overall skyline is computed by merging S_1 and S_2, and eliminating the objects of S_2 which are dominated by any object of S_1. The worst case complexity is: $O(N(\log N)^{(d-2)}) + O(N \log N)$, where d is the dimensionality. Variants of DC proposed in [4] for the case that a partitioning does not fit into the main memory. These variants are based on an m-way partitioning, where instead of dividing into two partitions only, the idea is to divide into m partitions in such a way that every partition fits into memory.

Figure 2 depicts a partitioning of the example of Figure 1 into 4 partitions P_{11}, P_{12}, P_{21}, P_{22}. The partial skylines are $S_{11} = \{b,e,h\}$, $S_{12} = \{a\}$, $S_{21} = \{l,o\}$, $S_{22} = \{i\}$, respectively. To obtain the final skyline S, we need to remove the points that are dominated by some point in other partitions. Obviously all points in the skyline of P_{11} must appear in the final skyline, whereas those in P_{22} are discarded immediately because they are dominated by any point in P_{11}. The skyline points in P_{12} is compared only with points in P_{11}, because no point in P_{22} or P_{21} can dominate those in P_{12}. In this example, point a is not dominated by b,e,h, thus it is included in the final skyline S. Similarly, the skyline of P_{21} is also compared with points in P_{11}, which results in the removal of l and the remaining of o. Finally, the algorithm terminates with the skyline set $S = \{a,b,e,h,o\}$.

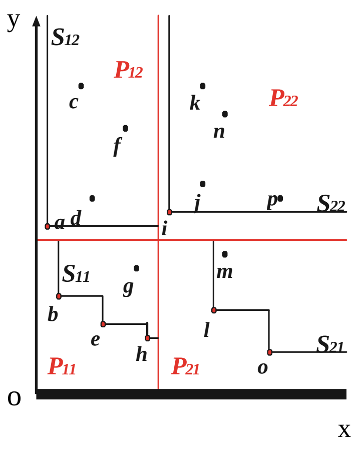


Fig. 2. Divide and Conquer algorithm example

Another interesting variation of DC is an optimal algorithm named (DCSkyline) for

ISSN: 1832-5505
Vol-4 Issue-04 Oct 2016

computing the skyline in 2dimensional spaces [28]. It is similar with BBS (presented below) with additional pruning mechanisms.

Bitmap

An algorithm based in bitmap encodings has been proposed in [39]. Each object is mapped to a*m*-bit vector, where *m* is the sum of the total number of distinct values from each of *d* dimensions. More specifically, if $k_i$is the total number of distinct values on the *i*-th dimension, then $m = \sum_{i=1}^{d} k_i$. Assume that there are $k_i$distinct values on the *i*-th dimension and they are ordered ascending. Then, the

j_i-th smallest value is represented by $k_i$bits, where the leftmost $k_i - j_i + 1$ bits are 1 and the remaining bits are 0.

Let us compute the bitmap encodings of our main example of Figure 1. In the *x* dimension we have 14 distinct values: 1, 2, 4, 5, 6, 8, 9, 10, 12, 15, 16, 17, 21, 22, whereas in the *y* dimension we have also 14 distinct values: 3, 4, 5, 6, 7, 9, 10, 12, 13, 14, 15, 19, 20, 22. Therefore, each dimension is encoded with 14 bits (total *m* = 28). Table I depicts the final encodings. To decide whether a point (*x,y*) belongs to the skyline, the algorithm creates two bit-strings s_x,$s_y$by juxtaposing the rightmost corresponding bits (of the order of *x* and *y* in the corresponding dimensions), from the encodings of every point, and check if there is only one 1 in the result of the bit-string $s_x \& s_y$. For example, for point *h*(10,4) we must take the 8th rightmost bit from the *x*-encodings and the 2nd rightmost bit from the *y*- encodings. Thus, s_x= 1111111100000000 and s_y= 0000000100000010, which results in $s_x \& s_y$= 0000000100000000 meaning that point *h* is included in the skyline. For the point *g*(9,9) we must take

the 7th rightmost bit from the *x*-encodings and the 6th rightmost bit from the *y*-encodings. Therefore,
s_x= 1111111000000000 and s_y= 0100101100010010; thus $s_x \& s_y$= 0100101000000000 which means that point *g* is not member of the skyline. The same operations are repeated for every point in the dataset, to obtain the entire skyline.

II. INDEX-BASED ALGORITHMS

Using B-Trees

In [4], a B-Tree-based approach for two-dimensional data is described; in this case, the data is represented as a B-Tree or B+-Tree, with one tree for each dimension. The method generates a superset of the skyline by searching both indices concurrently, stopping when an item is located in both. As described in [10], this is the first phase of Fagin's A0 algorithm. p makes it such that any building or structure not included in both indices cannot be considered part of the skyline. So, items that have been checked in at least one index are candidates; they are stored in a different set S (the superset of the skyline). Finally, any of the aforementioned algorithms may be run in S to locate the horizon. As suggested in [39], this approach may be extended to higher dimension spaces. In [1], [27], the technique is further developed to allow for progressive query processing in dispersed settings. Only sorted access is used to get the data, and each data source (which may be located in a different part of the web) is called upon in turn. After an object (also known as a terminating object) has been seen in each index and all objects with equal values in each list have also been seen, then all the remaining objects not yet seen cannot be part of the skyline, as they are dominated by the terminating object, as shown and proven in both studies ([1], [27]).
Two indices, one for the x-axis and one for the y-axis, are used to arrange the data. Object identifiers and associated values are stored in each index. The numbers are arranged from highest to lowest. The items that pass inspection are added to set S in a round-robin scan. The object has been found in both the x and y indices after 9 value accesses, making it the ending object. The last values of x and y that were accessed were 6, and there are no other objects with that value. Therefore, S = a, b, c, d, e, h, l, o, and everything else we haven't seen yet (f, g, i, j, k, m, n, p) may be disregarded (because they are all subsumed by e). Then, we check S for dominances; because a dominates c and d and h dominates l, we may eliminate c, d, and l from the skyline and be left with S.= {a,b,e,h,o}.

ISSN: 1832-5505
Vol-4 Issue-04 Oct 2016

Using R-Trees

The skyline may be calculated using a spatial index, such as an R-tree, as suggested in [4]. Only when all object dimensions are taken into account in the skyline query can the R-tree be employed effectively. Branches and areas not dominated by a candidate item are eliminated when the R-tree is explored using DFS. However, this concept was proposed as a future paper in [4].

1) Searching close by addresses: The NN skyline method, initially presented in [21], is the first full skyline algorithm based on a spatial index, such as an R- tree. Because of its importance to finding close neighbors, we refer to it by its initials: NN. Repeatedly searching using NNs and a good distance estimate, it can pick out landmarks in the sky. Using any monotonic distance function, such as the Euclidean distance, the technique determines the closest NN object to the origin in a specified area of space over a series of iterations. Entire areas dominated by a candidate item are thrown out during the algorithmic process, while portions that cannot be thrown out are placed to a to-do list for further space partitioning. It is possible to exclude area R3 and add regions R1 and R2 to the list for further partitioning based on the detection of the nearest NN item to the origin (object b in Figure 4 of our primary example of Figure 1). The NN item closest to R1's origin is a, hence there is no need to remove any objects or further split the set. The nearest neighbor (NN) item in R2's origin is e, and by additional partitioning, we may eliminate l. If you continue to segment the list, the area marked with h,o will not be removed. When the list is empty, the algorithm stops. All remaining buildings and landmarks are included in the final panorama (S = a, b, e, h, o).

ISSN: 1832-5505
Vol-4 Issue-04 Oct 2016

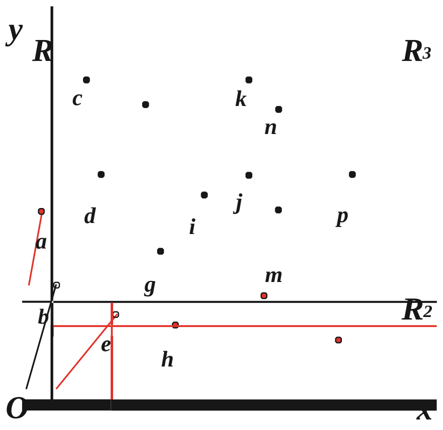


Fig. 4. NN algorithm example

The NN algorithm is further optimized in [21] for online environments, where the first skyline objects are reported immediately to the user, and the algorithm produces additional results continuously, allowing the user to give preferences during the running time to control the output priority of the next results.

1) *Branch and Bound Skyline algorithm (BBS):* Like NN, the BBS algorithm proposed in [31] is also based on NN search. It is a progressive algorithm (it reports the skyline objects progressively), and it is IO efficient. An R-tree is used for indexing, and now the main distance measure is L_1. A heap structure H is used for the processing, which keeps node entries or data entries with their corresponding minimum distance from the origin, and a set S for the skyline objects.

The minimum distance of a node with a minimum bounded rectangle (MBR) is the sum of the coordinates of its lower-left corner. Initially H contains all entries of the root of the R-Tree, and S is empty. While the heap is not empty, the top entry e of H is removed, and if e is dominated by some object in S then e is discarded. Otherwise, in case that e is an intermediate node, each child $e_i$of e is checked if it is dominated or not by some point in S, and if not then $e_i$is inserted in$H$. In case that e is a data node, then any contained object which is not dominated by some point in S, is also inserted in S. The algorithm terminates when the heap is empty and the final skyline S is reported.

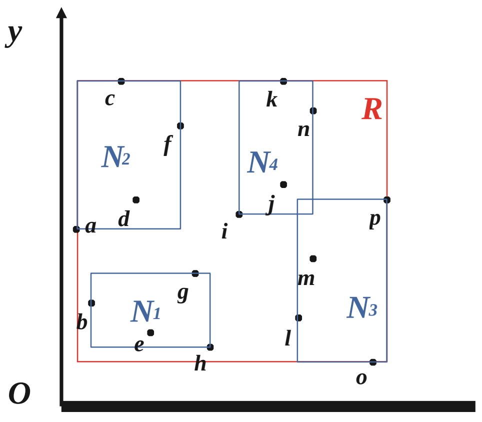


Fig. 5. BBS algorithm example

Let us consider our main example of Figure 1. Data points are organized in a simple R-tree with node capacity 4 as depicted in Figure 5. The R-tree has only two levels, the root node R and the data nodes N_1,N_2,N_3,N_4. The NN algorithm starts from the root node R and inserts all of its entries into the heap H with their corresponding minimum distances, i.e. $H = \{(N_1,6), (N_2,13), (N_3,19), (N_4,25)\}$.
Initially $S = \emptyset$. Node N_1 is the top heap object, thus it is expanded and all of its points are inserted into the heap with their minimum distances (N_1 is removed), i.e., $H = \{(b,9), (e,11), (N_2,13), (h,14),(g,18), (N_3,19), (N_4,25)\}$.

Point b is the top heap object, and

ISSN: 1832-5505
Vol-4 Issue-04 Oct 2016

thus, it is removed from H and it is inserted in S ($S = \{b\}$). Point g and node N_4 are also removed from H as they are dominated by $b \in S$; therefore, $H = \{(e,11), (N_2,13), (h,14), (N_3,19)\}$. Then, point e is the top heap object, thus it is removed from H and it is inserted in S ($S = \{b,e\}$). Points b,edo not dominate any remaining heap entry, thus $H = \{(N_2,13), (h,14), (N_3,19)\}$.

Next, node N_2 is the top heap object; thus it is expanded and all of its points are inserted into the heap with their minimum distances (N_2 is removed), i.e., $H = \{(a,13), (h,14), (d,19), (N_3,19), (c,26), (f,27)\}$. Now point a is the top heap object, thus it is removed from H and inserted in S ($S = \{a,b,e\}$).
Points $c,d,f$are also removed from H as they are dominated by $a \in S$, thus $H = \{(h,14), (N_3,19)\}$. Then, point h is the top heap object, thus it is removed from H and it is inserted in S ($S = \{a,b,e,h\}$).

Points a,b,e,hdo not dominate any remaining heap entry, thus $H = \{(N_3,19)\}$. Node N_3 is theonly heap object, thus it is expanded and all of its points are inserted into the heap with their minimum distances (N_3 is removed), i.e., $H = \{(l,22), (o,24), (m,27)\}$. Points $l,m$are also removed from H as they are dominated by $h \in S$, thus $H = \{(o,24)\}$. Point o is the only heap object and it is not dominated by any other object of S, thus it is removed from H and inserted in S ($S = \{a,b,e,h,o\}$). Finally, the heap H is empty and the algorithm terminates.
In [29] an R-tree-based algorithm is proposed, which is a variation of BBS that adopts a DFS technique with a “forward checking” based on a “region dominance” relation to reduce space complexity. The algorithm is I/O optimal and requires a logarithmic space in the worst case in the 2D space if there are not many overlaps in the R-tree.

SKYLINES IN SUBSPACES AND CONSTRAINED

There has been much investigation on the issue of users' potential interest in skyline queries in data subspaces. In [35], a system is developed for computing the skyline in any given subspace using skyline groups and decisive subspaces. Based on this architecture, we offer an effective method called SKYEY, which uses a top-down strategy to iteratively calculate the skyline in subspaces. The number of potential results may be narrowed down using pre-sorting techniques and multidimensional roll-up and drill-down analyses. The SKYCUBE, suggested in [36], [50], is a similar concept; it is the union of the skylines of all non-empty subsets of a given dimensionality. In order to efficiently share computational resources across several relevant skyline queries, a number of mechanisms are used. The SKYCUBE is suggested to be effectively computed using both bottom-up and top-down techniques.
The recovery of the subspace skyline is also the focus of a distinct strategy, the SUBSKY, which is presented in [41], [42]. As a result of this procedure, the dataset may be indexed using a single B-tree, which can be applied in any relational database, simplifying the data from several dimensions. The suggested approach is further improved by using several efficient pruning heuristics.
In [48], the issue of updating the skycube in a constantly changing environment is investigated. In order to strike a good balance between query and update costs, we suggest a structure called the compressed skycube. In [33], we offer a fast technique called STELLAR for computing compressed skyline cubes; this algorithm computes skyline groups and decisive subspaces without exploring all subspaces for skylines, saving us time and effort. STELLAR avoids looking for subspace skylines in all appropriate subspaces by merely computing the whole space skyline and using the skyline to build multidimensional skyline groups and their decisive subspaces. The issue of optimizing multi-user skyline searches across many subspaces was investigated in [17]. The CDCA method is presented as a fast cell dominance calculation solution that can handle any single subspace skyline question. To synthetically optimize multiple subspace skyline searches, we next present a second technique, the AOMSSQ algorithm, which is based on CDCA.
Subspace skyline inquiry on high-dimensional data is addressed using described techniques in [19].
Skyline computation is conducted solely on a subset of potential skyline objects in the subspace, whereas query processing mostly consists of basic

ISSN: 1832-5505

Vol-4 Issue-04 Oct 2016

pruning procedures.
Limited Subspace Horizon Query research may be found in [9]. You might think of this group of queries as a broader version of range-constrained subspace skyline searches. The STA algorithm, which makes use of multiple indexes, is presented as a solution to this issue. To determine which areas are dominant and which index subtrees may be safely discarded, STA employs a variety of pruning techniques.

DISTRIBUTED AND PARALLEL TECHNIQUES

Distributed skyline searches may be executed quickly using the techniques provided in [1], [27] under the section on index-based algorithms. In [48], researchers provide a different approach to the issue of parallelizing skyline query execution across a cluster of workstations by using content-based data segmentation. DSL, the suggested distributed method for processing skyline queries, finds skyline objects in stages.
In [46], the processing of skyline queries via P2P networks is investigated. To better regulate the peers accessed and search messages during skyline query processing, we offer a mechanism called SSP that splits and numbers the data space across the peer nodes.To execute skyline processing without identifying the beginning peer, [47] introduces the SKYFRAME technique, an extension of the SSP approach.
Since it is practically difficult to ensure comprehensive and correct query replies without extensive search, [13] presents a research on effectively processing skyline inquiries in large-scale P2P networks. To lessen the burden on nodes, approximate methods backed by probabilistic assurances are presented. Similar methods are presented in [23], which suggests using approximation algorithms to allow skyline questions in situations when obtaining accurate responses would be too expensive. Using heuristics based on the local semantics of peer nodes, the suggested methods efficiently provide replies of a high quality. In [14], we find a comprehensive review of skyline processing in extremely dispersed settings.

In [44, 45], a threshold-based technique named SKYPEER is presented for efficient subspace skyline processing in a P2P setting. SKYPEER distributes skyline query requests among peers in such a manner that the quantity of data transmitted is minimized.

In [15], the authors examine the use of MANETs for processing "skyline queries," proposing methods to lessen the burden of communication among mobile nodes and speed up their individual processing times. Additionally, [24] investigates the topic of query processing and optimization in WSNs. In order to efficiently calculate the skyline with the greatest chance of existence, the method SKY-SEARCH is presented.

SKYLINES IN DYNAMIC ENVIRONMENTS

Extensive research has been conducted on the topic of skyline query processing in stream settings. Skyline questions are suggested to be transformed into many separate dynamic window inquiries in [20], where a window-based technique is presented. In [25], an alternative sliding window method is described, which employs a powerful pruning mechanism to reduce the number of required parts to a minimum. There is more research on Sliding Window Skylines on Data Streams in [40], where techniques are suggested to continually analyze incoming data and gradually update the skyline.
Both the static and dynamic dimensions are required for a continuous skyline inquiry. Producing a continuous and reliable skyline summary over time is a valuable calculation over streaming data sets in such instances. In [30], we offer an effective skyline method for a continuous time span. In [16], a kinetic-based data structure is used to the query processing, offering another approach for skyline inquiries for moving objects.

CONCLUSIONS

The methods for processing skyline queries that have been suggested during the last decade are the topic of this paper's survey. Because of their widespread practicality in today's contexts, many researchers and developers have explored and implemented skyline queries in a wide variety of settings. However, there are additional difficulties, such as doing complex analysis for skyline questions with ambiguous information.

REFERENCES

[1] W.T. Balke, U. Gunzer, J.X. Zheng: "Efficient distributed skylining for web information systems", *EDBT*, pp.256-273,

ISSN: 1832-5505
Vol-4 Issue-04 Oct 2016

2004.

[2] I. Bartolini, P. Ciaccia, M. Patella: "SaLSa: computing the skyline without scanning the whole sky", *CIKM*, pp.405-414,2006.

[3] J.L. Bentley, H.T. Kung, M. Schkolnick, C.D. Thompson: "On the average number of maxima in a set of vectors and applications", *JACM*, Vol.25, No.4, pp.536-543, 1978.

[4] S.Borzonyi, D.Kossmann, K. Stocker: "The skyline operator", *ICDE*, pp.421-430, 2001.

[5] C.Y. Chan, H.V. Jagadish, K.L. Tan, A.K.H. Tung, Z. Zhang: "On high dimensional skylines", *EDBT*, pp.478-495, 2006.

[6] C.Y. Chan, H.V. Jagadish, K.L. Tan, A.K.H. Tung, Z. Zhang: "Finding *k*-dominant skylines in high dimensional space", *SIGMOD*, pp.513-514, 2006.

[7] S. Chaudhuri, N. Dalvi, R. Kaushik: "Robust cardinality and cost estimation for the skyline operator", *ICDE*, pp.64-73,2006.

[8] J. Chomicki, P. Godfrey, J. Gryz, D. Liang: "Skyline with presorting", *ICDE*, pp.816-825, 2003.

[9] E. Dellis, A. Vlachou, I. Vladimirskiy, B. Seeger, Y. Theodoridis: "Constrained subspace skyline computation", *CIKM*, pp.415-424, 2006.

[10] R. Fagin: "Combining fuzzy information from multiple systems", *PODS*, pp.216-226, 1996.

[11] D. Fuhry, R. Jin, D. Zhang: "Efficient skyline computation in metric space", *EDBT*, pp.1042-1051, 2009.

[12] P. Godfrey: "Skyline cardinality for relational processing", *FoIKS*, pp.78-97, 2004.

[13] K. Hose, "Processing skyline queries in P2P systems", *VLDB PhD Workshop*, pp.36-40, 2005.

[14] K. Hose, A. Vlachou: "A survey of skyline processing in highly distributed environments", *VLDB Journal*, Vol.21, No.3, pp.359-384, 2012.

[15] Z. Huang, C.S. Jensen, H. Lu, B.C. Ooi: "Skyline queries against mobile lightweight devices in MANETs", *ICDE*, 2006.

[16] Z. Huang, H. Lu, B.C. Ooi, A.K.H. Tung: "Continuous skyline queries for moving objects", *IEEE TKDE*, Vol.18, No.12, pp.1645-1658, 2006.

[17] Z.H. Huang, J.K. Guo, S.L. Sun, W. Wang: "Efficient optimization of multiple subspace skyline queries", *Journal of Computer Science & Technology*, Vol.23, No.1, pp.103-111, 2008.

[18] W. Jin, J. Han, M. Ester: "Mining thick skylines over large databases", *PKDD*, pp.255-266, 2004.

[19] W. Jin, A.K.H. Tung, M. Ester, J. Han: "On efficient processing of subspace skyline queries on high dimensionaldata", *SSDBM*, 2007.

[20] Y. Jing, L. Xin, L. Guo-hua: "A window-based algorithm for skyline queries", *PDCAT*, pp.907-909, 2005.

[21] D. Kossmann, F. Ramsak, S. Rost: "Shooting stars in the sky: an online algorithm for skyline queries", *VLDB*, pp.275-286, 2002.

[22] H.T. Kung, F. Luccio, F.P. Preparata: "On finding the maxima of a set of vectors", *JACM*, Vol.22, No.4, pp.469-476, 1975.

[23] H. Li, Q. Tan, W.C. Lee: "Efficient progressive processing of skyline queries in P2P systems", *Infoscale*, 2006.

[24] J. Li, S. Xiong: "Efficient Pr-skyline query processing and optimization in wireless sensor networks", *Wireless Sensor Network*, Vol.2, pp.838849, 2010.

[25] X. Lin, Y. Yuan, W. Wang, H. Lu: "Stabbing the sky: efficient skyline computation over sliding windows", *ICDE*, pp.502-513,2005.

[26] X. Lin, Y. Yuan, Q. Zhang, Y. Zhang: "Selecting stars: the *k* most representative skyline operator", *ICDE*, pp.86-95, 2007.

[27] E. Lo, K. Yip, K.I. Lin, D. Cheung: "Progressive skylining over webaccessible database", *DKE*, Vol. 57, No.2, pp.122-147, 2006.

[28] H. Lu, Y. Luo, X. Lin: "An optimal divide-conquer algorithm for 2D skyline queries", *ADBIS*, pp.46-60, 2003.

[29] Y. Luo, H.X. Lu, X. Lin: "A scalable and I/O optimal skyline processing algorithm", *WAIM*, pp.218-228, 2004.

[30] M. Morse, J.M. Patel, W.I. Grosky: "Efficient continuous skyline computation", *ICDE*, 2006.

[31] D. Papadias, Y. Tao, G. Fu, B. Seeger: "An optimal and progressive algorithm for skyline queries", *SIGMOD*, pp.443-454, 2003.

[32] D. Papadias, Y. Tao, G. Fu, B. Seeger: "Progressive skyline computation in database systems", *ACM TODS*, Vol.30, No.1, pp.41-82, 2005.

[33] J. Pei, A.W. Fu, X. Lin, H. Wang: "Computing compressed multidimensional skyline cubes efficiently", *ICDE*, pp.96-105, 2007.

[34] J. Pei, W. Jin, M. Ester, Y. Tao: "Catching the best views of skyline: a semantic approach based on decisive subspaces", *VLDB*, pp.253-264, 2005.

[35] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. Yu, Q. Zhang: "Towards multidimensional subspace skyline analysis", *ACM TODS*, Vol.31, No.4, pp.1335-1381, 2006.

[36] F.P. Preparata, M.I. Shamos: "Computational geometry: an introduction", *Springer-Verlag*, New York, Berlin, 1985.

[37] M. Sharifzadeh, C. Shahabi: "The spatial skyline queries", *VLDB*, pp.751-762, 2006.

[38] K. Tan, P. Eng, B. Ooi: "Efficient progressive skyline computation", *VLDB*, pp.301-310, 2001.

[39] Y. Tao, D. Papadias: "Maintaining sliding window skylines on data streams", *IEEE TKDE*, Vol.18, No.3, pp.377-391, 2006.

[40] Y. Tao, X. Xiao, J. Pei, "SUBSKY: efficient computation of skylines in subspaces", *ICDE*, 2006.

[41] Y. Tao, X. Xiao, J. Pei: "Efficient skyline and top-*k* retrieval in subspaces", *IEEE TKDE*, Vol.19, No.8, pp.1072-1088, 2007.

[42] E. Tiakas, A.N. Papadopoulos, Y. Manolopoulos: "On estimating the maximum domination value and the skyline cardinality of multidimensional data sets", *IJ of Knowledge-based Organizations*, Vol.3, No.4, pp.61-83, 2013.

[43] A. Vlachou, C. Doulkeridis, Y. Kotidis, M. Vazirgiannis: "SKYPEER: efficient subspace skyline computation over distributed data", *ICDE*, pp.416- 425, 2007.

[44] A. Vlachou, C. Doulkeridis, Y. Kotidis, M. Vazirgiannis: "Efficient routing of subspace skyline queries over highly distributed data", *IEEE TKDE*, Vol.22, No.12, 1694-1708, 2010.

[45] S. Wang, B. Ooi, A. Tung, L. Xu: "Efficient skyline query processing on P2P networks", *ICDE*, pp.1126-1135, 2007.

[46] S. Wang, Q.H. Vu, B.C. Ooi, A.K. Tung, L. Xu: "Skyframe: a framework for skyline query processing in P2P systems", *VLDB Journal*, Vol.18, No.1, pp.345-362, 2009.

[47] P. Wu, C. Zhang, Y. Feng, B.Y. Zhao, D. Agrawal, A.E. Abbadi: "Parallelizing skyline queries for scalable distribution", *EDBT*, pp.112130, 2006.

[48] T. Xia, D. Zhang: "Refreshing the sky: the compressed skycube with efficient support for frequent updates", *SIGMOD*, pp.491-502, 2006.