Journal of Computer Engineering
ISSN 10003428

Original Research Article
Vol 12, Issue 5, 2023 | ISSN: 1000-3428

Efficient Computation of Continuous Skyline Queries with Unified Attribute Handling

Amira S. Elsayed, Kareem F. Abd El-Kader

Department of Computer Science, Ain Shams University, Cairo, Egypt; Faculty of Computers and Artificial Intelligence, Benha University, Benha, Egypt

***Abstract*—**Although most of the existing skyline queries algorithms focused basically on querying static points through static databases; with the expanding number of sensors, wireless communications and mobile applications, the demand for continuous skyline queries has increased. Unlike traditional skyline queries which only consider static attributes, continuous skyline queries include dynamic attributes, as well as the static ones. However, as skyline queries computation is based on checking the domination of skyline points over all dimensions, considering both the static and dynamic attributes without separation is required. In this paper, we present an efficient algorithm for computing continuous skyline queries without discriminating between static and dynamic attributes. Our algorithm in brief proceeds as follows: First, it excludes the points which will not be in the initial skyline result; this pruning phase reduces the required number of comparisons. Second, the association between the spatial positions of data points is examined; this phase gives an idea of where changes in the result might occur and consequently enables us to efficiently update the skyline result (continuous update) rather than computing the skyline from scratch. Finally, experimental evaluation is provided which demonstrates the accuracy, performance and efficiency of our algorithm over other existing approaches.

I. Introduction

WITH the expanding number of sensors, wireless communications and mobile applications and the fast developments in technologies for tracking the positions of moving objects, algorithms for efficiently answering queries about large numbers of moving objects are progressively required. This in turn surges the interest for location-based services (LBS).

In general, a moving object is an object whose location and/or geometry changes continuously over time [1]. Moving object databases (MODs) are databases developed to satisfy the need of new technologies to consider the huge amounts of continuously acquired location information. Unlike traditional databases which are most appropriate for static data, MODs are appropriate for dynamic data [17]. In addition, MODs are customized for high frequency of updates that is a regular result of rapidly changing location information [1]. Such information requires new types of queries which can query this spatial information, among those queries are: Range
queries, Nearest Neighbor queries, and Skyline queries.

Skyline queries are an important operator of LBS. Skyline computation has received considerable attention in the database community, especially for enabling LBS. A result of skyline query produced from a given data set is a subset of interesting points that are not dominated by any other point[2]. For example, mobile users could be interested in restaurants that are near, reasonable in pricing, and provide good food, service, and view. Skyline query results are based on the current location of the user, which changes continuously as the user moves. Using the common example in the literature shown in Fig. 1, there is information about hotels; the distance to the beach and the price for each data point is recorded. Consider a two dimensional plot of the dataset, where the distance and price are assigned to the X, Y axis of the plot. The goal of the skyline query is to find all the hotels not worse than any other hotel in both distance from the beach and the price. Hotels *a*, *i*, and *k* are interesting and can be inferred by the skyline query, for their distances to the beach and prices are desirable over those of other hotels.

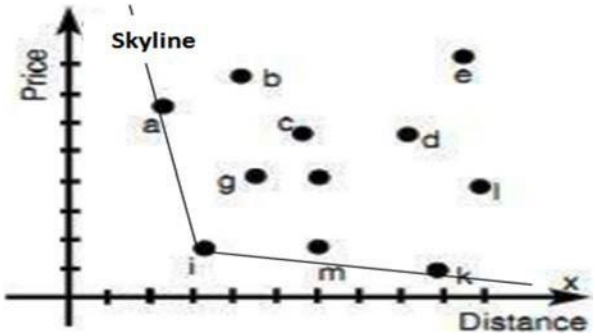

Fig. 1 An example of skyline in static attribute

In the previous example, the data set is static, where both the query point and the data points are static. What if the query point is a moving object? In this case, the distance between the query point and each point will no longer remain unchanged, it will change continuously. Now, let us change the example to the scenario of a tourist walking about to choose a hotel for his stay. For ease of illustration, we again consider just two factors, namely the distance to the hotel and the price per night. In this new scenario a new challenge exists, this challenge occurs from the fact that the distance from the tourist (i.e. a moving object) and every hotel becomes dynamic and changes as the tourist moves. Fig. 2 shows the movement of the moving object (i.e. the

How to cite: Amira S. Elsayed, Kareem F. Abd El-Kader (2023). Efficient Computation of Continuous Skyline Queries with Unified Attribute Handling. Journal of Computer Engineering, 12(5), 72–80. DOI: https://doi.org/10.5281/zenodo.19348821

tourist) which causes the updates on the skyline result. In Fig. 2 (a), X, Y represent the 2D spatial location of each hotel and t1, t2 represent the position of the tourist at two different time instances where the tourist moves from time t1 to t2, whereas Fig. 2 (b) shows their respective prices.. The skyline, i.e. interesting hotels, changes with respect to the tourist's position. Such problem is common in moving databases [3], [4].

(a) (b)

Fig. 2 An example of skyline in dynamic attribute

In this paper, we address the problem of continuous skyline query processing, where the skyline query point is a moving object and the skyline result changes continuously due to the movement of the query point. To solve this problem, we first distinguish the data points that will not be in the initial skyline result using the divide and conquer technique presented in [2]. Next, we investigate the connection between data points' spatial locations and their dominance relationship, which provides an indication of where to find changes in skyline result and update the query result according.

The rest of this paper is organized as follows. In Section II, we present a brief review of related work. In Section III, we propose our solution for continuously maintaining the skyline query. The experimental results are presented in Section IV. Finally, Section V concludes and proposes directions for possible future work.

II. Related Work

In this section, we will briefly explore previous work in skyline query computation. Three main approaches will be discussed: (1) Static skyline query computation, (2) Dynamic skyline query computation, (3) Parallel skyline query computation. For static skyline query computation approach, several algorithms have been introduced including: In [2], the authors proposed two algorithms namely, Block-Nested-Loop (BNL) and a Divide-and-Conquer (D&C) algorithm. The BNL algorithm iteratively compares each point in the dataset with all current skyline points existing in memory and returns the dominating points which fulfill the criteria of domination over all dimensions. On the other hand, the D&C algorithm divides the whole data space into a number of partitions which can fit in memory. For each partition, it uses the same way of BNL algorithm to check the domination over all dimensions of the points in the same partition and returns the skyline result for each one. Then it computes the final skyline result through merging the skyline result of each partition and producing the dominating points. In [5], the authors proposed a new progressive algorithm named Branch-and-Bound Skyline (BBS). The proposed approach is based on the use of an Rtree as an index structure. It takes data from the R-tree into a heap and sorts them based their distance to the query point. A new entry on the heap will be discarded if it is dominated by any skyline point, or inserted into the skyline result if it is not dominated by another point in the skyline. In [6] the authors proposed an algorithm namely, Group-based skyline (Gskyline) algorithm. G-skyline is interesting in analyzing a group of points rather than individual points. Skyline result includes the groups that are not dominated by other groups. The authors partitioned data into multi-layers and represented data in directional skyline graph including the dominance relationship between points in all layers to efficiently compute the group that has the best values along all dimensions. After that the skyline result will include all points existing in layer1. On the other hand, for the dynamic skyline query computation several approaches have been introduced including: In [4], the authors proposed an event-driven approach to maintain the result of k-NN query on moving objects. It first puts all moving objects into a list, then sorts them based on their current distance to the query point. Then, it creates events which determine the points of intersection (i.e. intersection represents when two adjacent moving objects will exchange their positions). All created events are pushed into a queue which sorts them based on the time of firing the intersection event; the priority will be to the event with earlier time. In [7], the authors proposed

another algorithm for continuous skyline queries. The proposed algorithm discriminates between the static dimensions and dynamic ones; however, it computes the skyline using the static dimensions only to retrieve the points that will be permanent skyline points, and uses the result to drive the farthest point, consequently enables the exclusion of the points which will not be in the initial skyline result. Then it computes the dynamic skyline by pre-computing the points of updates, and finally it merges the result of the static part and dynamic ones to conduct the final skyline result. In [8], the authors proposed direction-oriented continuous skyline query algorithm. The proposed algorithm computes the skyline points according to two approaches: (1) Any direction around the moving object, (2) The same direction of the moving object. In the first approach, the skyline points are the dominant points which gained the best values over all dimensions and located in any direction around the moving object. In the second approach, the skyline points are the dominant points which gained the best values over all dimensions and located along the moving object's direction. The authors used the same approach presented in [9] to retrieve the skyline points. In [9], the authors used the same technique proposed in [7] but considered the number of levels (k). The number of levels represents the number of iterations for which the dynamic part of the skyline will be computed. In [10], the author proposed a
new schema for continuous skyline query computation over skewed data. The proposed schema partitions the data into multi-layer grids. For each grid layer it conducts the skyline influence region which contains the cells that cannot be dominated by any other cell in the space. When the number of data points within one cell grows too large, then a second layer grid must be created; and its influence region is computed. The final skyline result will be all points existing in all influence regions. On the other hand, for the parallel skyline query computation several approaches have been introduced including: In [11], the authors proposed two algorithms for skyline computation using MapReduce framework namely, MR-BNL, and MR-SFS. The MR-BNL algorithm used the BNL algorithm presented in [2] to compute the skyline for all data points in each reduce task after partitioning data by map task. Finally the final skyline are computed by merging all the skyline results produced from each reduce and return the best points from all results. The MR-SFS apply the same procedure in the MR-BNL, but it sorts the file of data first to reduce the number of comparisons. In [12], the authors proposed an algorithm for parallel processing the skyline using map reduce. The algorithm first excludes the non skyline points which are dominated by the other points over all dimensions by using the quad tree. Then it partitions the remaining points in to a set of partitions based on the regions conducted from the quad tree and compute the skyline for each region using map reduce.

Because of the important role of skyline queries in many applications, such as multi-criteria decision making, data mining, and user preference queries, in this paper we follow the approaches presented in [9], [7] and present a new algorithm that provides better performance and accuracy.

III. Continuous Skyline Query

In general, a moving object is an object whose location and/or geometry changes continuously over time [1], [18]; this requires continuous evaluation for the query as the query result varies with the changing in query point location over time. Continuous skyline query processing has to re-compute the skyline when the objects move. Notwithstanding this, updating the skyline of the previous moment is more efficient than conducting a snapshot query at each moment and computing the skyline from scratch. For intuitive illustration, we limit the data and the moving query points to a twodimensional (2D) space. In Table I, we summarize the symbols used in this paper.

TABLE I
Symbols and Description

Symbol	Description
Dist|ti (pi, q)	Distance between data point (pi) and query point (q) at time (ti)
SK	The skyline result
≺	To denote that a point p1 dominate another point p2
⊀	To denote that a point p1 does not dominate another point p2

Because of dealing with moving query points, we consider the distance function to be the time parameterized distance which has been used in literature to help processing queries in MODs [13]-[15] rather than the traditional Euclidean distance. For a moving data point p_i starting from (,) with velocity (,), and a query point q starting from (,) moving with velocity (,), the distance between them can be expressed as following:

$$Dist|_t\,(p_i, q) = \sqrt{\qquad\qquad} \qquad (1)$$

where a, b, and c are constants determined by their starting positions and velocities with = =0 as all data points are static.

$$a = \qquad (2)$$

$$b = 2[(\ -\ )(\ -\ ) + (\ -\ )(\ -\ )] \qquad (3)$$

$$c = \qquad (4)$$

Let p_1, p_2 be two data points with k static attributes, where $k \geq 1$. Let $p_i.k_a$ denotes the value of static attribute 'a' of data point p_i $\forall$ i in the data set.

Definition1. Let p_1 and p_2 be 2 data points, if $Dist|_{t1}(p_1, q) \leq Dist|_{t1}(p_2, q)$ and $p_1.k_a \leq p_2.k_a$, $\forall$ k, and $\exists$ k, such that $p_1.k_a < p_2.k_a$, we say $p_1 \prec p_2$ at time t_1 (i.e. p_1 dominates p_2 at time t_1). **Definition2**. Let p_1 and p_2 be 2 data points, if $p_1.k_a = p_2.k_a$, $\forall$ k, And $Dist|_{t1}(q, p_1) < Dist|_{t1}(q, p_2)$ we say $p_1 \prec p_2$ at time t_1. **Definition3**. A continuous skyline query CSQ is defined as

CSQ = (p_{j1}, p_{j2},..., p_{jn}): where (p_{j1}, p_{j2},..., p_{jn}) are the best points which are not dominated by any other point in S.

In our solution, we only compute the initial skyline for the starting position at the start time t_0; subsequently, updating the skyline result instead of computing a new one from scratch each time.

Proposed Algorithm

Many of the data points may have the same values for all attributes. If these points have the best value in a specific dimension; this means that all these points will be in the skyline result as they gain the best value for a static dimension as mentioned in [9], [7]. This can cause incorrect results if using the term of skyline which depends on checking the domination over all dimensions. As we deal with a moving query point which continuously updates its location, this means that at a specific time it will be closer to a point than the others, however; this point will dominate other points at this time. Another disadvantage of the approach proposed in [9], [7] is that it needs to scan the data more than once, first time for checking domination on static attributes and return permanent skyline points, and a second time to compare the distance of the points which have distance less than the farthest point with each point in the skyline.

In this paper we present an algorithm for continuous skyline query processing without discrimination between static and dynamic attributes which provides better performance and accuracy. Our algorithm is composed of the following three phases:

Phase1. *Data Preprocessing*

After we compute the distance between each data point and the query point using (1)-(4); in the first step in this phase, we try to exclude the points which are guaranteed to be out of the initial skyline query, consequently reducing the number of comparisons in checking the domination afterwards. The algorithm distinguishes these points by partitioning the data into a number of partitions based on the median of each attribute, then, excluding the extreme points that have values greater than the value of the median of each attribute. In the next step we create a view "V" for the data set with the points excluded according to step 1. The procedure for computing the initial data points is presented in Fig. 3.

Phase2. *Compute Initial Skyline*

In this phase, we use the view "V" created in phase 1 to check the domination over all points in this view and return the dominant points that have the best values (i.e. the points where there is no other point in the data set with better value along all dimensions); these dominant points represent the skyline result at starting position t_0. Fig. 4 shows the algorithm of domination.

Phase3. *Compute Continuous Skyline*

In this phase, we try to perform an early catch for the positions where the skyline result is expected to change instead of computing the skyline at every time. We use the time parameterized distance function presented in [13] to build the equation representing the distance between each data point and the query point. Then applying the sweep line algorithm presented in [16] on the resulting equations to compute the points of intersection which may affect the skyline result. At each point of intersection we have five cases. Table II shows the possible cases for the intersection of two points (P1, P2) and its effect on the skyline result (SK).

Fig. 3 Compute Initial Data Points Algorithm

Algorithm: Check_Domination (p, L_{sk})

Input: - p (Non skyline point)

- L_{sk} (list of current skyline points)

Output: L_{sk} (skyline points after comparing p against current skyline points)

```
1.  For each point S in Lsk
2.      If(Lsk.last)  // last element in skyline list
3.          If(s⊀ p, ∀ k)
4.              Lsk = Lsk ∪ {p}
5.              If(p≺ s, ∀ k)
6.                  Lsk = Lsk - {s}
7.      If(s≺ p, ∀ k)
8.          Break;
9.      Else
10.         If(p≺ s, ∀ k)
11.             Lsk = Lsk - {s}
12.             Continue
```

Fig. 4 Check Domination Algorithm

- **Case-1:** The two points are not skyline points; in this case, the skyline result will not change, so we do not have to check domination between the two points (i.e. we just ignore this intersection)
- **Case-2:** The two points are skyline points and have different values in the static attributes (i.e. no one of them dominates the other in any static attribute), so checking domination isn't needed.
- **Case-3:** The two points are skyline points, but one of them dominates the other in all static attributes and due to the position exchange its distance becomes less, then the second point must be deleted from the skyline result.

TABLE II
THE POSSIBLE CASES FOR THE INTERSECTION OF TWO

Case	Conclusion
P1 ∉ SK & P2 ∉ SK	SK will not change
$p_1 \in SK$ & $p_2 \in SK$ & $p_1 \nprec p_2$, ∀ k & $p_2 \nprec p_1$, ∀ k	SK will not change
p1 ∈ SK & p2 ∈ SK & p1≺ P2, ∀ k & Dist|ti (p1, q) < Dist|ti (p2, q)	SK will change and P2 will leave SK
P1 ∈ SK & P2 ∉ SK & P1.ka = P2.ka, ∀ k	SK will change and P1 will leave SK and P2 will enter to SK
P1 ∈ SK & P2 ∉ SK & P1.ka ≠ P2.ka, ∀ k	SK will change and P2 will enter to SK

- **Case-4:** One of the two points is a skyline point and the second point is not, but they have the same values over all static attributes and non skyline point distance gets less; in this case the non skyline point becomes a skyline point and the skyline point becomes a non skyline point.
- **Case-5:** One of the two points is a skyline point and the second point is not, but they do not have the same values over all static attributes and non skyline point distance gets less; in this case the non skyline point becomes a skyline point and the skyline point remains in the skyline

IV. EXPERIMENTAL EVALUATIONS

In this section, we present our experiments for evaluating our proposed (ECSQ) algorithm. We evaluated the performance of the ECSQ algorithm by comparing it with the MCSQ algorithm [9] and CSQ presented in [7]. We conducted our experiments on a laptop running on MS Windows 7 professional. The laptop has a Core(TM) i5 2.53GHz CPU and 4GB memory. All experiments were coded in java. In this set of experiments, we used synthetic data sets of data points with 2D spatial attributes as well as 2 non-spatial attributes. For each data set, all data points are distributed randomly within the spatial space domain of 10,000 x 10,000, and the nonspatial attributes' values range from 1 to 100,000. The speed of each moving query point is also randomly generated and ranges from 10 to 80 km/hr. In the experiments we compare our algorithm ECSQ with the MCSQ algorithm presented in [9] and CSQ presented in [7] and used different data sizes and different number of static dimensions. In The first experiment we used two static attributes and we varied the size of the data set (100, 200, 300, 400, 600, and 800) and observed the query performance and CPU time. Fig. 5 shows that as cardinality increases, the CPU time cost of our solution grows steadily, in a rate much less than that of the two other algorithms. In the second experiment we fixed the data set size (i.e. 200 objects) and varied the number of static dimensions (2, 3, 4, and 5) and observed the query performance and CPU time. Fig. 6 shows that as number of static dimensions increases, the CPU time cost of our proposed solution outperforms the cost encountered by the other algorithms.

V. CONCLUSIONS & FUTURE WORK

In this paper, we presented ECSQ algorithm for efficiently computing continuous skyline queries. The presented algorithm updates Skyline query results rather than recomputing the skyline every time the dynamic attributes are changed. Experimental studies show that the proposed method is robust and efficient. For future work we aim to try to compute the continuous skyline query using large volume of data in a distributed framework.

REFERENCES

[1] O. Wolfson, B. Xd, S. Chamberlai, and L. Jiang, "Moving Objects Databases: Issues and Solutions," Proceedings of the 10th International Conference on Scientific and Statistical Database Management, pp.111122, 1998.

[2] S. Borzonyi, D. Kossmann, and K. Stocker, "The Skyline Operator," Proc. Int'l Conf. Data Eng., pp. 421-430, 2001.

[3] R. Benetis, C. Jensen, G. Karciauskas, and S. Saltenis, "Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving

Objects," Proc. Int'l Symp. Database Eng. & Applications, pp. 44-53, 2002.

[4] H. Mokhtar, J. Su, and O. Ibarra, "On Moving Object Queries," Proc. 21st ACM PODS Symp. Principles of Database Systems, pp. 188- 198, 2002.

[5] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An Optimal and Progressive Algorithm for Skyline Queries," Proc. 2003 ACM SIGMOD Int'l Conf. Management of Data, pp. 467-478, 2003.

[6] Liu, Jinfei, et al. "Finding pareto optimal groups: group-based skyline." Proceedings of the VLDB Endowment 8.13 (2015): 2086-2097.

[7] H. Zhiyong, L. Hua, O. Beng Chin, and K. H. T. Anthony, "Continuous Skyline Queries for Moving Objects." vol. 18: IEEE Educational Activities Department, pp. 1645-1658, 2006.

[8] E. El-Dawy, Eman, Hoda MO Mokhtar, and Ali El-Bastawissy. "Directional skyline queries." *Data and Knowledge Engineering*. Springer Berlin Heidelberg, 2012. 15-28.

[9] El-Dawy, Eman, Hoda M.O. Mokhtar, and Ali El-Bastawissy. "Multilevel continuous skyline queries (MCSQ)." Data and Knowledge Engineering (ICDKE), 2011 International Conference on. IEEE, 2011.

[10] Li, He, and Jaesoo Yoo. "An efficient scheme for continuous skyline query processing over dynamic data set." Big Data and Smart Computing (BIGCOMP), 2014 International Conference on. IEEE, 2014.

[11] Zhang, Boliang, Shuigeng Zhou, and Jihong Guan. "Adapting Skyline
Computation to the MapReduce Framework: Algorithms and Experiments." *DASFAA Workshops*. 2011.

[12] Park, Yoonjae, Jun-Ki Min, and Kyuseok Shim. "Parallel computation of skyline and reverse skyline queries using mapreduce." *Proceedings of the VLDB Endowment* 6.14 (2013): 2002-2013.

[13] K. Raptopoulou, A. Papadopoulos, and Y. Manolopoulos, "Fast NearestNeighbor Query Processing in Moving-Object Databases,"
GeoInformatica, vol. 7, no. 2, pp. 113-137, 2003.

[14] Iwerks, Glenn S., Hanan Samet, and Ken Smith. "Continuous k-nearest neighbor queries for continuously moving points with updates." *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment, 2003.

[15] Tao, Yufei, and Dimitris Papadias. "Time-parameterized queries in spatio-temporal databases." *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 2002.

[16] T. T. El-midany, A. Elkeran, and H. Tawfik, "A Sweep-Line Algorithm and Its Application to Spiral Pocketing," vol. 2, no. 1, 2002. [17] Hoda M. O. Mokhtar and J. Su." A Query Language for Moving Object Trajectories", *Proceedings of the International Scientific and Statistical*
Database Management Conference (SSDBM), University of California, Santa Barbara, June 27-29, 2005.

[18] Hoda M. O. Mokhtar and J. Su. "Universal Trajectory Queries for Moving Object Databases", Proceedings of IEEE International Conference on Mobile Data Management, Berkeley, CA, January 19-22, 2004.