2012 23rd International Workshop on Database and Expert Sytems Applications

Making Recommendations using Location-based Skyline Queries

Marlene Goncalves and Daniela Torres
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
{mgoncalves,daniela}@ldc.usb.ve

Graciela Perera
Computer Science Department
Youngstown State University
Ohio, USA
gcperera@cis.ysu.edu

***Abstract*— Including geo-spatial features into queries over Internet is possible with location aware devices. On the other hand, there are current applications where a user is interested in viewing the best objects chosen from a large collection, based on multiple criteria. Skyline filters out those objects that best match user's preferences. The fusion of geo-location and preferences makes possible a new kind of Skyline queries that takes into account both location proximity and user's preferences. A location-based Skyline is an extension of Skyline, which depends on geographical locations of interest objects. In this work, we have developed a web tool that displays the results of a location-based Skyline query and propose a new algorithm to evaluate location-based Skyline queries. Our experimental study shows our proposed algorithm outperforms existing algorithms for high-dimensional queries when data are non-duplicated.**

Keywords: Skyline queries, location-based queries, georeferencing systems.

I. INTRODUCTION

The advent of mobile devices and application development has enabled access to large amounts of data over the Internet that may be queried using geo-spatial features. Currently, map engines such as Google Maps include support for location-based queries. Location-based queries consider location nearness to an object of interest described by latitude and longitude coordinates. Moreover, these objects may be associated with user's ratings.

Suppose a tourist will travel to Paris next summer and prefers to stay in a hotel near the Eiffel Tower. He wants to plan his accommodation in terms of criteria such as service quality, price, and nearness to the Eiffel Tower. For this purpose, he searches the Zagat hotel-review website[1] for user's ratings. He retrieves the service quality and price of hotels in Paris. Afterwards, he uses the Google Maps website[2] to find the hotels within walking distance to the Eiffel Tower. He may make a decision based on three equally important criteria. That is, hotels with low prices, high service quality, and with the shortest walking distance to the Eiffel Tower. Thus, a hotel is selected if and only if there is no other hotel with shorter distance to the Eiffel Tower, higher service quality, and lower price. For example, consider the following four hotels, *X*, *Y*, *Z*, and *W* with their respective distance to the Eiffel Tower, service quality, and price. That is, *X*(0.1 miles, 66%, 150€), *Y*(0.3 miles, 100%, 100€), *Z*(0.4 miles, 83%, 100€), and *W*(1 mile, 83%, 50€). Hotel *Y* is better than hotel *Z* because *Y* has lower distance, higher service quality, and equal price than *Z*, i.e., *Y* dominates *Z*. However, hotel *X* is closer to the Eiffel Tower when compared to *Y* but hotel *Y* has better service quality and price than *X*. Consequently, *X* and *Y* are incomparable.

Intuitively, the tourist's decision-making process may be supported by a location-based Skyline query. A Skyline query retrieves non-dominated hotels; a hotel dominates another hotel if it is as good or better than another hotel in all criteria and better in at least one criterion. Skyline queries may help to make a decision when different and conflicting criteria are considered. In our example, in spite of hotel *X* being the closest to the Eiffel Tower, its price is the highest. Even though, hotel *W* has the lowest price, it is the farthest hotel from the Eiffel Tower. Thus, it is not possible to find a hotel closer to the Eiffel Tower with a lower price and higher quality service. It is hard to select an economical hotel that is close to the Eiffel Tower because the hotels near the Eiffel Tower are expensive. Thus, economical and nearness are conflicting criteria. Skyline is ideal in this situation as it discards hotels that are worse on all criteria than some other.

The tourist's criteria are a fusion of geo-location and preferences. This makes possible a new kind of Skyline queries that takes into account both location proximity and user's preferences. A location-based Skyline is an extension of Skyline, which depends on geographical locations of interest objects. A distance function from the interest object location to a given geographical location is part of Skyline criteria. Participatory sensing mobile applications are very popular as they combine user generated data with latitude and longitude coordinates from GPS enabled mobile devices. These applications enable individuals and communities to gather and analyze a broad range of data along with their location-based data [4]. Surely, many upcoming applications will require location-based Skyline queries.

Because Skyline queries need to be efficient and useful, its computation has received the attention of many researchers. Divide and Conquer extension [1], Block-Nested-Loops (BNL) [1], Sort-Filter-Skyline (SFS) [3] and LESS (Linear Elimination Sort for Skyline) [6] are algorithms than scan the whole table in order to evaluate Skyline queries. Also, index-based Skyline algorithms have been introduced. B-tree [17], bitmap [17], nearest neighbor [10], BBS [12] are all algorithms based on composite

[1] www.zagat.com
[2] http://maps.google.fr

1529-4188/12 $26.00 © 2012 IEEE
DOI 10.1109/DEXA.2012.44

indices. BDS (Basic Distributed Skyline) [2] and PDS (Progressive Distributed Skylining) [11] were thought over Web sources. These solutions work on attributes whose values are stored into a database. Therefore, to evaluate a location-based Skyline query we must pre-compute the distance function for each object of interest. On the other hand, a location-based Skyline query algorithm is presented in [9]. The algorithm uses categorical data from mobile environments. It first creates a table containing categorical data. Then, it discards elements of the table until the table is empty. Since this table stores categorical data, its size is small. However, creation and management of this table is not scalable when criteria involve non categorical attributes because the size of this table can still be very large. Thus, the algorithm is impractical if the criteria have non categorical attributes. In [16], Spatial Skyline queries and their evaluation algorithms were defined. Nevertheless, Spatial Skyline queries were thought as a more general case of location-based Skyline queries where multiple distances are considered as part of Skyline criteria.

In this work, we have developed a web tool that evaluates location-based Skyline queries and displays the best objects as points in a map. This tool may be used to make recommendations to the user. It will indicate the best places to visit according to his current location. Additionally, we propose an algorithm to evaluate location-based Skyline queries.

Finally, this paper comprises five sections. We present in Section 2 the basic definitions of Skyline and establish a theorem that is the foundation on how our proposed algorithm can evaluate location-based Skyline queries. Section 3 describes our web tool called MapSkyline and our proposed algorithm for location-based Skyline queries. In Section 4, we describe our experimental results. Finally, in Section 5, the concluding remarks and future work are pointed out.

II. PRELIMINARES

The skyline of a set O of multi-dimensional points (objects) consists of all the points of O not dominated by any other point. A point o_i is said to dominate another point o_j, if o_i is better or equal than o_j in all dimensions and o_i is better than o_j in at least one dimension.

Given a set $O = \{o_1, \ldots, o_n\}$ of spatial objects. Each spatial object o_i in O has a location in the space determined by $o_i.loc$ and is described by r attributes. We denote the distance between an object o_i and a location l by $d(o_i.loc,l)$. Let a preference function m be defined over the distance between each object of O and a given location l, and r attributes $\{a_1, \ldots, a_r\}$ of the objects belonging to O. The Location-based Skyline LS_m according to preference function m is presented in Definition 1. For simplicity, we suppose that the dimensions (or attributes) related to the preference function are maximized.

Our algorithm, which computes location-based Skyline queries, is based on Theorem 1. We observe that the Skyline for a set of attributes A= $\{a_1, \ldots, a_r\}$ may be part of the Skyline for a set of attributes A $\cup$ $\{a_{r+1}\}$.

Definition 1 (Location-based Skyline):

$$LS_m = \left\{ o_i \in O \;/\; \neg\exists\, o_j \left(\begin{array}{l} o_j \in O \wedge o_i.a_1 \le o_j.a_1 \wedge \\ \wedge \ldots \wedge o_i.a_r \le o_j.a_r \wedge \\ \wedge\, d(o_i.loc, l) \le d(o_j.loc, l) \wedge \\ \wedge\, \exists q \left(\begin{array}{l} 1 \le q \le r \wedge o_i.a_q < o_j.a_q \vee \\ \vee\, d(o_i.loc, l) < d(o_j.loc, l) \end{array} \right) \end{array} \right) \right\}$$

Theorem 1. Given a set O of spatial objects, a preference function m defined over a set of attributes A = $\{a_1, \ldots, a_r\}$ of the objects belonging to O, a preference function m' defined over the set of attributes A $\cup$ $\{a_{r+1}\}$ of the objects belonging to O. Each location-based Skyline object in LS_m is a location-based Skyline object in LS_m'; or is dominated by another location-based Skyline object in LS_m' with better or equal value in a_{r+1}.

Proof. For each location-based Skyline object o_i in LS_m, if there is another object o_j such that is better or equal in a_{r+1} than o_i, o_j dominate o_i according to preference function m' if o_i is better or equal than o_j in all attributes $\{a_1, \ldots, a_r\}$. Otherwise, if such a skyline object o_j does not exist, o_i is a Skyline object in LS_m' because no other object can be better than o_i in all attributes $\{a_1, \ldots, a_r\} \cup \{a_{r+1}\}$.

III. MAPSKYLINE

We have developed a Web tool that allows identifying the location-based Skyline and shows it on a map. This tool is called MapSkyline[3]. MapSkyline architecture has three-layers: interface, logical, and database. Figure 1 shows the architecture of MapSkyline. The Interface layer allows the user to interact with the tool to enter the Skyline query and then display its result on the map in Google Maps. Google Maps was integrated with MapSkyline using the API of Google Maps [7]. Moreover, JQuery library was used for the implementation of JavaScript scripts.

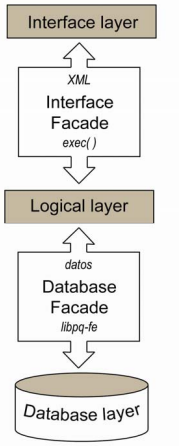

Figure 1 MapSkyline Architecture

The logical layer is composed of the algorithms implemented as part of the tool; it was written in C++. The choice of C++ is due to efficiency. If the algorithms are

[3] http://mapskyline.ldc.usb.ve

implemented directly in PHP, their response time may be higher [15].

The Interface layer was implemented in PHP. PHP allows to invoke external binary programs by means of the function exec(). This establishes the connection between the logic layer and the interface layer. Interface collects the search parameters entered by the user using JavaScript. Then, an AJAX call passes the parameters to a method implemented in PHP. This method makes a call to the exec(), which runs the algorithm associated to the logic layer with the parameters received from the Interface. The query results are returned through a CSV-formatted text which is stored in a PHP variable. This variable is used to generate an XML file that is finally returned to the JavaScript and allows visualization of the resulting points on the map.

Finally, the database layer was handled by a library that allows to connect the logical layer with a PostgreSQL database. Using PostgreSQL, a developer can create and manipulate spatial databases by using the PostGIS library [13].

A. Our Algorithm

In this section, we propose the LOcation-based Skyline (LOS) algorithm. LOS is an algorithm based on nearest neighbor search for location-based Skyline query evaluation. LOS is presented in the Figure 2. LOS gets as input a set of spatial objects O, a preference function m defined on r attributes of the objects belonging to O, a given location l, and an R-Tree structure. We use an R-Tree to retrieve the objects in order of closeness using the nearest neighbor search [14].

Algorithm 1 LOS
Input: O: spatial objects; l: location; m: preference function; rt: R-Tree
Output: A set of Location-based Skylines, S

1. Compute the Skyline S_m in terms of the function m
2. While *rt.hasNext()* do
   If NN(l, rt) is not a Location-based Skyline object then discard it else add it to S
3. return S

Figure 2 LOS Algorithm

First, LOS computes the Skyline considering only the preference on r attributes of the objects belonging to O. In this step, LOS is able to compute a subset of the Skyline required to produce the location-based Skyline objects.

According to Theorem 1, the Skyline on a set of r attributes of A is part of the Location based Skyline on $A \cup \{a_{r+1}\}$. In our case, a_{r+1} represents the distance function. To complete the Skyline on $A \cup \{a_{r+1}\}$, LOS must scan the R-tree defined on a_{r+1} or distance function.

In the second step, LOS orderly scans the R-tree to retrieve the nearest neighbor of l. NN(l, r) is a function that returns the next nearest neighbor to l while there exist objects in the R-tree rt, this is, it produces the dataset sorted by distance in ascending order. Because we have the dataset sorted by distance from smallest to largest, we only need to compare the object with its predecessors. Its successors in the dataset can not dominate the object as they have worse (i.e., larger) distance values.

LOS uses the index rt to get the next nearest neighbor in each iteration. The next nearest neighbor is a location-based Skyline if it is in the Skyline S_m or none of the previous nearest neighbors dominates it. If none of the previous nearest neighbors dominates it, then it is added to the Skyline S_m. The Skyline is completed with these objects non-dominated and non-seen during the step 1.

IV. Experimental Study

In this section, we present an experimental study in order to analyze the performance of the algorithms implemented to evaluate location-based skyline queries.

A. Datasets and Queries.

The study was conducted on synthetic and real-world datasets. Synthetic datasets consist of relational tables populated with 10,000 and 50,000 objects. Each table contained an identifier, a location (latitude and longitude), and fourteen columns that represent the scores; values range from 0.0 to 1.0. A column may have duplicated values. Synthetic datasets were generated using the data analysis software of Microsoft Excel; data generated was uniformly distributed.

The real-world dataset was downloaded from the Zagat Web site. Since Zagat does not provide spatial information beyond the address of objects, it was necessary to use a system based on georeferencing [5] in order to obtain the latitude and longitude of each object. This georeferencing system takes an address and performs some calculations to approximate the latitude and longitude associated with a given address. Finally, we load the data into relational tables that contain 11,005 objects. Data may have duplicate values in several dimensions. A GIST index was created for each of the tables based on the latitude and longitude of each object. The distance function was indexed using the GIST index associated with each table.

We randomly generated up to ten queries characterized by the following properties: (a) only one table in the FROM clause; (b) the attributes in the preference function were chosen randomly among the attributes of the table, following a uniform distribution; (c) directives for each attribute of the preference function were selected randomly considering only maximizing and minimizing criteria; (d) the number of attributes of the preference function varied between 2 and 14.

B. Implementations and Metrics.

Data were stored in PostgreSQL 8.4.8 using the PostGIS library. LOS, BNL and SFS algorithms were implemented in C++ using g++ compiler version 4.5.2. We decided to implement SFS and BNL because they are well-known solutions that do not require the implementation of indexes. The algorithms based on composite indices depend on the user preferences. These algorithms work with composite indexes for more than one attribute. Thus, they must build the composite indices before evaluating a query. If a user changes preference, the indices must be pre-computed. Thus, our investigation does not consider algorithms based on

composite indices since they incur in much computational overhead when user changes preference. A requirement of our application is that users can change their criteria or have dynamic input. Thus, performance of other well-known indexed solutions that assume prior knowledge of the input may be affected by pre-computing indexes each time users change criteria. Particularly, our problem requires queries to be aware of user's location; an R-Tree index is built on locations of spatial objects.

We considered two metrics to analyze the performance of each of the algorithms implemented. These metrics are runtime and the number of dominance comparisons. The number of dominance comparisons is the number of the preference function evaluations performed by an algorithm. Runtime is the amount of time of the algorithm evaluation. This metric is measured by using the time command available in the distribution of Linux system used. Before running a query, we cleaned the tables and reloaded their data to avoid keeping data in main memory that could benefit any of the algorithms.

The experiments were evaluated on a HP dv6 machine equipped with 1 processor Intel Core i7 of 1.6GHz, 4 GB of memory and a 500GB disk running on Ubuntu 11.04.

C. *Skyline Size*

In Figure 3, we present the average size of the Skyline for synthetic data of 10,000 objects as the number of dimensions varies on the criteria of the Skyline. The horizontal axis represents the number of attributes per query, and the vertical axis the Skyline size. The Skyline size is the average size of three Skyline queries by each dimension. These experiments allowed us make some conclusions on performance of the algorithms in the following sections.

In Figure 3, we confirmed that increasing dimensions increases the Skyline size.

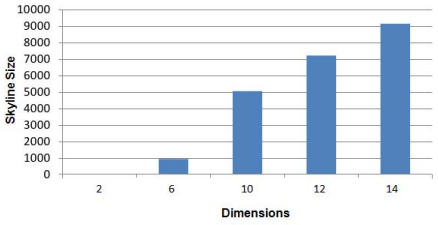

Figure 3 Skyline Size

Note that for queries of fourteen attributes, the average size of the Skyline is very close to the cardinality of the dataset. Skyline grows as the dimensions increase; the number of dominance comparisons also increases because an object may be compared against all Skyline objects. Therefore, the runtime is affected by a greater number of dominance comparisons.

D. *Performance of BNL, SFS and LOS.*

We study the performance of the BNL, SFS and LOS algorithms. We computed the average number of comparisons, and calculate runtime, for each of the three algorithms. This experiment consisted of executing of ten location-based Skyline queries of two, four and five attributes over the real-world dataset.

Figure 4 and Figure 5 show the average number of comparisons and the runtime required for each algorithm. We varied the number of dimensions on the real-world dataset. The horizontal axis represents the number of attributes per query, and the vertical axis the number of dominance comparisons or runtime of the algorithms.

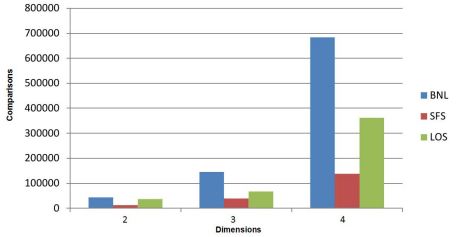

Figure 4 Number of Comparisons in Real-world Dataset

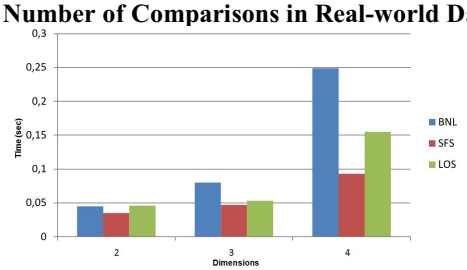

Figure 5 Time in Real-world Dataset

In Figure 4, we can observe that SFS performs the fewest number of dominance comparisons. This occurs due to the presence of duplicate data within the data set. LOS and SFS compare each object accessed with its predecessors. For LOS, predecessors are sorted by distance while SFS predecessors are sorted by an entropy function [3]. Furthermore because the order of predecessors are different for LOS and SFS, if there are duplicates, LOS requires pairwise comparisons in order to check the set of duplicated objects against the current accessed object. Thus, the runtime is higher because the number of comparisons increases.

Moreover, BNL does not sort data. Without knowledge of data ordering, BNL should check if the objects are dominated or dominate each object being accessed. Consequently, BNL requires a longer time since the number of dominance comparisons performed.

Finally, because of the Skyline size increases with number of dimensions, the number of dominance comparisons and runtime required by the algorithms increase as the number of dimensions increases.

We also study the performance of the BNL, SFS and LOS algorithms with a synthetic dataset. We executed three location-based Skyline queries of 2, 6, 10, 12, and 14 attributes in order to compare the three algorithms.

Figure 6 and Figure 7 show average number of comparisons and runtime required for each algorithm by varying the number of dimensions.

Figure 6 shows LOS performs the fewest number of dominance comparisons for high-dimensional queries; high dimensional Skyline queries have received the attention of some researchers [8]. Consequently, runtime of LOS is less than the SFS for queries of 10 and 12 attributes. The low number of dominance comparisons of LOS is because data

are non-duplicated and therefore does not perform additional checks for the case of equal-value attributes. With increasing the number of dimensions on the Skyline criteria, the cost of making comparisons may exceed the cost of scanning the R-Tree by distance. In [6] the worst case, complexity in dominance comparisons between objects is $O(n \log^{d-2} n)$ and $O(n)$ is the complexity of scanning the R-Tree index, where n is the data size and d the number of dimensions. Therefore, LOS has a lower runtime that SFS for queries of 12 and 14 attributes.

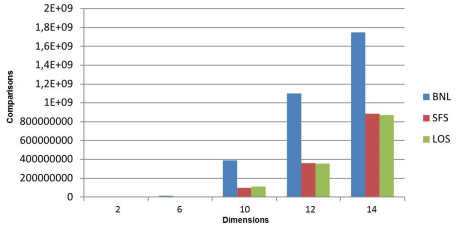

Figure 6 Number of Comparisons in Synthetic Dataset

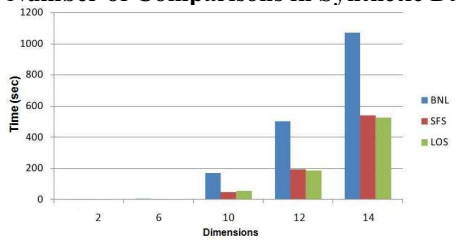

Figure 7 Time in Synthetic Dataset

Finally, BNL has the highest average number of dominance comparisons and runtime because it does not sort the data.

V. CONCLUSIONS

In this work, we build a Web tool called MapSkyline. It identifies location-based Skyline and displays the results on Google Maps. MapSkyline users enter their preferences filter the interesting objects. The filter choice will facilitate effective decision-making, whereas the number of objects that are part of the result is greatly reduced in most cases.

MapSkyline uses three evaluation algorithms for location-based Skyline queries. Two of these algorithms are BNL [1] and SFS [3]. The third algorithm was proposed as an additional solution to the evaluation of such queries. This algorithm called LOS, uses the nearest neighbor technique and is based on the property that the Skyline of a set of attributes A can be contained in the Skyline of set B, where B is the union of A with an additional attribute.

Moreover, the results of experimental study show that in data sets with duplicates, SFS performs better than LOS. LOS execution requires more time due to data duplication that causes some additional comparisons. Conversely, if the dataset contains non-duplicated data and have a large number of attributes in the Skyline criteria, LOS performs better than SFS.

Additionally, in the case for non-duplicated datasets and large number of attributes for LOS; the experiments revealed that the time consumed by the index scan may not be significant when compared to the time spent in making the comparison between objects.

Future work includes extending our experimental evaluation. Data will be partitioned in separate tables and consider the join over the tables before computing the Skyline. This is interesting, as BNL and SFS need objects in a single table, while LOS can work with separate tables. Also, we will study the optimal order to evaluate the attributes for the skyline. Furthermore, we perform a user study to determine the applicability and usability of MapSkyline in real-world situations.

REFERENCES

[1] Börzsönyi, S., Kossmann, D. and Stocker, K. "The skyline operator". Proceedings of International Conference on Data Engineering, (2001), 421-430.

[2] Balke, W-T., Güntzer, U., and Zheng, J. (2004). "Efficient Distributed Skylining for Web Information Systems". Proceeding of the International Conference on Extending Database Technology (EDBT), pp. 256-273.

[3] Chomicky, J., Godfrey , P. Gryz, J. and Liang, D. "Skyline with Presorting". Proceedings of 19th International Conference on Data Engineering, (Mar. 2003).

[4] Estrin, D. "Participatory sensing: applications and architecture". Internet Computing, IEEE , vol.14, no.1, pp.12-14, Jan. 2010.

[5] Geo Coder US. Geo coder us. Avalaible at http://rpc.geocoder.us.

[6] Godfrey, P., Shipley, R., and Gryz, J. "Maximal Vector Computation in Large Data Sets". Proceeding of the Conference on Very Large Data Bases (VLDB), (2005), pp. 229-240

[7] Google. Google maps api. Avalaible at http://code.google.com/apis/maps/index.html.

[8] Jin, W., and Tung, A., Ester, M., and Han, J. "On Efficient Processing of Subspace Skyline Queries on High Dimensional Data". Proceedings of SSDBM (2007), p.12

[9] Kodama, K., Lijima, Y., Guo, X., and Ishikawa, Y. (2009). "Skyline Queries Based on User Locations and Preferences for Making Location-Based Recommendations". Proceeding of the 2009 International Workshop on Location Based Social Networks, pp. 9-16

[10] Kossmann, D., Ramsak, F., and Rost, S. (2002). "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries". Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), pp. 275-286

[11] Lo, E., Yip, K., Lin, K-I., and Cheung, D. (2006). "Progressive Skylining over Web-Accessible Database". Jornal of Data and Knowledge Engineering, pp. 122-147.

[12] Papadias, D., Tao, Y., Fu, G., and Seeger, B. (2005). "Progressive Skyline computation in database systems". ACM Transactions Database Systems, 30 (1), pp. 41-82.

[13] PostGIS. About postgis. Avalaible at http://postgis.refractions.net.

[14] Roussopoulos, N., Kelley, S., and Vincent, F. Nearest neighbor queries. Proceedings of International ACM SIGMOD in Data Management, 71-79, 1995.

[15] Scott, M. Programming Language Pragmatics. Morgan Kaufmann, 1999.

[16] Sharifzadeh, M., and Shahabi, C., (2006). "The Spatial Skyline Queries". Proceedings of the 32nd international conference on Very Large Data Bases (VLDB), pp. 751-762

[17] Tan, K.-L., Eng, P.-K., and Ooi, B. C. (2001). "Efficient progressive Skyline computation". Proceeding of the Conference on Very Large Data Bases (VLDB), pp. 301-310.