Australian Journal of Basic and Applied Sciences, 9(10) Special 2015, Pages: 203-110

ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com

A Survey of Skyline Computing Algorithms

[1]A. Sairam and [2]Dr. C. Suresh Gnana Dhas

[1]*Research Scholar, Department of Computer Science and Engineering, Manonmaniam Sundaranar University, Tamilnadu*
[2]*Professor & HOD, Department of Computer Science and Engineering, Vivekananda Engineering College for Women, Tamilnadu*

ARTICLE INFO

Article history:
Received 28 January 2015
Accepted 25 February 2015
Available online 6 March 2015

Keywords:

ABSTRACT

Skyline queries are receiving interesting attention to the large database and data mining field and its main advantage is it is used for multi-criteria decision making and identifying interesting tuples with overall low formulation. Skyline queries mean it exposes a set of non-dominated points or better points from the given data points. This survey gives an overview about the existing algorithms of skyline computing technique.

To Cite This Article: A. Sairam and Dr. C. Suresh Gnana Dhas, A Survey of Skyline Computing Algorithms. ***Aust. J. Basic & Appl. Sci.,*** *9(10): 203-210, 2015*

INTRODUCTION

The skyline operator have more reasonable attention due to its multi-criteria decision making, user preference queries and data mining approach in most of the application. Given a hotel database *d1*, and it has a set of objects *q1,q2,…qn*. (Börzsönyi, 2001) an object *qi* is said to be in skyline of *d1*, if there is no other object *qj* in *d1* such that *qj* is better than *qi* in all dimensions. If there is exist such a *qj*, then we say that *qi* is dominated by *qj*, or *qj* dominates *q i*.

Most commonly using example in the literature is Holiday resort or Hotels for holiday destination, assume in figure 1.1.we have a set of hotels with its distance (x axis) and rate (y axis) from the beach. The most interesting hotels b, j and k for which there is no point that is better on both dimensions (Papadias, 2005).For instance, Hotel *a* in Figure 1.1 (Papadias, 2005) is better than hotels *b and e* is nearest to the beach and low-cost in price.

The big deal in the skyline queries is to finding skylines over high dimensional databases. Most of the existing skyline algorithms are work efficiently with the small database with low dimension. In this paper we try to expose the skyline algorithms for high dimensional database. Major leading problems such as Top-K queries, convex hull, nearest neighbor search. For the convex hull contains the subspace of skyline points that may be excellent only for linear functions, and the Top-K (or ranked) queries recover the best K objects that minimize a set the desire function. Nearest neighbor queries indicates a query point q and output the objects nearer to q, in increasing order of their distance.

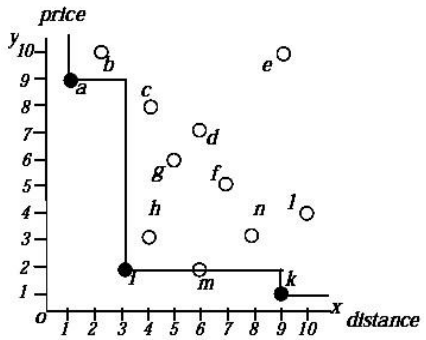

Fig. 1.1: Example dataset and skyline.

Analysis of skyline algorithms:

In this section we discuss about the classification of skyline algorithms and its advantages and disadvantages.

1) Sort based skyline algorithms
a. Bitmap algorithm
b. Index Algorithm
c. Sort first skyline algorithm(SFS)
d. LESS
e. SaLSa

Corresponding Author: A. Sairam, Research Scholar, Department of Computer Science and Engineering, Manonmaniam Sundaranar University, Tamilnadu.
E-mail: a.sairam@yahoo.co.in

2) Divide and- conquer algorithms
3) Hierarchical Index- based algorithms
a. R-Tree
A. Nearest Neighbor
B. Block Nested Loop (BNL)
b. B-Tree
A. ZB-Tree
B. Z-Sky
C. Zinc

1) Sort Based Skyline Algorithms:

Sort based algorithm (Dimitris Papadias, 2003) efficiently reduce the number of possibility objects and reduce the computing price. Which is used to sort the dataset objects based on analysis situs with comparable to decreasing and increasing marks of a monotone function, the calculation based on the skyline becomes forthright because objects below a certain threshold cannot be a part of the skyline. The key doctrine element affecting the performance of the sorting based algorithm is the choice of the sorting function and the threshold. However, this requires high-cost sorting process to prune the non-skyline object. Some more sort based algorithms are as follows.

Table 2.1: The bitmap approach.

id	Coordinate	bitmap representation
a	(1,9)	(1111111111, 1100000000)
b	(2,10)	(1111111110, 1000000000)
c	(4,8)	(1111111000, 1110000000)
d	(6,7)	(1111100000, 1111000000)
e	(9,10)	(1000000000, 1000000000)
f	(7,5)	(1111000000, 1111110000)
g	(5,6)	(1111110000, 1111100000)
h	(4,3)	(1111111000, 1111111100)
i	(3,2)	(1111111100, 1111111110)
k	(9,1)	(1000000000, 1111111111)
l	(10,4)	(1000000000, 1111111000)
m	(6,2)	(1111100000, 1111111110)
n	(8,3)	(1110000000, 1111111100)

a. Bitmap algorithm:

A Bitmap[12] approach encodes in bitmaps of all the information needed to decide whether a point is in the skyline. A data point $q = (q1, q2,..., qn)$ and where d is the number of dimensions, is mapped to a m-bit vector, where m is the total number of distinct values over all dimensions.

Let ki be the total number of distinct values on the i-th dimension (i.e., $m=\Sigma i=1\sim dki$). In Figure 1.1, for example, there are $k1=k2=10$ distinct values on the x-, y-dimensions and $m=20$. Estimate that qi is the ji-th smallest number on the i-th axis, then, it is represented by ki bits, where the leftmost $(ki - ji + 1)$ bits are 1, and the remaining ones 0.

Table 2.1 shows the bitmaps for points in Figure 1.1. Since point *a,* has the smallest value (1) on the x-axis, all bits of *a1* are 1. Similarly, since *a2* (=9) is the 9-th smallest on the y-axis, the first 10−9+1=2 bits of its representation are 1, while the remaining ones are 0.

Consider that we want to decide whether a points *c* with bitmap representation (1111111000, 1110000000), belongs to the skyline. The rightmost bits equal to 1, are the 4th and the 8th, on dimensions x and y, respectively. The algorithm creates two bit-strings, *cX* = 1110000110000 and *cY* = 0011011111111,by juxtaposing the corresponding bits (i.e., 4th and 8th) of every point.

These bit-strings (Shown in bold) contain 13 bits (one from each object, starting from *a* and ending with *n*). The 1's in the result of *cX* & *cY*=0010000110000, indicate the points that dominate *c,* i.e., *c, h* and *i*.

The main advantage of this approach is instantly return the first few skyline points based on their insertion order, and the major drawback is Expensive, because each point is inspected. Space consumption is high. This approach is not suitable for dynamic datasets where insertions may alter the rankings of attribute values.

b. Index algorithm:

The "*index*" approach (Papadias, 2005) coordinates a set of d-dimensional points into d lists. A set of d-dimensional points into d lists such that a point $q = (q1, q2, \ldots, qn)$ is assigned to the i-th list $(1 \le i \le d)$, if and only if its coordinate pi on the i-th axis is the minimum among all dimensions, or formally, $qi \le qj$ for all $j \ne i$. Initially, the algorithm loads the first batch of each list, and handles the one with the minimum *minC*.

A set of d-dimensional points into d lists such that a point $q = (q1, q2, \ldots, qd)$ is assigned to the i-th list $(1 \le i \le d)$, if and only if its coordinate qi on the i-th axis is the minimum among all dimensions, or formally, $qi \le qj$ for all $j \ne i$. Initially, the algorithm loads the first batch of each list, and handles the one with the minimum *minC*. In Table 2.2, the first batches $\{a\}$, $\{k\}$ have identical *minC*=1, in which case the algorithm handles the batch from list 1.

Processing a batch involves,
(i) Computing the skyline inside the batch,
(ii) Among the computed points, it adds the ones not dominated by any of the already-found skyline points into the skyline list.

Table 2.2: The index approach.

list 1		list 2	
a (1, 9)	*minC*=1	*k* (9, 1)	*minC*=1
b (2, 10)	*minC*=2	*i* (3, 2), *m* (6, 2)	*minC*=2
c (4, 8)	*minC*=4	*h* (4, 3), *n* (8, 3)	*minC*=3
g (5, 6)	*minC*=5	*l* (10, 4)	*minC*=4
d (6, 7)	*minC*=6	*f* (7, 5)	*minC*=5
e (9, 10)	*minC*=9		

Points in each list are sorted in ascending order of their minimum coordinate (*minC*, for short) and indexed by a B-tree.

Since batch {*a*} contains a single point and no skyline point is found so far, *a* is added to the skyline list. The next batch {*b*} in list 1 has *minC*=2, thus, the algorithm handles batch {*k*} from list 2. Since *k* is not dominated by *a*, it is inserted in the skyline. Similarly the next batch handled is {*b*} from list 1, where *b* is dominated by point *a* (already in the skyline).

The algorithm proceeds with batch {*i*,*m*}, computes the skyline inside the batch that contains a single point *i* (i.e., *I* dominates *m*), and adds *i* to the skyline. At this step the algorithm does not need to proceed further, because both coordinates of *i* are smaller than or equal to the *minC* (i.e., 4, 3) of the next batches (i.e., {*c*}, {*h*,*n*}) of lists 1 and 2.All the remaining points (in both lists) are dominated by i and the algorithm terminates with *{a, i, k}*.

The merits of the index approach is quickly return skyline points at the top of the lists, and the demerits are with the *bitmap* approach, the order that the skyline points are returned is fixed, not supporting user-defined options and the lists figure out for *d* dimensions cannot be used to fetch the skyline on any subset of the dimensions.

c. Sort first algorithm:

The SFS[1] variation of BNL (Börzsönyi, 2001) lightens these problems by first sorting the entire dataset according to a (monotone) preference function.

Possibility points are added into the list in ascending order of their scores, because points with lower scores are likely to dominate a large number of points, thus rendering the pruning more effective.

Based on figure 1.1 dataset points, SFS show the dynamic behavior because the pre-sorting ensures that a point *p* dominating another *q'* must be visited before *q'*, hence we can immediately output the points inserted to the list as skyline points.

First SFS has to scan the entire data file to return a complete skyline, because even a skyline point may have very large score and thus appear at the end of the sorted list. Main merits are SFS can efficiently reduce the number of possibility objects and thus reduce the computing cost, and the key affecting factor is the choice of the sorting function and the threshold. It requires high-cost sorting process to prune non-skyline objects.

d. LESS:

LESS *(Linear Elimination Sort for Skylin*e) (Godfrey, 2007) is the combination of Sort First Skyline(SFS) and Block Nested Loop(BNL) (Börzsönyi,2001). Thus LESS sorts the records first, then filters the records via a skyline-filter (SF) window, as does SFS.

LESS makes two major changes:

1. It uses an elimination-filter (EF) window in pass zero of the external sort routine to eliminate records quickly, and
2. It combines the final pass of the external sort with the first skyline-filter (SF) pass. The external sort routine used to sort the records is integrated into LESS.

Let b be the number of buffer pool frames allocated to LESS. Pass zero of the standard external sort routine reads in b pages of the data, sorts the records across those b pages (say, using quick sort), and writes the b sorted pages out as a b-length sorted run.

All subsequent passes of external sort are merge passes. During a merge pass, external sort does a number of (b − 1)-way merges, consuming all the runs created by the previous pass. For each merge, (up to) b − 1 of the runs created by the previous pass are read in one page at a time, and written out as a single sorted run. LESS additionally eliminates records during pass zero of its external-sort phase. It does this by maintaining a small elimination-filter window. Copies of the records with the best entropy scores seen so far are kept in the EF window. The EF window acts similarly to the elimination window used by BNL. In effect, LESS has all benefits of SFS's with no disadvantages. LESS should normally perform better than SFS. Some buffer-pool space is allocated to the EF window in pass zero for LESS which is not for SFS.

Therefore, the initial runs produced by LESS's pass zero are smaller than SFS's, this may sometimes force that LESS will require an additional pass to complete the sort. Of course LESS saves a pass since it combines the last sort pass with the first skyline pass. LESS also has advantages of BNL's advantages and effectively none of its disadvantages.

BNL has the top of tracking when window records can be promoted as known maximals. LESS does not need this. Maximals are classified more efficiently once the input is effectively sorted. Thus LESS has the same advantages as does SFS in comparison to BNL.

e. SaLSa:

SaLSa (*Sort and Limit Skyline algorithm*) (Ilaria Bartolini), is differs from other collective algorithms in that it consistently *limits* the number of points on which dominance tests need to be executed. If the input relation *r* is sorted according to a suitably chosen monotone function, then it is possible to determine the skyline of *r* without applying the skyline filter to all the points. In general, this might highly reduce the number of tuples to be read and, depending on the specific instance and sorting function; it might reduce the number of dominance tests as well.

Since SaLSa shares with SFS the idea of presorting the input relation, it also keeps all the SFS strengths: simplified management of the window,

incremental delivery of results, and optimal number of passes of the filter phase. We illustrate how SaLSa works when a single pass is sufficient to complete the evaluation. Extension to the case where skyline size exceeds the available main memory is managed as in SFS, and not reported here for brevity.

SaLSa starts by initializing to *r* the set *u* of *unread tuples*. It also makes use of a *stop point*, *pstop*, which is used to earlier terminate reading tuples. Step 2 sorts *u* according to decreasing values of a monotone function *F*. This is actually done by issuing the following standard, i.e., non-skyline,

1: $\mathcal{S} \leftarrow \emptyset$, $\mathcal{U} \leftarrow r$, $stop \leftarrow false$, $p_{stop} \leftarrow$ undefined
2: sort $\mathcal{U}$ according to $\mathcal{F}$
3: **while not** $stop \wedge \mathcal{U} \neq \emptyset$ **do**
4: $p \leftarrow$ get next point from $\mathcal{U}$, $\mathcal{U} \leftarrow \mathcal{U} \setminus \{p\}$
5: **if** $\mathcal{S} \not\succ p$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \{p\}$, update p_{stop}
6: **if** $p_{stop} \succ \mathcal{U}$ **then** $stop \leftarrow true$
7: **return** $\mathcal{S}$

Algorithm 1.1: *SaLSa*

Each time a new point *p* is read from *u*, *p* is compared against the current skyline *S*. If none of the points in *S* dominates *p* (i.e., *S* __ *p*), *p* is inserted into *S*. This might possibly trigger the update of the stop point (step 5). At step 6 SaLSa checks if it has gained sufficient evidence to conclude that no further point in *u* can be part of the skyline, i.e., all points in *u* are dominated by *pstop* (*pstop* _ *u*).If this is so, the algorithm terminates.

2) Divide and- conquer algorithms:

Divide and- conquer (Börzsönyi, 2001), Divide the large dataset into several smaller partitions. Continue till each smaller partition of the dataset fits in the main memory figure1.2 (a). Compute partial skylines in each partition. Compute global skylines by merging them. Calculate the median *mp* (or some approximate median) of the input for some dimension *dp*. Divide the input into two partitions.

1. *P*1 contains all tuples whose value of attribute *dp* is better than *mp*.
2. *P*2 contains all other tuples.

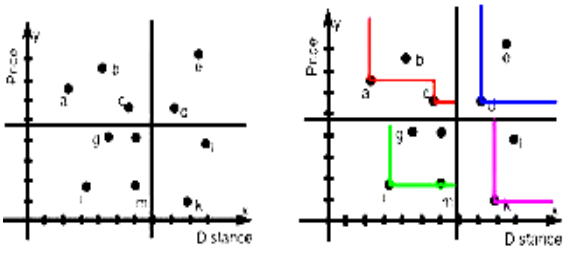

Fig. 1.2: a) Partition of the dataset b) Partial Skyline for each partition.

The data space is divided into 4 partitions *s1*, *s2*, *s3*, *s4* as show in the figure 1.2 (b), with partial skylines {*a*,*c*,*g*}, {*d*}, {*i*}, {*m*,*k*}, respectively. Compute the Skylines *S*1 of *P*1 and *S*2 of *P*2. This is done by periodically applying the whole algorithm to *P*1 and *P*2 i.e., *P*1 and *P*2 are again partitioned.

The recursive partitioning stops if a partition contains only one (or very few) tuples. In this case, computing the Skyline is trivial. Compute the overall skyline result by merging of S1 and S2.By eliminating those tuples of S2 which are dominated by tuple in S1.

To getting the final skyline, we need to remove the points that are dominated by some point in other partitions. Points *c*, *g* are removed because they are dominated by *i*. Finally, the algorithm terminates with the remaining points {*a*,*i*,*k*}.the attractive feature of D&C is very fast compare to other algorithm and this also has some demerits they are very sensitive to main memory size and the dataset characteristics.

3) Hierarchical Index- based algorithms:

The Hierarchical index-based algorithm has two classifications: one is most popular index structure, as B-Tree and the other is R-Tree.

1) By the use of popular index structures, such as B-tree and R-tree (Theodoridis, 2000) presents a genuine way to totally minimize the size of a skyline possibility set.

2) The dataset points that are nearer the base point have higher chance of being the skyline. Thus, the computation of the skyline can be implemented through *k* nearest neighbor search (*k*NN) (Kossmann, 2002).

The advantage of hierarchical index-based approach is its forward looking behavior that can quickly return the initial results without having to scan the entire dataset. This method also has some other built-in setbacks, by restrain their usefulness to only some cases. This approach adopts sophisticated techniques such as the smart partitions of B-tree index, the clever use of R-tree and the intelligent use of multicore architectures (Lee, 2007) to accelerate the skyline computation by parallelizing the most CPU-intensive parts, the dominance tests, as well as the fundamental limitation of hierarchical index based solutions.

a. R-Tree:

R-trees (Mehdi Sharifzadeh, 2010) for indexing multi-dimensional data marked a new generation in developing innovative R-tree-based algorithms for various forms of Nearest Neighbor (NN) queries. These algorithms utilize the simple rectangular grouping principle used by R-tree that represents close data points with their Minimum Bounding Rectangle (MBR). R-tree is a height-balanced tree similar to a B-tree with index records in its leaf nodes containing pointers to data objects Nodes correspond to disk pages If the index is disk-resident, and the structure is designed so that a spatial search requires

visiting only a small number of nodes The index is completely dynamic; inserts and deletes can be intermixed with searches and no periodic reorganization is required.

i. Nearest Neighbor (NN):

NN uses the results of nearest neighbor search to partition the data universe recursively. Consider the application of the algorithm to the dataset of Figure 1.1, which is indexed by an R-tree (Theodoridis, 2000). Find nearest neighbor point with minimum distance *(mindist)* from the beginning of the axis (point *o*) to create skyline.

Prune all the points in the dominance region of this point Divide the space by the nearest neighbor point. Partitions are inserted into a to-do list. Compute recursively until empty space. Consider the application of the algorithm to the dataset of Figure 1.1, which is indexed by an R-tree.for more instance, assumes that a spatial index structure on the data points is available for use.

Identifies skyline points by repeated application of a nearest neighbor search technique on the data points, using a suitably defined L_1 distance norm. The nearest neighbor in the data-point is as it is closest to the origin when an L_1 distance measure is assumed as like in the figure 1.3.

- Divides the space into 2^d non-disjoint region, which now must now be recursively searched for more skyline points.
- However, region 4 and 2 need not be searched. The rest of the2^d-2 regions need to be searched
- No closer point than *i* in *2* (by virtue that *i* is the nearest neighbor to the origin). Any data-point in the space *2* is dominated by *i*.

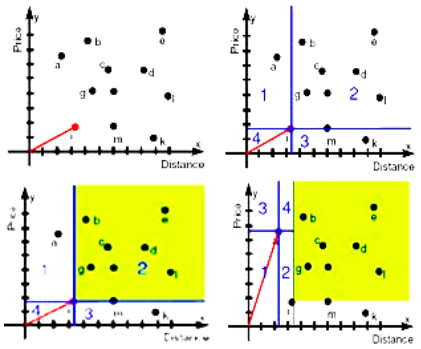

Fig. 1.3: NN algorithm Skyline.

- Recursively apply the search on region-1. The nearest neighbor in region- 1would be *a*, explode region to form additional regions .region- 4 is added to the pruned region and need not be searched the number of unexplored regions grow rapidly *O (dataset)*. The non-disjoint condition is relaxed for high-dimensional datasets.

The partitions resulting after the discovery of a skyline point are inserted in a *to-do* list. While the *to-do* list is not empty, NN removes one of the partitions from the list and recursively repeats the same process. The merits are fast running time to finding the first result and Progressiveness. And the demerits are Redundant I/O computation, Gets worse as dimensionality increases Explosive to-do list size

ii. Block Nested Loop (BNL):

Block Nested Loop (BNL (Godfrey, 2007) a honorable method to calculate the skyline is to compare each point *q* with every other point; if *q* is not dominated, then it is a part of the skyline. BNL builds on this concept by scanning the data file and keeping a list of candidate skyline points in main memory. The first data point is inserted into the list. For each subsequent point q, there are three cases:

(i) If *p* is dominated by any point in the list, it is discarded as it is not part of the skyline.

(ii) If *p* dominates any point in the list, it is inserted into the list, and all points in the list dominated by *p* are dropped.

(iii) If *p* is neither dominated, nor dominates, any point in the list, it is inserted into the list as it may be part of the Skyline.

The list is self-organizing because every point found dominating other points is moved to the top. This reduces the number of comparisons as points that dominate multiple other points are likely to be checked first. A problem of BNL (Börzsönyi, 2001) is that the list may become larger than the main memory. When this happens, all points falling in third case (cases (i) and (ii) do not increase the list size), are added to a temporary file.

This fact necessitates multiple passes of BNL. In particular, after the algorithm finishes scanning the data file, only points that were inserted in the list before the creation of the temporary file are guaranteed to be in the skyline and are output. The remaining points must be compared against the ones in the temporary file. Thus, BNL has to be executed again, this time using the temporary (instead of the data) file as input. The advantage of BNL is its wide applicability, since it can be used for any dimensionality without indexing or sorting the data file. Its main problems are the reliance on main memory and its inadequacy for on-line processing because it has to read the entire data file before it returns the first skyline point.

a. B-Tree

The Calculation of the skyline can also be facilitated by using *Hierarchical* index structures. In (Börzsönyi, 2001), a method based on B-tree was described. Assuming that each record has d dimensions and there is an index for every dimension, the skyline can be calculated as follows.

- Scan the entire indexes simultaneously to and first match, i.e., the first record to be seen by all the indexes during the scan. (Table 1.1)

- The first match is absolute part of the skyline and can be returned immediately, providing a fast initial response.
- Scan the rest of the index entries of the first dimension's index. If the record has not been seen before (i.e., the index entries of this record in the other indexes have not been examined prior to the first match), it is definitely not in the skyline and can thus be eliminated. If any of the other indexes contain an index entry to this record prior to the first match, then the record may or may not be in the skyline. To determine whether it is in the skyline, an existing skyline computation algorithm can be applied.

Table 1.1: B-Tree scanning the data list.

hotel	price	hotel	distance
h_1	$25	h_{17}	0.1 miles
h_3	$27	h_2	0.2 miles
h_{25}	$30	h_{35}	0.3 miles
h_2	$35	h_{25}	0.3 miles
h_{35}	$40	h_1	0.7 miles
h_{17}	$70	h_3	1.0 miles
...		...	

A critical factor that will affect the performance of this algorithm is how fast the first match can be found. If a match is found late (which is likely to be the case for large number of dimensions), it will result in a high initial response time. Nevertheless, we can expect this algorithm to perform well in general, when the skyline is small and the first match can be found quickly.

i. ZB-Tree:

ZB-tree method (Lee, 2007), This method is designed for data where all the attributes have TO domains. It first maps each multidimensional data point to a one-dimensional Z-address according to Z-order curve by interleaving the bit string representations of the attribute values of that point.

For instance, given a 2D data point (0,5), its bitstring representation is (000,101) and its Z-address is (010001). Figure 1.4(a) depicts an example of Z-order curve on the set of 2D data points shown in Figure. 1.1. By ordering data points in non-descending order of their Z-addresses, ZB-tree has two very useful properties. The monotonic ordering property states that a data point p cannot be dominated by any point that succeeds p in the Z-order.

Due to the monotonic ordering property of ZB-tree, each visited data point in a leaf node that is not dominated by any skyline point in SL is guaranteed to be a skyline point and is inserted into SL and output to the users immediately. The clustering property of ZB-tree enables many index sub tree traversals to be efficiently pruned leading to its superior performance over BBS (Bin Liu, 2011).

ii. Z-SKY:

Z-SKY (Lee, 2010)for skyline queries and alternative, and skyline result are carrying based on Z-order curves, Z-SKY is especially for large datasets with high data dimensionality, and the extensibility of a processing framework to support skyline query variants. The powerful view of our Z-SKY framework is illustrate in Figure. 1.4. It consists of four main components, namely,

1. *A data source (SRC),*
2. *A set of skyline possibilities (SL),*
3. *An algorithm library, and*
4. *Dominance tests.*

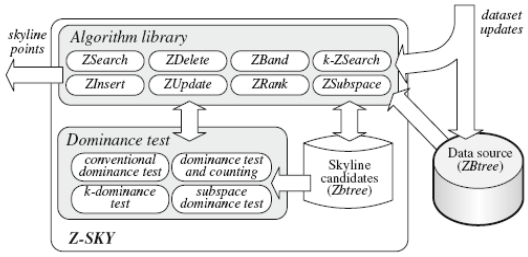

Fig. 1.4: The Z-SKY skyline query processing framework.

Specifically, *SRC* is a set of source data points indexed by a ZBtree (Ken, 2007). ZBtree indexes data points based on their values on a Z-order curve. Second, *SL* maintains skyline possibilities(candidates) indexed by another ZBtree. Third, the dominance tests provide different types of dominance relationship tests, such as (traditional) dominance tests, dominance and enumerate that complete the number of dominating points for a data point, *k*-dominance tests and subspace dominance tests that support *k*-dominant skyline queries and subspace skyline queries, respectively.

The figure 1.4 , the algorithm library maintains a suite of algorithms, namely, (1) *ZSearch*, which processes skyline queries, (2) *ZInsert*, *ZDelete* and *ZUpdate*, which incrementally update skyline query results, (3) *ZBand*,which evaluates skyband queries, (4) *ZRank*, which returns skyline points that dominate the most data points, (5) *k-ZSearch*, which answers *k*-dominant skyline queries and (6) *ZSubspace*, which performs skyline searches on specified subsets of dimensions. Upon receiving a skyline query (or its variant), a corresponding algorithm is then invoked to access and examine data points from *SRC* with corresponding dominance tests. The candidates are kept in *SL*. After evaluations, skyline points are delivered. Likewise, to maintain a skyline result in presence of dataset updates, a skyline result update algorithm is triggered that determines the change of a result preserved in *SL* and accesses required data points from *SRC* if needed. It is noteworthy that the underlying operations and data structures follow a coherent idea and concept developed based on Z-order curves.

iii. ZINC:

ZINC (for Z-order indexing with Nested Code) (Bin Liu Chee, 2010) that supports efficient skyline computation for data with both Totally Ordered (TO) as well as Partially Ordered (PO) attribute domains. ZINC is basically a ZB-tree that uses a novel encoding scheme to map PO domain values into bitstrings. Once the PO domain values have been mapped into bitstrings, the mapped bitstrings of all the attributes (whether TO or PO domains) of the records will be used to construct a ZB-tree index. Thus, the index construction and search algorithms for ZINC is equivalent to those of ZB-tree except that ZINC uses a different method for dominance comparisons between PO domain values. ZINC is able to encode partial orders of varying complexity in a brief manner while maintaining a good clustering of the PO domain values.

Conclusion:

This paper made a brief survey on skyline computing an overview of some algorithms and techniques. Recently skyline algorithm receives an attractive attention in data mining and big data field. Skyline queries retrieve the non-dominated points from a large database system based on the user preference so it can be used in partial based applications. However, all existing database algorithms for skyline Computations have several deficiencies, which severely limit their applicability. BNL, SFS and D&C are not progressive. Bitmap is applicable only for datasets with small attribute domains and cannot efficiently handle updates. Index cannot be used for skyline queries on a subset of the dimensions. SFS, like all above algorithms, does not support user-defined preferences. LESS should consistently perform better than SFS. SaLSa algorithm is the ability of computing the result without having to apply dominance tests to all the objects in the input relation.

REFERENCES

Angel Bency, C., S. Deepa Kanmani, 2014. ”A SURVEY OF SKYLINE PROCESSING IN VARIOUS ENVIRONMENT”, IJCSE, ISSN: 0976-5166, 5(1).

Apeksha Aggarwal, Harsh Kumar Verma , 2014. ”Skyline Computation Algorithms - A Study”, Dr B.R. Ambedkar NIT, Jalandhar, India, 2014, IJARCSSE.

Bin Liu Chee, Yong Chan, 2010. ”ZINC: Efficient Indexing for Skyline Computation”P roceedings of the VLDB Endowment, Volume 4 Issue 3.

Bin Liu, 2011. , “ZINC Efficient Indexing for Skyline Computation”, Proceedings of the VLDB Endowment, Vol. 4, No. 3 August 29th - September 3rd 2011.

Börzsönyi, S., D. Kossmann and K. Stocker, 2001. “The skyline operator”. Proceedings of ICDE, pp: 421-430.

Dimitris Papadias, 2003. “An Optimal and Progressive Algorithm for Skyline Queries”. ACM SIGMOD'2003, June 9-12, San Diego, California, USA.

George Trimponias, Ilaria Bartolini, Member, 2013. IEEE, Dimitris Papadias, and Yin Yang, “Skyline Processing on Distributed Vertical Decompositions”. IEEE Transactions On Knowledge And Data Engineering, Vol. 25, No. 4.

Godfrey, P., R. Shipley and J. Gryz, 2007. “Algorithms and Analyses for Maximal Vector Computation,” Int’l J. Very Large Data Bases, 16(1): 5-28.

Godfrey, P., R. Shipley and J. Gryz, 2005. Maximal vector computation in large data sets. In VLDB.

Godfrey, Shipley, Gryz, 2006. “Algorithms and Analyses for Maximal Vector Computation”. VLDB Journal, pp:1-22.

Ilaria Bartolini, Paolo Ciaccia , Marco Patella,”SaLSa: Computing the Skyline without Scanning the Whole Sky”.

ILARIA BARTOLINI, PAOLO CIACCIA and MARCO PATELLA, 2008. “Efficient Sort-Based Skyline Evaluation”. ACM Transactions on Database Systems, Vol. 33, No. 4, Article 31, Publication date: November.

Jan Chomicki, Parke Godfrey, Jarek Gryz and Dongming Liang, 2005. “Skyline with PresortingTechnical Report, Computer Science, York University, Toronto, ON, Canada, Oct.

Ken, C.K. Lee, 2007. “Approaching Skyline in Z-order” VLDB, Sept 07.

Ken, C.K. Lee, Baihua Zheng, Li. Huajing, Wang-Chien Lee, 2006. ”Approaching the Skyline in Z Order”.

Kossmann, D., F. Ramsak, S. Rost, 2002. “Shooting Stars in the Sky: an Online Algorithm for Skyline Queries”. VLDB.

Kossmann, D., F. Ramsak, S. Rost, 2002. Shooting Stars in the Sky: an Online Algorithm for Skyline Queries. VLDB.

Lee, C.K. Ken Lee, Wang-chien, ZHENG, Li. Baihua, Huajing; and Tian, Yuan, Z-SKY, 2010. An Efficient Skyline Query Processing Framework Based on Z-Order. (2010). VLDB Journal, 19(3): 333-362.

Lee, K., B. Zhang, H. Li, and W.C. Lee, 2007. “Approaching the Skyline in Z Order,” Proc. 33rd Int’l Conf. Very Large Data Bases (VLDB).

Mehdi Sharifzadeh, Cyrus Shahabi, 2010. ”VoRTree:Rtrees with Voronoi Diagrams for Efficient Processing of Spatial Nearest Neighbor Queries”, Proceedings of the VLDB Endowment, 3(1).

Ms. Kulkarni1, R.D. and Prof. Dr. B.F. Momin,

2012. " Future Research Directions in Skyline Computation". International Journal of Computer Engineering Science (IJCES), Volume 2 Issue .

Papadias, D., Y. Tao, G. Fu and B. Seeger, 2005. "Progressive Skyline Computation in Database Systems," ACM Trans. Database Systems, 30(1): 41-82.

Su Min Jang and Choon Seo Park, 2010. "Skyline Minimum Vector". 12th International Asia-Pacific Web Conference.

Tao, Y., X. Xiao and J. Pei, 2006. "SUBSKY: Efficient Computation of Skylines in Subspaces," Proc. 22nd Int'l Conf. Data Eng. (ICDE).

Theodoridis, Y., E. Stefanakis, Sellis, 2000. "T. Efficient Cost Models for Spatial Queries Using R-trees". TKDE, 12(1): 19-32.

Vlachou, C. Doulkeridis and Y. Kotidis, 2008. "Angle-Based Space Partitioning for Efficient Parallel Skyline Computation," Proc. ACM SIGMOD Int'l Conf. Management of Data.

Vlachou, C. Doulkeridis, Y. Kotidis and M. Vazirgiannis, 2007. "SKYPEER: Efficient Subspace Skyline Computation over Distributed Data," Proc. Int'l Conf. Data Eng. (ICDE).

Wang, S., Q.H. Vu, B.C. Ooi, A.K.H. Tung and L. Xu, 2009. "Skyframe: A Framework for Skyline Query Processing in Peer-to-Peer Systems," Int'l J. Conf. Very Large Data Bases, 18(1): 345-362.

Yunjun Gao, 2014. . "On efficient reverse skyline query processing" ELSEVIER, Expert Systems with Applications, 41: 3237–3249.