International Journal of Advanced Science and Technology
Vol. 28, No. 8s, (2019), pp. 812-818

# The Skyline Operator Algorithms: A Review

Siti Nurulain Mohd Rum, Noor Afiza Mohd Ariffin, Aziah Asmawi
Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia Selangor, Malaysia

Siti Fatimah Mohd Rum
Faculty of Computer and Mathematical Sciences,Universiti Teknologi MARA Melaka Kampus Jasin,Merlimau, Melaka, Malaysia

Zarina Dzolkhifli
Faculty of Computer System and Software Engineering,Universiti Malaysia Pahang, Pahang, Malaysia

***Abstract*—** In the present decade, there is the revival of interest to find the best way to query the database that able to provide not only a single answer, but also a set of answers that can be the most preferred by users. In many cases, the overflow of data generated from social media, and many other data stored and shared over the Internet makes the data access becoming near-infinite to the users. This has led to the need of intuitive formulation that able to provide the best choices from every unconceivable situation. Recently, the skyline computation has gained a lot of attention in the database research community for advanced queries semantic. The skyline query is introduced to be the syntax extension in SQL query to support multi-criteria data selection involving advanced queries. This paper surveys the techniques employed in initial algorithms of the skyline query processing. Some trade-offs about those different approaches are also identified throughout this study.

***Keywords*—** *Skyline query, Block-nested loop, Branch and Bound, Divide and Conquer, Bitmap, Nearest Neighbour.*

## I. INTRODUCTION

The growth in demand of decision support system and the increasing size of multidimensional data have led the researchers to find a new efficient way for data processing to get the useful insights. The operational research science is the field area that supports the decision-making process by using various advanced analytical methods such as mathematical models, statistical analysis and data mining. The skyline operator proposed by [1] is the most preference queries that is more specific and relevant for data filtering in the database [2] to give flexibility in the database [3], as well as to support the data complexity [4]. Conflicting in multi-criteria data selection can be intelligently processed to help users in complex decision making process. Skyline operator relies on a Pareto dominance concept that is extremely important in multi objective optimization. In such cases, there is no single point of answers for the best value for a specific objectives and constraints. Instead, the best answers, often called non-dominated set that always need to compromise with the quality for at least one objective or constraint, while improving at least one other. The skyline queries computing problem has also been investigated under the problem of maximum vector field [5]. The skyline operator can be static (absolute), where there are no changes of values in attributes (data points) for the minimization. It can also be relative, in which, it depends on the best distance of coordinate between the data points for the minimization. The syntax of skyline queries was formally presented in [1] and it is written as the extension of the syntax in SQL as shown in Fig 1. The set of dimensions specified in the SKYLINE clause are the criteria set by users for selection optimization. The MIN property in SKYLINE clause enables the SQL to find the minimum criteria of a specific dimension. Whereby, the MAX property in SKYLINE clause is use to find the maximum criteria of a specific dimension. The DIFF property indicates that no comparisons need to be made on two tuples with different values of specific dimensions. The existing studies on skyline queries assume the absolute setup, in which there are no changes in the distance between data points and queries points. As mentioned in [1], the dataset cannot completely fits in the main memory (RAM) for processed. This is likely to be the case with many modern database systems, where the dataset is retrieved from an external memory such as disks. Methods that do not rely on external memory are DD&C [6], LD&C[7] and FLET [8]. Given a set of attributes (objects) in a database, object X is said to be dominated by another object Y with the assumption that Y is equal or better than X in all attributes and at least has one better attribute. To illustrate on how the skyline operation works, given a set of data values for hotels, lets the x-axis is the distance between each hotel to the beach and the y-axis is the price for each hotel. Given the two points p=(p1,...,pd) and p0=(p0 1,...,p0 d) in relations, p dominates p0 if $p_k \leq p0_k$ for $1 \leq k \leq d$ and $p_m < p0_m$ for some $1 \leq m \leq d$. To make it easily understand, the dominance area of a 2-dimensional point is the North-East quadrant of the space that occurs by imaginably drawing an x-y axis system with origin point, the point of interest that is examined. The dominance area of a point will be in the dominance area of a second point (indicated as the second point is dominated by the first point) only if the first point is as good in all dimensions as well as good at one best dimension based on the

evaluation criteria. The skyline refers to the points that are dominating the other points in a given set of database. In the house-metro station for example, "better" means minimizing the values. To illustrate this scenario, Fig 2(b) is the skyline for a set of houses database presented in Fig 2(a). The houses indicated by point's h2, h6 and h4 is not the skyline as it is not the top choice. This is because in each one of them, at least one house is better in terms of the price or the distance. House h4=(4,125) is dominated by the house h6=(3,75) because it has better price and distance. The houses h1, h3, and h5 are the dominance points and are the skyline points as they are not dominated by any other houses. The skyline points are connected by a line as shown in Fig 2(b).

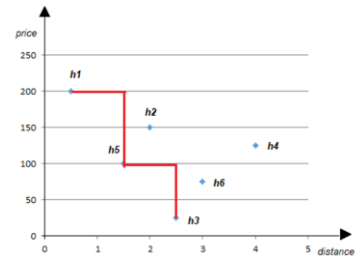
The computation in the skyline research is similar to the problem of maximal vector in computational geometry [9] or equivalently to the problem of Pareto optimal in operations research [6, 9]. The maximal vector computation involved the process of identifying the maximal set of vectors such that each one of them free from domination by other vector from the data set. The maximal vector research was first initiated in sixty centuries by a group of researchers in mathematical field. As mentioned in [19], the primary issue highlighted by the skyline query is the disability of the main memory (RAM) to completely hold the dataset for processed. This issue mostly related with many modern databases in which, the retrieving process is done at the external memory. Authors in [10, 11] stated that the initial algorithms (maximal approach) has an issue on performance, concerning the dimensionality. This approach divides the initial problem into sub-problems equally. Each sub-problem is solved separately that eventually merged together to yield the end result [12]. In addition, these algorithms assumed that the whole datasets are able to fits in memory without considering the memory limitations. Such kind of approach suffers from the "curse of dimensionality" [13], which was first used by Bellman [14] and is often used to demonstrate that high dimensionality causes problems due to the increases of computational cost. This problem was observed and solved by the introduction of the skyline operator [11], which proposes an algorithm called a divide and conquer that is believe to be efficient for external memory to be integrated into a database system. With the advent research of skyline operator, numbers of efficient computation algorithms have been developed over the years. These algorithms make used the strategies consist of the nearest neighbor search [15], divide-and-conquer [1], index structures [1, 16, 17] and sorting [18] to answer the skyline query in general. There are also number of studies that are focusing on the processing of skyline query in many other domains such as data streams [7] as well as the data on cellular gadgets [19]. This paper surveys the initial algorithms of skyline query processing to solve many universal problems.

```
SELECT ... FROM ... WHERE ...
GROUP BY ... HAVING ...
SKYLINE OF [DISTINCT] d1 [MIN | MAX | DIFF],
                  ..., dm [MIN | MAX | DIFF]
ORDER BY ...
```

Fig 1. Skyline Operator Syntax [1]

| Hotel | Distance (Km) | Price |
|---|---|---|
| *h1* | 0.5 | 200 |
| *h2* | 2 | 150 |
| *h3* | 2.5 | 25 |
| *h4* | 4 | 125 |
| *h5* | 1.5 | 100 |
| *h6* | 3 | 75 |

(a)




(b)

Fig 2. A two dimensional database

## II. Block Nested Loop (BNL)

Block Nested Loop (BNL) algorithm [1] is particularly works well in a small size of skyline and usually stops after one or two iterations to fits into the memory. BNL is based on the concept of scanning a set of the candidate points and place it in the memory. The first list of the candidate point is treated as the baseline in memory and the subsequent point *p* is then process based on three different cases. In the first case, if other points in the list dominates *p*, then *p* is discarded from the skyline. In the second case, all points that have been dominated by *p* are discarded and *p* is then added into the list. In the third case, *p* will be added in the list, if *p* is not either dominates or been dominated by others. In case if the memory overflow due to the growing number of points in the list, all points in the third case will automatically located in temporary file on the disk as shown in Fig. 3. For the small size of memory, to process the skyline operator, multiple passes of Block Nested Loop is required. As a matter of fact, the only candidate points inserted after the first pass can be classified as skyline before the temporary list is created. Any insertion of points into the list after the temporary list is created may not be considered as skyline points due to the fact that they haven't gone through the comparison process against the list points located in the temporary file. The process is then repeated all over again for the next passes in which all these pointers in the temporary list together with the candidate pointers are later be treated as the input. The most expensive process in BNL is to make comparison between a point and the set of points from the candidate list. To reduce the cost of this process, the self-sorting process of the pointers should be take placed before the comparison can be made, to allow the dominated points to be located into the top of the list. With this method, the top of the candidate's list will always consist of the highest dominating points and will be first to be compared with the subsequent points. The benefits of this algorithm is that, its simplified the process in which, any

dimensionality can be applied without the need to sort and index the related data file first. As a matter of fact, it can also be used to the different types of constraints as long as it specified two tuples in the preferences relation. The adequacy of this algorithm is depend on the main memory as well as its progressive processing. If the input is large and the memory size is too small to fit the input, multiple iterations are needed to compute the results. The reading need to be done for the entire input file before the result of the first point (skyline) can be yielded.

## III. Divide and Conquer (D&C)

In this algorithm [3], the dataset is recursively split into partitions $\{P_1,\ldots P_m\}$ in order to fits in the memory size (see Fig 3.). The boundaries of the partitions are identified through the compute of the q-quintiles of the dataset that eventually divided the database into q-1 into equal subsets. The results that yielded in each partition are merged together to yield the final result of the skyline. The interesting facts about this technique is that, some of the partitions P, may be ignored during the merging process as all points in the partition P have been dominated by some other partitions. The upper right partition as shown in Fig 3. for example, can be ignored and discarded as it's dominated by the lower left partition. This algorithm is mostly efficient if the datasets size is small and able to fit in memory. It is also unable to yield the skyline points progressively due to fact that the skyline point (first dominated points) can only be revealed when the entire dataset has gone through the scanning process. However, for the huge datasets, the apportioning procedure requires the read-write process for the entire dataset in any event once. This could possibly cause the critical cost of I/O. The process can performs better if the main memory size increases as it needs the partitions to be placed in the in-memory. The D&C computation is less sensitive compare to the BNL computation in terms of its correlations and the number of dimensions in the database. The algorithm of the divide and conquer is works in two steps, ($1^{st}$ step) form a given set of input points, the α-quintiles is computed along with a specific dimension $d_p$. The points are then split into m partitions so that each partition ($P_1$, ,,,,,$P_m$) able to fits in the memory and ($2^{nd}$ step) using the computation algorithm, the skyline points Si is computed in Pi partition where *i* in (1,,,m) and the overall points of skyline are then merged together and compute to yield the definite points of skyline.

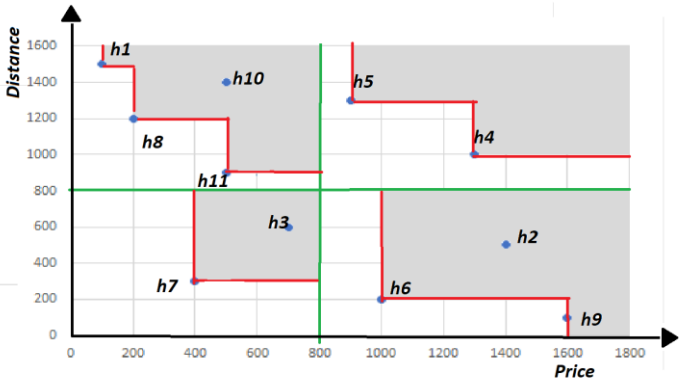



Fig 3. The Divide and Conquer technique

## IV. Bitmap

This techniques introduced by [16], requires the encoding process for every dimensions *i* into a *k* bit binary vector in which, *k* is the distinct value of *i* dimension. Each dimension need to be examined to determine whether it is dominated by other or not. If the answer is negative point, then it turns out to be an interesting point. Fig 4. is the example of an object with two dimensions d1 and d2. Dimension d1 consist of 7 distinct values for d2 consists of 6 distinct values. To determine whether the point c (4, 4) is in the skyline point or not, the three steps involced for the Bitmap computation are as follows:

- Step1: Let A be the BITWISE-AND operation of the dimensions, with *i* is the dimension and $p_i$ is the $m_i^{th}$ distinct value of the dimension. This operation requires a bit-slice on A. Each dimension has $n^{th}$ value, that equal or greater to the corresponding value of x dimension (e.g. The vertical bit slice for point (4,4) is (00101111, 01111000). The AND operation is performed for all bit slices and the result produce from this operation is 00101000) (see Fig. 4.).
- Step 2: Let B be the BITWISE-OR operation of the dimensions. The OR operation is performed on the preceding bit-slice of the smallest value that is greater than $x_i$. The bit-slice is then set to 0 if the preceding bit-slice is not exist for the *i* dimension. In this operation, if $n^{th}$ point is greater than the corresponding value of x's dimension, then it will be set to 1. (e.g. The vertical bit slice for (4,4) is (00101111, 01111000). The OR operation is then performed for all bit slices and the result produce from this operation is 01111111) (see Fig. 4.).
- Step3: Let C is the result produced from the AND operation of A and B. In this operation, value 1 is set to the *n*th point if it is greater than the value of x's dimension (e.g., point c). In this operation, we can say that x is dominated by the $n^{th}$ point. The zero value produced in the bit-slice operation shows that there are no other points dominates x, and x is said to be the definite skyline point. (see Fig. 4.).

| | d1 | d2 |
|---|---|---|
| a | 2 | 3 |
| b | 3 | 5 |
| c | 4 | 4 |
| d | 1 | 7 |
| e | 6 | 5 |
| f | 4 | 1 |
| g | 7 | 1 |
| h | 5 | 2 |

| Bitmap | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| distinct d1= 7 | | | | | | | distinct d1= 6 | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 6 | 5 | 4 | 3 | 2 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

| Object | Coordinate | Bitmap | Step1 : Bitwise AND Operation (A) | Step2: Bitwise OR Operation (B) | Step3 : Bitwise AND Operation (A) AND (B) |
|---|---|---|---|---|---|
| **a** | **(2,3)** | (11101111, 11111000) | 11101000 | 11111111 | 11101000 |
| **b** | **(3,5)** | (01101111, 01011000) | 01001000 | 01111111 | 01001000 |
| **c** | **(4,4)** | (00101111, 01111000) | 00101000 | 01111111 | **00000000** |
| **d** | **(1,7)** | (11111111, 00100000) | 00100000 | 11111111 | 00100000 |
| **e** | **(6,5)** | (01111111, 01011000) | 01011100 | 01111111 | 01011111 |
| **f** | **(4,1)** | (00101111, 11111111) | 00101111 | 11111111 | 00101111 |
| **g** | **(7,1)** | (00000010, 11111111) | 00000010 | 11111111 | 11111101 |
| **h** | **(5,2)** | (00001011, 11111001) | 00001001 | 11111011 | 11111011 |

Fig 4. Bitmap Computation Method

## V. BRANCH AND BOUND (BBS)

This algorithm proposed by [20] provides the ability to progressively yield the skyline points. The R-tree is utilized to list the multi-dimensional tuples, albeit other ordering strategies can be utilized as well. In the Branch and Bound algorithm setting, a new n-dimensional is specified by a dynamic skyline query depend on data space (the original d-dimensional). For each entry that related to the minimum bounding region (MBR) or leaf in the R-Tree, a parameter named mindist data point is measured to represent the short distance to the origin entry/leaf. The mindist value is calculated by adding all its coordinates and the MBR is calculated by adding all its lower left coordinates. The BBS algorithm is presented in Fig 5. The entries starts from the root of the R-tree to the heap and the insertion process always keep all the entries in ascending order based on the mindist. In this algorithm, the top of the heap in the R-Tree is expanded with the smallest entry of mindist and the child nodes is inserted with the points that are not dominated by any discovered skyline points in the tree. This algorithm always removes the points of the top entry to the next of the top entry if the points have been dominated by any other discovered skyline points. To accelerate the dominance checking process, the in-memory is created for all discovered skyline points in the R-Tree. The authors [20] has proved that this algorithm provide an optimal I/O processing (O(sh), s=result size and h = R-Tree height). In other words, its only visits the region with skyline points and discard the possibility of multiple access of the same node. For this situation, $\Theta(s)$ is the memory required for the BBS where it is equal to the skyline size. This algorithm depends on its capacity to trim the intermediate entries as the optimal measure if the entries fall under the dominance regions of the discovered skyline points. Hence, the point-to-point checking between the entry points and skyline points can be eliminated to speed up the entire process. The illustration of BBS algorithm is presented in Fig 5. The algorithm proposed by [17] is the most significant work in BBS. However, BBS only addressed a more general problem. The authors in [21] proposed the spatial skyline queries due to the limitation of BBS where it can only addressed the general problem of skyline query and overlooks the geometric properties. In SSQ, the checking process in each dominance requires the dynamic distance attributes computation. It utilized the R-tree-based $B^2S^2$ which is the improvement version of BBS [8]. In $B^2S^2$, the expensive dominance checks are avoided to get the skyline points and it is said better than BBS it also prunes query points that unnecessary to minimize the examination cost.

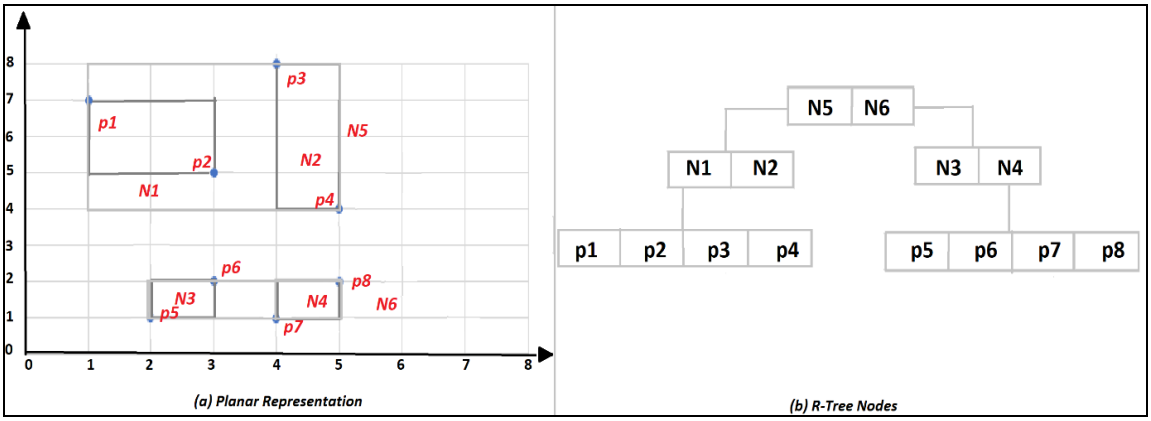



Fig 5. BBS Algorithm Illustration

## VI. Nearest Neighbour (NN)

The Nearest Neighbour (NN) algorithm [15] is the first algorithm that utilized the widespread of R-tree index structure [22, 23], that massively discard unwanted points by eliminating the redundant dominance checks process. In the NN search, the algorithm mentioned in [24, 25], which based on any monotone distance function are applied recursively. NN is an interesting algorithm for skyline query which returns the nearest neighbor points to origin as skyline. The first initial point is obtained by segmenting the space into four partitions. The great thing about this method is that the first initial point can be returned very fast. In the first iteration, the partition which consists of the points that are dominated by the preceding NN search point (the minimum distance from the axes) will be removed. The data space is then is divided into four spaces again. Only the partitions that are not dominated with the NN search point will be processed recursively by the To-Do list. If a region is empty, it will is not sub divide any further and will be removed from the To-Do list. The algorithm terminates when the To-Do list is empty. Fig 6. and Fig 7. Illustrates the first two partitioning process that recursively done in NN. Initially, the nearest neighbour of the point a to the origin need to be located and the universe need to be divided into three regions (1) subdivision of segment 1 and 2 which form the first region [0,2)[0,∞), (2) the subdivision of 1 and 3 which form the second region [0,∞)[0,3) and (3) the subdivision of 4 which form the third region (2,∞)(3,∞). Since segment 4 is dominated by point a, it can be pruned to be removed from the list. NN algorithm is applied on all four segments one by one. The NN point is e. The NN is applied again in which the e point divides this segment into subsegments. The skyline points may exist in those segments and will be explored further using NN (subdivision 1' and 2' and subdivision 1 and 3) . NN's performance on the data with more dimension I (I > 2 ) is not satisfactory due to the overlapping existed among the partitions. Another drawback of this algorithm is the size of the to-do-list may be exceeded although the dimensions of the data set is low.

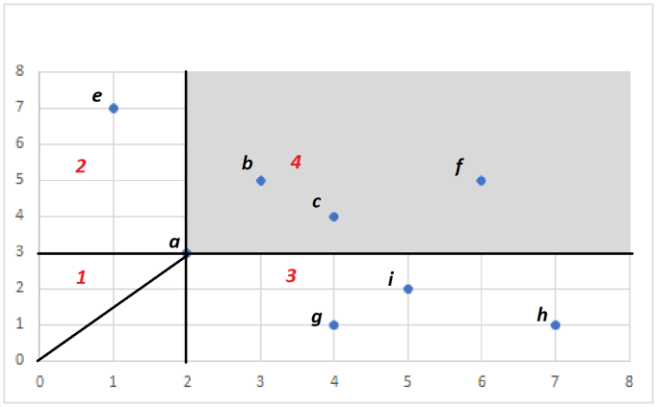



Fig 6. The first iteration process of NN

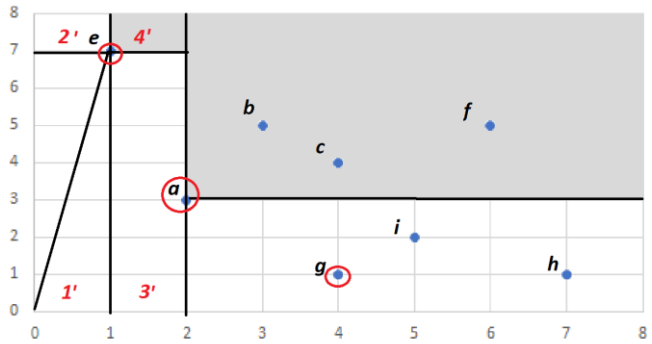



Fig 7. The second iteration process of NN

## VII. Conclusion

This paper summarized the important work of some initials studies in skyline query processing computation algorithms. Throughout this study, we have identified the techniques employed in each algorithm and the summary is presented in Table 1. Some trade-offs about those different approaches is also identified throughout this study. In the Block Nested Loop (BNL) algorithm for example, its computation is particularly work very well in a small size of

skyline. If the input is larger than the memory size to fit the input, multiple iterations are needed to compute to get the results. This is similar to the Divide and Conquer algorithm, where the apportioning procedure on huge dataset requires the read and writes process on the entire dataset that causing the critical cost of I/O. The Bitmap algorithm requires the encoding process for every dimension in the database. Even though this algorithm is a progressive index-based computation, all dataset points need to be encoded in order to get the definite skyline points which could also lead to the expensive cost of operation because the bitmap computation is requires for all points each time a point is inspected. This approach performs well when the number of distinct values per dimension is small. In the Branch and Bound algorithm, the R-tree is utilized to list the multi-dimensional tuples. This algorithm lies in its capacity to trim the intermediate entries as the optimal measure if the entries fall under the dominance regions of the discovered skyline points. The Nearest Neighbor algorithm utilized the widespread of R-tree index structure. The performance of this algorithm on the dataset with more than two dimensions is not satisfactory due to the overlapping existed among the partitions. Another drawback of this algorithm is the to-do-list size may be exceeded even there is low dimensions of the dataset. It can be concluded that most of the algorithms presented in this study are designed for small dataset with low dimensional setting. The curse of dimensionality could happen during the analyzing and organizing data due to the high dimensionality of data spaces that need to be processed. This can eventually affect the accuracy of the result in skyline processing. In reality, skyline query tends to grow exponentially due to the increasing number dimension. Therefore a new method is crucially needed to address these issues. Improving the computation performance without compromising the accuracy of skyline query processing data is a promising research area and is the future direction of this research work.

TABLE 1. Summary of skyline query processing algorithms

| Algorithm | Technique |
|---|---|
| Block Nested Loop [1] | Pair wise comparison. based on the concept of scanning the set of candidate points and placed in memory |
| Divide and Conquer[1] | Data cut into smaller pieces to make it fit into memory. Each of them needs to be processed using the in-memory algorithms. The results obtained from each process are then merged to yield the definite skyline point. |
| Bitmap [26] | Every dimension for all tuples is encoded using bitmap and the skylines point is yielded through the computation of fast bitwise. |
| Branch and Bound[20] | Exploring all possible skyline points via the branch-down process in R-tree entries and prune away the entries that dominated by the pre-found skyline point. |
| Nearest Neighbor[15] | The nearest neighbor is used to search the skyline points and the space is further divide for recursive processing. |

## *References*

[1] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Data Engineering, 2001. Proceedings. 17th International Conference on*, 2001, pp. 421-430.

[2] J. Chomicki, "Preference formulas in relational queries," *ACM Transactions on Database Systems (TODS)*, vol. 28, pp. 427-466, 2003.

[3] W. Kießling, "Foundations of preferences in database systems," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, 2002, pp. 311-322.

[4] S. Chaudhuri, N. Dalvi, and R. Kaushik, "Robust cardinality and cost estimation for skyline operator," in *null*, 2006, p. 64.

[5] H.-T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *Journal of the ACM (JACM)*, vol. 22, pp. 469-476, 1975.

[6] H.-T. Kung, F. Luccio, and F. P. J. J. o. t. A. Preparata, "On finding the maxima of a set of vectors," vol. 22, pp. 469-476, 1975.

[7] J. L. Bentley, H.-T. Kung, M. Schkolnick, and C. D. Thompson, "On the average number of maxima in a set of vectors and applications," CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE1977.

[8] J. L. Bentley, K. L. Clarkson, and D. B. J. A. Levine, "Fast linear expected-time algorithms for computing maxima and convex hulls," vol. 9, pp. 168-183, 1993.

[9] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*: Springer Science & Business Media, 2012.

[10] J. L. J. C. o. t. A. Bentley, "Multidimensional divide-and-conquer," vol. 23, pp. 214-229, 1980.

[11] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in *Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 229-240.

[12] D. Papadias, Y. Tao, G. Fu, and B. J. A. T. o. D. S. Seeger, "Progressive skyline computation in database systems," vol. 30, pp. 41-82, 2005.

[13] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?," in *International conference on database theory*, 1999, pp. 217-235.

[14] R. E. Bellman, *Adaptive control processes: a guided tour* vol. 2045: Princeton university press, 2015.

[15] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in

*Proceedings of the 28th international conference on Very Large Data Bases*, 2002, pp. 275-286.

[16] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *VLDB*, 2001, pp. 301-310.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Transactions on Database Systems (TODS)*, vol. 30, pp. 41-82, 2005.

[18] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting: Theory and optimizations," in *Intelligent Information Processing and Web Mining*, ed: Springer, 2005, pp. 595-604.

[19] X. Lin, Y. Yuan, W. Wang, and H. Lu, "Stabbing the sky: Efficient skyline computation over sliding windows," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, 2005, pp. 502-513.

[20] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 467-478.

[21] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 751-762.

[22] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *Acm Sigmod Record*, 1990, pp. 322-331.

[23] A. Guttman, *R-trees: A dynamic index structure for spatial searching* vol. 14: ACM, 1984.

[24] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *ACM sigmod record*, 1995, pp. 71-79.

[25] G. R. Hjaltason and H. J. A. T. o. D. S. Samet, "Distance browsing in spatial databases," vol. 24, pp. 265-318, 1999.

[26] P. Godfrey and W. Ning, "Relational preference queries via stable skyline," Citeseer2004.