Skyline Queries: an Introduction

Eleftherios Tiakas
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
Email: tiakas@csd.auth.gr

Apostolos N. Papadopoulos
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
Email: papadopo@csd.auth.gr

Yannis Manolopoulos
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
Email: manolopo@csd.auth.gr

***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, divide-and-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.**

I. Introduction

Skyline queries received great attention in the database community during the past decades. The skyline computation became crucial to many multi-criteria decision making applications. A significant number of algorithms were proposed and studied extensively.

A. Definition

Given a dominance relationship in a dataset, a skyline query returns the objects that cannot be dominated by any other objects. In the case of a dataset consisting of multidimensional objects, an object dominates another object if it is as good in all dimensions, and better in at least one dimension. The definition of skyline queries in multidimensional datasets is identical with the known maximum vector problem [3], [22]. In these early works, skyline computation was an algorithmic problem in nature, and all data were assumed to reside in memory. However, nowadays we face big datasets which are stored in secondary memory. Having the data on disk(s), the proposed algorithms for skyline query processing are separated in two categories: index-based algorithms and non-index-based algorithms.

B. 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 $\{a, b, e, h, o\}$.

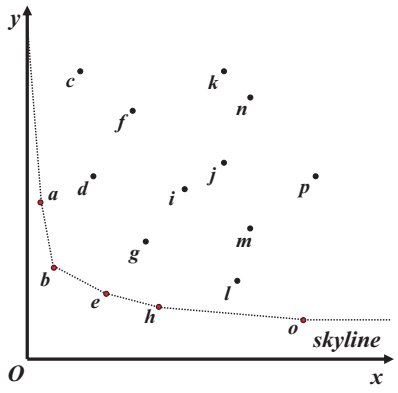

Fig. 1. Skyline Example

II. Non-Index-based algorithms

A. Block-Nested-Loop (BNL)

A naive algorithm to compute a skyline query is to compare every object with every other object of the dataset by using a nested-loop. However, the quadratic complexity $O(N^2)$ makes this algorithm very inefficient (N is the total number of objects into the dataset).

The Block-Nested-Loop algorithm [4] applies the same idea, but uses a window (memory block with limited space), which holds a limited number of data objects. Any candidate object p is checked if it is dominated by any other object within the window. If this happens, then p is eliminated. If p dominates any object of the window then these objects are eliminated, and p is inserted into the window. Finally, if p is incomparable with all objects in the window then it is inserted into the window. In case the window is full, a temporary disk file is used to hold the candidate objects. BNL algorithm works well when the skyline result is relatively small; it requires a predefined limited memory size (the window). The worst case complexity remains $O(N^2)$, however, with a much better I/O behavior in practice. In addition, variants of BNL have been proposed in [4], by maintaining the window as a self-organized list, and by replacing objects in the window to keep the most dominant set.

Another variation of BNL is the sort-filter-skyline algorithm (SFS) proposed in [8], which is based on a topological

sort with respect to the skyline dominance partial relation. The pre-sorting step of SFS makes the query processing efficient and well behaved in a relational setting. A further extension of SFS, proposed in [2], is the SaLSa algorithm (Sort and Limit Skyline algorithm), where the number of required domination checks is significantly reduced.

B. 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 $S = \{a, b, e, h, o\}$.

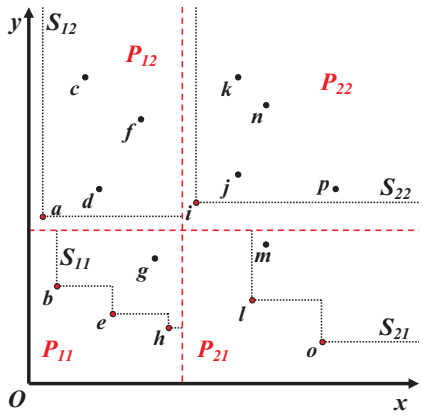

Fig. 2. Divide and Conquer algorithm example

Another interesting variation of DC is an optimal algorithm named (DCSkyline) for computing the skyline in 2-dimensional spaces [28]. It is similar with BBS (presented below) with additional pruning mechanisms.

C. 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.

TABLE I. BITMAP ENCODINGS EXAMPLE

Point	Bitmap representation
$a(1, 12)$	(11111111111111, 11111110000000)
$b(2, 7)$	(11111111111110, 11111111110000)
$c(4, 22)$	(11111111111100, 10000000000000)
$d(5, 14)$	(11111111111000, 11111000000000)
$e(6, 5)$	(11111111110000, 11111111111100)
$f(8, 19)$	(11111111100000, 11100000000000)
$g(9, 9)$	(11111111000000, 11111111100000)
$h(10, 4)$	(11111110000000, 11111111111110)
$i(12, 13)$	(11111100000000, 11111100000000)
$j(15, 15)$	(11111000000000, 11110000000000)
$k(15, 22)$	(11111000000000, 10000000000000)
$l(16, 6)$	(11110000000000, 11111111111000)
$m(17, 10)$	(11100000000000, 11111111000000)
$n(17, 20)$	(11100000000000, 11000000000000)
$o(21, 3)$	(11000000000000, 11111111111111)
$p(22, 14)$	(10000000000000, 11111000000000)

III. INDEX-BASED ALGORITHMS

A. Using B-Trees

An algorithm based in B-Trees for two-dimensional data has been proposed in [4], where the data have two ordered indices, e.g. a B-Tree or B$^+$-Tree, one for each dimension. The algorithm computes a superset of the skyline by scanning simultaneously through both indices and stops as soon as an

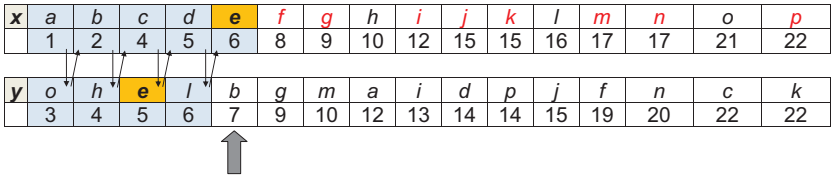

Fig. 3. Progressive Skyline Example

object p has been found in both indices. This is the first step of Fagin's A_0 algorithm in [10]. Any object which has not been inspected in both indices is definitely not part of the skyline, because it is dominated by p. Therefore, candidate objects are those which have been already inspected in at least one index; these objects are kept in a separate set S (the superset of the skyline). Finally, any of the previous algorithms can be executed in S to find the skyline. This algorithm can be generalized for more than two dimensions, as proposed in [39]. This algorithm is extended further in [1], [27] to support progressive query processing in distributed environments, in particular. The data are retrieved by sorted access only, and each data source (which can be in a different location in the web) is invoked in a round-robin fashion. In both studies ([1], [27]), the same important property of this algorithm (used for pruning) has been presented and proved: after an object has been seen in each index (which also is referred as a terminating object) 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.

Figure 3 depicts that moment for our main example of Figure 1. The data are organized in two indices, one for dimension x and one for dimension y. Each index keeps the object id and its corresponding value. The values are sorted in ascending order. During the round-robin scan the inspected objects are inserted in the set S. After 9 value accesses the object e has been detected in both x and y indices; thus, it is the terminating object. No more objects have equal values to 6 (which is the last accessed values on x and y). Therefore, $S = \{a, b, c, d, e, h, l, o\}$, and all other objects not yet seen (f, g, i, j, k, m, n, p) can be discarded (as they are dominated by e). Then, we check S for dominations: a dominates c, d and h dominates l; thus, c, d, l are removed from S, and the final skyline is $S = \{a, b, e, h, o\}$.

B. Using R-Trees

A spatial index, e.g. an R-tree, can be used to compute the skyline, as proposed in [4]. The R-tree involves all dimensions of the objects, thus it can be used only when all dimensions are considered into the skyline query. The R-tree is traversed in a DFS way, whereas branches and regions dominated by any candidate object are pruned. However, in [4] this idea was presented as a future work.

1) Nearest Neighbor search: The first complete skyline algorithm based on a spatial index, e.g. an R-tree, is the NN skyline algorithm, proposed in [21]. It is called NN due to its relevance to the nearest neighbor search. It identifies skyline objects by a repeated NN search using a suitable distance measure. The algorithm iteratively finds the NN object to the origin in a given region of space based on any monotonic distance function, e.g. the Euclidean distance. During the algorithmic process, entire regions dominated by a candidate object are discarded, and regions that cannot be discarded are added to a to-do list for further space partitioning. For example, when the NN object to the origin is detected (object b in Figure 4 of our main example of Figure 1), region R_3 can be discarded because all the contained objects are dominated by b, and regions R_1 and R_2 are added to the list for further partitioning. Continuing our example, the NN object to the origin of R_1 is a and is the only object, thus there are not any objects to discard or any further partitioning. The NN object to the origin of R_2 is e, and by further partitioning we can discard object l. The region of objects h, o is remaining to the list and by further partitioning the objects are not discarded. The list becomes empty and, thus, the algorithm terminates. The final skyline contains all objects that have not been discarded, i.e., $S = \{a, b, e, h, o\}$.

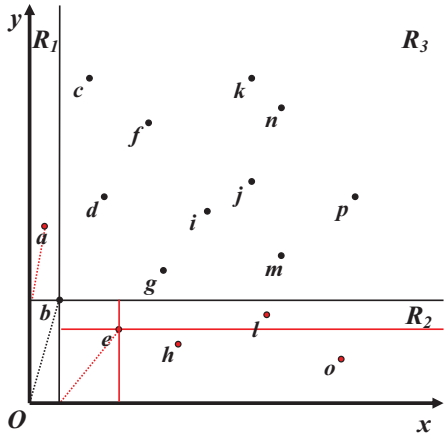

Fig. 4. NN algorithm example

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.

2) 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.

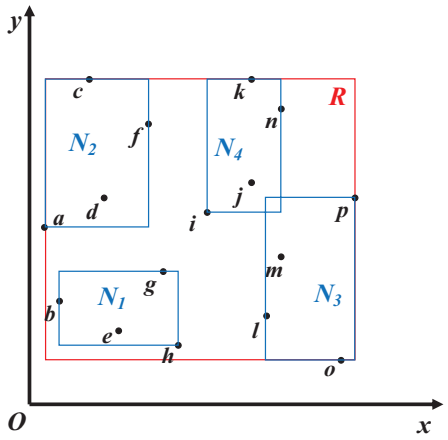

Fig. 5. BBS algorithm example

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 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, e do 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, h do not dominate any remaining heap entry, thus H = $\{(N_3,19)\}$. Node N_3 is the only 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.

IV. Skylines in Subspaces and Constrained

An important research has been made to answer the problem that users may be interested for skyline queries in subspaces of the data. In [35] a framework is proposed which uses skyline groups and decisive subspaces, to compute the skyline in any required subspace. Upon this framework an efficient algorithm is proposed, named SKYEY, which applies a top-down approach to recursively compute the skyline in subspaces. Pre-sorting strategies and multidimensional roll-up and drill-down analysis reduce the set of objects to be searched. A similar approach, the SKYCUBE, is proposed in [36], [50], which is the union of the skylines of all possible non-empty subsets of a given set of dimensions. Several computation sharing strategies are used, based on effectively identifying the computation dependencies among multiple related skyline queries. Bottom-Up and Top-Down algorithms are proposed to compute the SKYCUBE efficiently.

A different approach, the SUBSKY, is proposed in [41], [42], which also addresses the subspace skyline retrieval. This method transforms the multidimensional data into one-dimensional, and therefore permits indexing the dataset with a single B-tree, which can be implemented in any relational database. Moreover, the proposed method becomes more efficient by using several effective pruning heuristics.

In [49] the problem of updating the skycube in a dynamic environment with frequent updates is examined. To balance the query cost and update cost, a structure is proposed, the compressed skycube, which concisely represents the complete skycube. Efficient compressed skyline cube computation is presented in [33] by developing a fast algorithm STELLAR which computes skyline groups and decisive subspaces without searching all subspaces for skylines. STELLAR computes the full space skyline only and use the skyline to shape multi-dimensional skyline groups and their decisive subspaces, thus avoids searching for subspace skylines in all proper subspaces.

[17] examined the problem of optimizing multiple subspace skyline queries in multi-user environments. An efficient cell-dominance computation algorithm is proposed, the CDCA algorithm, for processing arbitrary single subspace skyline query. Then, another algorithm is proposed, the AOMSSQ algorithm, which is based on CDCA to synthetically optimize multiple subspace skyline queries.

In [19] methods are proposed for answering subspace skyline query on high dimensional data. The query processing involves mostly simple pruning operations, whereas skyline computation is performed only on a small subset of candidate skyline objects in the subspace.

Constrained Subspace Skyline Queries are studied in [9]. This class of queries can be thought of as a generalization of subspace skyline queries using range constraints. To address this problem the algorithm STA is proposed, which exploits multiple indexes. STA uses different pruning strategies to identify dominated regions and to discard irrelevant sub-trees of the indexes.

V. Distributed and Parallel Techniques

The algorithms of [1], [27], which have already been presented in the index-based algorithms section, can efficiently perform skyline queries in distributed environments. Another study that addresses the problem of parallelizing skyline query execution over a large number of machines by leveraging content-based data partitioning, is presented in [48]. The proposed distributed skyline query processing algorithm, named DSL, discovers the skyline objects progressively.

Skyline query processing on Peer-to-Peer (P2P) networks is studied in [46]. A method named SSP is proposed which partitions and numbers the data space among the peer nodes such that the target subspace (region) number can be derived with good accuracy to control the peers accessed and search messages during skyline query processing. A generalization of the SSP method is the SKYFRAME method presented in [47], which performs skyline processing without the need to determine the starting peer.

A study on efficiently processing skyline queries in large-scale P2P systems, where it is nearly impossible to guarantee complete and exact query answers without exhaustive search, is presented in [13]. Approximate algorithms with probabilistic guarantees are proposed to reduce the number of queried peers. Another similar approach is proposed in [23] where approximate algorithms are proposed to support skyline queries where exact answers are too costly to obtain. The proposed algorithms produce high quality answers using heuristics based on local semantics of peer nodes. A detailed survey also for skyline processing in highly distributed environments is presented in [14].

In [44], [45] a threshold based algorithm for efficient subspace skyline processing in a P2P environment is proposed, called SKYPEER, which forwards the skyline query requests among peers, in such a way that the amount of transferred data is significantly reduced.

Skyline query processing in MANETs is studied in [15] where techniques are proposed to reduce the costs of communication among mobile devices and reduce the execution time on each single mobile device. Query Processing and Optimization in Wireless Sensor Networks is studied also in [24]. The algorithm SKY-SEARCH is proposed which computes the skyline with the highest existence probability in a computational and energy-efficient way.

VI. Skylines in Dynamic Environments

Skyline query processing in stream environments studied also extensively. A window-based algorithm for skyline queries is proposed in [20], which transforms skyline queries into many different dynamic window queries. Another sliding window approach is proposed in [25], which applies an effective pruning technique to minimize the number of elements to be kept. Sliding Window Skylines on Data Streams are studied also in [40], where the proposed algorithms continuously monitor the incoming data and maintain the skyline incrementally.

A continuous skyline query involves not only static dimensions but also the dynamic one. In such cases, a useful computation over streaming data sets is to produce a continuous and valid skyline summary over time. An efficient continuous time-interval skyline algorithm is proposed in [30]. Another proposal for skyline queries for moving objects is presented in [16], where a kinetic-based data structure is applied into the query processing.

VII. Other Variations

In [5], a metric is proposed, the skyline frequency of an object, which is the number of subspaces in which it is a skyline object. Intuitively, an object with a high skyline frequency is more interesting as it can be dominated on fewer combinations of the dimensions. An efficient approach is proposed to compute top-k frequent skylines using that metric. A similar variation called k-dominant skyline is proposed in [6] which relaxes the notion of dominance. An object p is said to k-dominate another point q if there are $k \leq d$ dimensions in which p is better than or equal to q and is better in at least one of these k dimensions. An object that is not k-dominated by any other objects is in the k-dominant skyline. Three different algorithms are proposed to solve the k-dominant skyline.

Enumerating queries, proposed in [32], return for each skyline object p, the number of objects dominated by p. This information provides some measure of goodness for the skyline objects. An interesting variation of the problem is the top k-dominating query, which retrieves the k objects that dominate the largest number of other objects. A variation of the BBS algorithm is proposed to answer top-k dominating queries. Another similar variation proposed in [26] is the k Most Representative Skyline query, which selects k skyline objects so that the number of objects, which are dominated by at least one of these k skyline objects, is maximized. A dynamic programming exact algorithm, and a scalable index-based randomized algorithm is proposed to solve this problem.

The thick skylines are proposed in [18], which recommend not only skyline objects but also their nearby neighbors within a predefined threshold distance. Two efficient algorithms are proposed, named Sampling-and-Pruning and Indexing-and-Estimating, to find such thick skylines. Another interesting variation of the skyline queries is the Spatial Skyline Queries (SSQ) which proposed in [38]. They are skylines on spatial attributes that are derived from the spatial distances between the set of the data objects and a predefined set of query objects. Two efficient algorithms, named B^2S^2 and VS^2, are proposed for SSQ with static query objects and an algorithm, named VCS^2, for SSQ with moving query objects.

VIII. Cost Estimation of Skyline Queries

In high-dimensional spaces the number of skyline objects becomes very large. This happens because the probability that an object dominates another objects becomes very low. More specifically, there is an "eliminating dimension", studied in [43], which is the dimensionality beyond which all domination values become zero (i.e. and the skyline is equal to the full dataset). Therefore, in high-dimensional spaces, the estimation of the skyline cardinality is crucial for query optimization. Parametric estimations are proposed in [12], [43], while non-parametric estimation methods are proposed in [7] which use uniform random sampling.

IX. Conclusions

In this paper a focused survey is presented for skyline query processing algorithms that have been proposed during the last decade. Numerous methods have been studied and implemented in assorted environments for skyline queries; this depicts their importance in modern applications. However, there are also further challenges. For example, to conduct advanced analysis for skyline queries on uncertain data remains an open problem at large [34].

References

[1] W.T. Balke, U. Gunzer, J.X. Zheng: "Efficient distributed skylining for web information systems", *EDBT*, pp.256-273, 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 dimensional data", *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.838-849, 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 web-accessible 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, B. Jiang, X. Lin, Y. Yuan: "Probabilistic skylines on uncertain data", *VLDB*, pp.15-26, 2007.

[35] 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.

[36] 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.

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

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

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

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

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

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

[43] 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.

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

[45] 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.

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

[47] 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.

[48] P. Wu, C. Zhang, Y. Feng, B.Y. Zhao, D. Agrawal, A.E. Abbadi: "Parallelizing skyline queries for scalable distribution", *EDBT*, pp.112-130, 2006.

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

[50] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. Yu, Q. Zhang: "Efficient computation of the skyline cube", *VLDB*, pp.241-252, 2005.