Efficient Continuous Skyline Query Processing Scheme over Large Dynamic Data Sets

He Li and Jaesoo Yoo

Performing continuous skyline queries of dynamic data sets is now more challenging as the sizes of data sets increase and as they become more volatile due to the increase in dynamic updates. Although previous work proposed support for such queries, their efficiency was restricted to small data sets or uniformly distributed data sets. In a production database with many concurrent queries, the execution of continuous skyline queries impacts query performance due to update requirements to acquire exclusive locks, possibly blocking other query threads. Thus, the computational costs increase. In order to minimize computational requirements, we propose a method based on a multi-layer grid structure. First, relational data object, elements of an initial data set, are processed to obtain the corresponding multi-layer grid structure and the skyline influence regions over the data. Then, the dynamic data are processed only when they are identified within the skyline influence regions. Therefore, a large amount of computation can be pruned by adopting the proposed multi-layer grid structure. Using a variety of datasets, the performance evaluation confirms the efficiency of the proposed method.

Keywords: Continuous skyline query, Query processing, Multi-dimension data, Large dynamic data, Skewed data.

I. Introduction

Recently, interest in skyline query processing has significantly increased since these queries can be used in many applications such as recommendations, queries, and data analysis in large database systems. Most of the existing research has focused on skyline query computations over static data sets [1]–[7]. However, since the data sets change dynamically in most production environments, it is necessary to process continuous skyline queries over dynamic data sets.

Example 1: A common example that illustrates the use of a skyline query is to assist a tourist to find a set of interesting hotels from a hotel database. [1] Studied the skyline query in the context of databases and proposed an SQL like syntax for a skyline query. Figure 1 shows a set of hotels with two attributes, the distance (x-axis) and the price (y-axis). We assume that smaller values on all dimensions are preferred. Subsequently, we show a simple example of a skyline query that identifies hotels with a less expensive price and are closer to 'BEIJING.'

```
SELECT *
FROM hotel
WHERE CITY = 'BEIJING'
SKYLINE OF price MIN, distance MIN;
```

The query results from the skyline query are shown in Fig. 1 (a), consisting of hotels A, B, C, D, and E, at time T_1. When the data in Fig. 1(a) is updated, the skyline query is executed again, obtaining different query results, for example, when the data point D expires at time T_2, the new output of skyline query is A, B, C, G, and E (Fig. 1(b)). Since G is adjacent to point D (in distance and price), it has to be added to the skyline results and D must be removed. Additionally, when a new data point K arrives, the skyline query is executed again. Since K dominates A, the query output of the new skyline

Manuscript received Jan. 11, 2016; accepted July 20, 2016.

He Li (heli@xidian.edu.cn) is with the School of Software, Xidian University, Xi'an, China.
Jaesoo Yoo (corresponding author, yjs@chungbuk.ac.kr) is with the School of Information and Communication Engineerin, Chungbuk National University, Cheongju, Rep.of Korea.

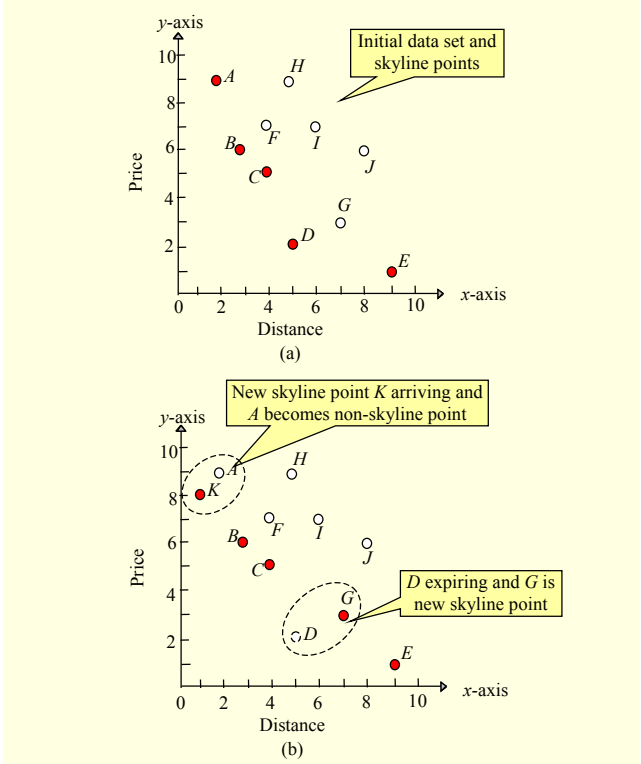

Fig. 1. Example of a continuous skyline query: (a) the initial data set and its skyline result at time T_1 and (b) the dynamic data set and its skyline result at time T_2.

result is *K*, *B*, *C*, *G*, and *E*. The skyline query of static data set is evaluated only once, while the continuous skyline query is executed continuously as updates are made against the base Relational data. Since the execution of any INSERT, DELETE, or UPDATE statement will modify rows of the base tables in the database, the skyline query must be executed after execution of each of these SQL statements makes the changes persistent (SQL COMMIT). As described in [1], the processing of a skyline database query is very expensive, especially for large dynamic data sets.

Some recent techniques have been proposed to compute skyline queries over sliding window data streams [8]–[11] and moving objects [12], [13]. The skyline computation of sliding window streams assumes that the lifespan of the data is pre-defined. However, the lifespan of a data tuple in most real applications is non-deterministic. The continuous skyline query method for moving objects assumes that each object has a single dynamic attribute (the location) and one or more static attributes. In this paper, we focus on processing continuous skyline queries over dynamic data sets without constraints. The number of dynamic attributes of each data tuple is arbitrary. The dynamic data is defined as a data tuple with an arrival time and expiration time. The arrival time, expiration time, and lifespan of each data tuple are random. Although the grid structure has been used to monitor continuous skyline results in [14], [15], the fixed cell size of the grid structure introduces restrictions such that the fixed cell size is suitable only for uniformly distributed datasets and small-to-medium size datasets.

It is difficult to fix the cell size of a grid structure because of the non-deterministic nature of the data distribution. In order to resolve this issue, we adopt a method that is based on the use of a multi-layer grid structure to filter out unnecessary data while executing continuous skyline queries over dynamic data sets. The data objects of an initial data set are processed to obtain a corresponding multi-layer grid structure and generate the skyline influence regions over the structure. Subsequently, the dynamic data are processed only when they are identified within the skyline influence regions. The skyline results and skyline influence regions are updated based on the processing of continuous skyline queries. Since a large unnecessary instruction path length can be pruned during the processing of continuous skyline queries, the proposed method is suitable for processing large data sets that are dynamic or skewed.

The remainder of this paper is structured as follows. Section II reviews related works regarding the existing skyline query processing scheme. Section III presents the preliminaries of this paper. In Section IV, we describe the proposed multi-layer grid structure based method .that efficiently processes a skyline query Section V discusses the performance evaluation of the proposed method relative to existing methods. Finally, Section VI concludes the paper.

II. Related Work

The skyline algorithm was first proposed by Kung and others as described in [16]. The relational database skyline operation was first introduced in [1] and two methods, Block Nested Loop (BNL), and Divide and Conquer were proposed. After that, the sort-filter-skyline method based on BNL was proposed in [5]. The skyline algorithm was given the name 'branch and bound skyline,' and was proposed in [3], [4]. Recently, some sub-space skyline queries were proposed to provide users with multiple skylines with different dimensions of a data set [17], [18]. They created a sky-cube structure and could process skyline queries with arbitrary dimensions over the data set. However, none of these methods are suitable for processing large dynamic data sets.

In order to process dynamic data sets, several continuous skyline computations were recently proposed. The *n*-of-*N* skyline query processing method proposed in [8] was used to capture skyline results from the most recent *N* elements of a relational data object-based stream. Some skyline query algorithms that generate a sliding window data stream were

proposed to efficiently update the continuous time-interval skyline queries over a given data stream [9], [10], [19]. In this case, they can only be used to process data streams that possess a fixed pre-defined input stream, and an expiration time. These queries are not appropriate for processing dynamic data sets that randomly change, and the lifespan of each data object is non-deterministic. A continuous reverse skyline query processing method that moves objects across environments was proposed in [20], and used as a verification technique to guarantee the results of a reverse skyline query. In [21], an efficient skyline search engine for continuous skyline computations was proposed. It described a continuous skyline computation system in an interactive environment. The update of skyline results is conducted only when data update requests originate from a service provider. The updates of continuous skyline results were executed with the adoption of pre-computed candidate skyline results. Creating the skyline set required an extensive amount of computational resources, which were performed within a separate procedure. However, the proposed method focuses on real-time environments, in which the skyline computations are performed continuously and synchronized with each data update.

The grid structure has been used to monitor continuous skyline results in [14], [15]. However, the fixed cell size of the grid structure is not suitable for skewed data sets. This is because a small number of cells can contain many data points, as well as a large number of unnecessary data points that cannot be pruned during continuous skyline query processing. Therefore, the concept of two layer grids was introduced in our previous paper [22], in which a second grid layer was created over the skewed data cells of the first grid layer. The two grid layer model assumes that the data distribution has not changed. Since the data distribution of large dynamic data set is not always fixed and the data set can become skewed, an extended dynamic multiple layer grid method is proposed in this paper. The results in [23] also proposed a grid-based skyline processing method in distributed environments. However, the researchers focused on the reduction of unnecessary data transmissions among distributed data servers connected by a network. In this paper, we focus on how to prune unnecessary computational path length during continuous skyline query processing over large dynamic data sets.

III. Preliminaries

1. Domination Relationships

Before describing the proposed method in detail, we first define some domination relationships. Given a n sized data set $D = \{O_1, O_2, \dots, O_n\}$ with d dimensions $\{d_1, d_2, \dots, d_d\}$, the skyline query retrieves all objects O_i such that O_i is not dominated by another object O_j in D. Without loss of generality, we assume that the data value of each dimension is greater than or equal to zero and the minimum value (for example, the low price and short distance for hotel selection) is preferable. In order to easily explain the proposed method, we assume that, for the remainder of the paper, the dimension of the data set is two. The data domination relationship is defined in Definition 1.

Definition 1. data domination: Given two d-dimensional data points O_1 and O_2, O_i represents the value of the ith ($i = 1, \dots, d$) dimension. Here the value of d is 2. If $\forall i, O_1.i \leq O_2.i$, and $\exists j, O_1.j < O_2.j$ ($i, j \leqslant d$), then we say that O_1 dominates O_2.

The key operation of skyline query processing is dominant comparison of the data.

Definition 2. grid structure: Given a d-dimensional data set D ($O_i \in D$), we adopt a grid structure G to manage data points O_i, in a given data set D. To create the grid structure G, the entire data space is partitioned into equal-sized cells at each dimension according to the given data set D (for example, the dotted line grid in Fig. 2. illustrates a 2-dimensional data space.). According to the attribute values of a given data point O_i, it is easy to find the cell within the grid G at which O_i is located (for example, Fig. 2 shows the given data points within a grid structure).

Cell domination is similar to data domination. In order to introduce cell domination, a d-dimensional cell in the grid can be represented by its 2^d vertices. In these vertices, there exists one data point which is not dominated by any data point in this cell. We define this vertex as the lower left corner coordinate of the cell. Conversely, there exists one data point which cannot dominate any other data point in the cell. We define this vertex as the upper right corner coordinate of the cell. Each corresponding cell has a left lower corner coordinate value and a right upper corner coordinate value in the grid. The coordinate value of the left lower corner and the right upper corner can be represented by a data point. The cell domination is based on the domination of coordinate values.

Definition 3. cell C_i dominates Cell C_j: Given two d-dimensional cells C_i and C_j ($i, j \leqslant d$), if the coordinate value of the right upper corner of C_i dominates the coordinate value of the left lower corner of C_j, then we say that C_i dominates C_j.

Definition 4. cell C_i is incomparable with Cell C_j: Given two d-dimensional cells C_i and C_j ($i, j \leqslant d$), if the coordinate value of the right upper corner of C_i does not dominate the coordinate value of the left lower corner of C_j, and vice versa, then we say that C_i is incomparable with C_j.

Definition 5. data point O_i dominates Cell C_j: Given a d-dimensional data point O_i ($i \leqslant n$), and a d-dimensional cell C_j ($j \leqslant d$), if the coordinate value of O_i dominates the coordinate

value of the left lower corner of C_j, then we say that O_i dominates C_j.

Definition 6. cell C_j contains data point O_i: Given a d-dimensional data point $O_i (i \leqslant n)$, and a d-dimensional cell C_j $(j \leqslant d)$, if the coordinate value of each dimension of O_i dominates the coordinate value of each dimension of the right upper corner of C_j and the coordinate value of each dimension of O_i is dominated by the coordinate value of each dimension of the left lower corner of C_j, then we say that C_j contains O_i.

Lemma 1: The data point p_i within a dominated cell C_i cannot be skyline data.

Proof: If a cell C_i is dominated then the data point O_i of its left-lower corner coordinate is dominated by existing skyline results. Since the data point O_i of the left-lower corner coordinate of cell C_i is already dominated by existing skyline data points, all data points contained by cell C_i are dominated by the point of O_i of the left-lower corner coordinate of cell C_i. Hence, all data points within a dominated cell C_i cannot be skyline data. ■

Definition 7. skyline influence region: The skyline influence region is composed of the cells that are not dominated by existing skyline results. The data points that are updated within the skyline influence region can affect the skyline results. The rest of the cells represent non-skyline region. As shown in Fig. 2, the grey regions represent the skyline influence region (the minimum value of each dimension is preferable).

Within the grid structure, all cells are marked with integer numbers (0 or 1), such that a value of '1' represents a cell resides in skyline influence region, and a value of '0' represents a cell resides in non-skyline influence region. The skyline influence region represents the cells that are not dominated by the existing skyline data. The skyline influence regions are changed according to the change of the current skyline data. The expiration of the existing skyline data can cause a non-skyline influence region to become a skyline influence region. The new arriving skyline data can cause the skyline influence region to become a non-skyline influence region. We assert that the data that falls into the non-skyline influence regions cannot be skyline data.

2. Problem Statement

For continuous skyline query processing over dynamic data sets, the skyline query has to be performed continuously with an update of the data set Therefore, the key problem of continuous skyline query processing is how to prune the unnecessary processing when an update of the data does not affect the skyline result.

We adopt a grid structure to manage the multi-dimension data and utilize it to identify un-promising data points (the data points were dominated and cannot be skyline result) during continuous skyline query processing. To illustrate this, Fig. 2(a) illustrates a data set with two dimensions and its corresponding grid structure. Since this paper focuses on continuous skyline query execution over dynamic data sets, the query type is not changed during continuous processing.

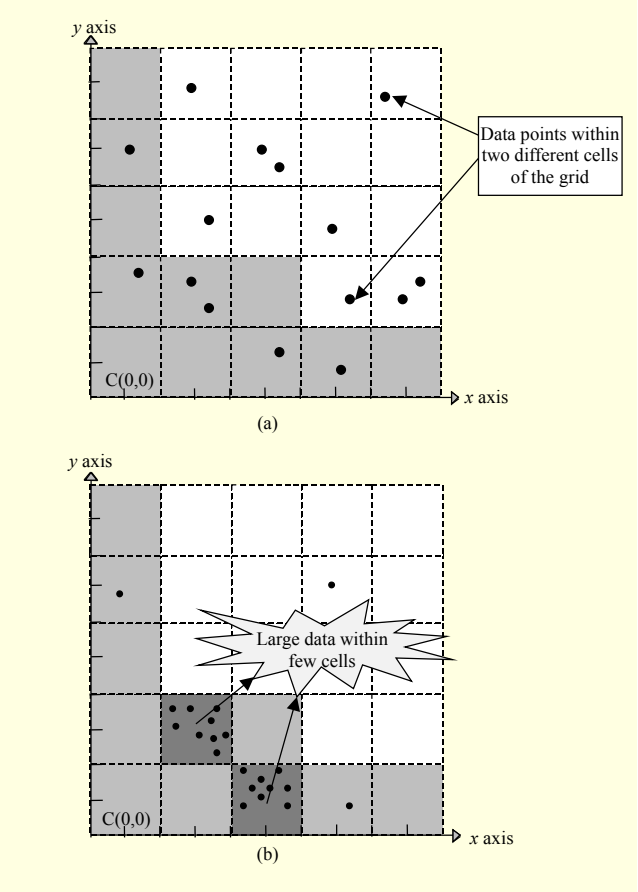

Fig. 2. Skyline processing over uniform data set and skewed data set: (a) uniform data set and (b) skewed data set.

The continuous skyline queries are executed only when the data object is updated within the skyline influence regions. That is to say, the new rows are retrieved the data will not be processed if that data is outside the skyline influence regions. Additionally, when existing data points expire, the continuous skyline query is re-executed only if the expiring data points are skyline data. Therefore, some un-promising data can be eliminated. This can eliminate unnecessary data processing without affecting the accuracy of the skyline results during continuous skyline query processing over dynamic data sets.

Definition 8. high data-density cell: If the number of data points within Cell C_i is larger than a predefined threshold value t, we define Cell C_i as a high data-density cell. For instance, the dark-grey cells in Fig. 2(b) which contain a large number of data points are high data-density cells.

If we select a small cell size of the grid structure, a large number of cells will be generated and many of these cells will be empty. In contrast, if we select a large cell size for the grid structure, one cell will contain a large number of data objects. The unnecessary data processing cannot be pruned by using the existing grid structures. If the grid structure is composed of a single large cell, the skyline query processing will degenerate to BNL [1]. [14], [15] has confirmed that the BNL method is not a suitable technique to process large dynamic data sets.

IV. Proposed Method

1. Multi-layer Grids Structure

In this section, our goal is to propose a multi-layer grid structure to improve the efficiency of performing continuous skyline queries over large dynamic data sets.

Definition 9. multi-layer grids structure: In the case of a high data-density cell, the cell is partitioned into a multi-layer grid structure. The processing of the $(i + 1)$th grid layer is identical to the processing of the ith grid layer. The partitioning of the multi-layer grid G^i iterates at each layer until the number of data points included in a cell is less than the predefined threshold value t.

Figure 3 illustrates a multi-layer grid structure, derived from an n sized data set $D = \{O_1, O_2, \ldots, O_n\}$ with d dimensions. A tuple v <OID, v_1, v_2, … , v_d > represents a data object O_i, such that 'OID' denotes the id of a data object, and is computed by adopting a hash function, $v_{i\,(1<i<d)}$ to represent the composite of the values of the d dimensional Data structure OL is a list whose elements are data objects. Data structure SL stores skyline results which are also data objects. The data objects are mapped to a multi-layer grid structure. The IR data structure stores skyline influence regions which are the cells that are not dominated by the skyline data.

2. Algorithms

The proposed multi-layer grid model is based on the continuous skyline query processing scheme and consists of two phases. In the first phase, the initial data are processed to create a corresponding multi-layer grid structure and its skyline influence regions. In the second phase, the dynamic data are processed only if they are identified in the skyline influence regions. The skyline result and skyline influence regions are updated according to the processing of rows retrieved by the continuous skyline queries. Figure 4 shows the pseudocode of the SL_Init algorithm to process the initial data set. During the computing of the initial skyline results and skyline influence regions, we use a queue data structure to

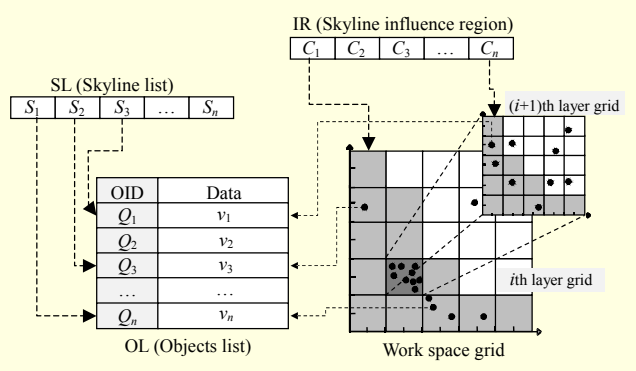

Fig. 3. Data organization.

order the storage of the data. The algorithm begins by inserting the left-bottom cell $C^1(0, 0, \ldots, 0)$ of the grid into Queue^i. Then, the data is processed with each queue operation. When cell $C^1(0, 0, \ldots, 0)$ is popped from the queue, the skyline data within it are evaluated and cell $C^1(0, 0, \ldots, 0)$ is marked as a skyline influence region if it is not dominated by skyline data. Subsequently, its neighboring cells have to be processed, such as $C^i(x, y + 1, \ldots, d)$, $C^i(x + 1, y, \ldots, d)$, … , and $C^i(x, y, \ldots, d + 1)$. If they have not been processed and they are not dominated by existing skyline data, then C^i is pushed onto the queue. Note that, when the number of data objects within cell $C^i(x, y, \ldots, d)$ is greater than that of the pre-defined threshold value t, the processing of the upper-layer grid based on $C^i(x, y, \ldots, d)$ will be triggered, and this processing is identical with the processing of the lower-layer grid. The algorithm terminates when the queue becomes empty. Upon exiting the procedure, we obtain the skyline results and the skyline influence regions while ignoring portions of the entire initial data set. As discussed in Lemma 1, the data points within the cells that are dominated by existing skyline points can be filtered for the skyline query processing.

Lemma 2: At the end of the initialization phase, the SL data structure (array) includes all of the skyline data. Further, the cells included within the IR data structure may contain future skyline data.

Proof: Skyline query processing is such that the original point O_i has the strongest domination capability. Since the SL_Init algorithm begins by checking the origin cell $C^i(0, 0, \ldots, 0)$, and a Queue^i is used to expand remaining cells, this can guarantee that all cells that are not dominated by existing skyline data points must be checked. Therefore, all skyline data points and all the cells that may contain future skyline data points are checked. If we assume that some skyline points exist in the cells that are dominated by existing skyline points, this contradicts the fact proved by Lemma 1. The contradiction implies that all promising data points that will be skyline data are checked. Hence, we can prove that the SL data structure

Algorithm. SL_Init

Input:
initial data set D and its corresponding cells C
Output:
SL (skyline list), IR (skyline influence region)
Description:
1. SL = NULL; Queue^i = NULL; int t;
2. insert $C^i(x, y, \dots, d)$ into Queue^i;
   // $x = y = 0 = 0$, $I = 1$, $C^1(0, 0, \dots, 0)$ represents the left-bottom cell of the grid.
3. **while** (Queue^i! = NULL) **do**
4. remove $C^i(x, y, \dots, d)$ from Queue^i;
5. **if** $0 \leqq$ sizeof $(C^i(x, y, \dots, d)) \leqq t$ **then**
6. adopt BNL to compute skyline SL in cell $C^i(x, y, \dots, d)$, $C^i(x, y, \dots, d) = 1$;
7. insert $C^i(x, y, \dots, d)$ into IR;
8. **if** $C^i(x, y + 1, \dots, d)$, $C^i(x + 1, y, \dots, d)$, … ,or $C^i(x, y, \dots, d + 1)! = 1$ && $(x + 1), (y + 1), \dots, (d + 1) <$ Max(ith-dimension) **then**
9. insert $C^i(x, y + 1, \dots, d)$, … , or $C^i(x, y, \dots, d + 1)$ into Queue^i;
10. **else then**
11. a $(i + 1)$th layer grid based on $C^i(x, y, \dots, d)$ is generated, the $(i + 1)$th layer grid is processed in the same way as the ith layer grid (repeat lines 1~12);
12. end **while**
13. end

Fig. 4. Initialization phase of the proposed method.

Algorithm. SL_Add

Input:
a new data object a
Output:
IR, SL
Description:
1. check the corresponding cell $C^i(x, y, \dots, d)$ of a;
2. **if** $C^i(x, y, \dots, d) = 1$ **then**
3. //re-compute skyline SL from $C^i(x, y, \dots, d)$;
4. insert $C^i(x, y, \dots, d)$ into Queue^i;
5. (the procedure is same as lines 3~12 of SL_Init);
6. **else** end;

Algorithm. SL_Del

Input:
an expiring data object e
Output:
IR, SL
Description:
1. **if** e is a skyline data, **then**
   check the corresponding cell $C^i(x, y, \dots, d)$ of e;
2. remove e from SL and OL;
3. insert $C^i(x, y, \dots, d)$ into Queue^i;
4. (the procedure is same as lines 3~12 of SL_Init);
5. **else** remove e from OL;
6. end;

Fig. 5. Maintenance phase of the proposed method.

includes only skyline data points. All the cells that are not dominated by existing skyline data points are added to the IR data structure (array), and, according to Lemma 1, the remaining cells cannot contain any skyline data points. Therefore, these conditions are sufficient to prove the correctness of the initial skyline evaluation initialization phase. ■

At the same time, the skyline results and skyline influence regions are updated. The SL_Add algorithm to process new row columns and the SL_Del algorithm to process the expiring data are shown in Fig. 5.

The SL_Init algorithm shows that when a new data point arrives, the SL_Add algorithm first examines the corresponding cell $C^i(x, y, \dots, d)$ of the arriving data. If $C^i(x, y, \dots, d)$ has multiple layer grids, then further processing is required only if $C^i(x, y, \dots, d)$ and the corresponding cells in the multiple layer grids are skyline influence regions. By performing this additional processing, more unpromising data points can be filtered out without processing the continuous skyline queries over highly dynamic data sets. When an existing data point expires, it will be deleted from the OL and the skyline results must be re-computed. Note that, if the expiring data are not skyline data points, the expiration of the data does not affect the skyline results or the skyline influence regions. Then, the continuous skyline queries are processed only if the expiring data are skyline points. As shown in Fig. 5, the SL_Del algorithm updates the skyline results and the skyline influence regions when the expiring data point represents skyline data. The processing of the SL_Del algorithm is similar to that of the SL_Init algorithm. The difference is that the SL_Del algorithm begins from the corresponding cell of the expiring data object.

Next we prove the correctness of the continuous skyline evaluation within the maintenance phase.

Lemma 3: If a data point p_i arrives outside the skyline influence region IR, the arrival of p_i does not affect the existing skyline results.

Proof: According to Lemma 1, any data point p within the dominated cells cannot be skyline data. For any cell C^i which is not contained within the skyline influence regions IR, it is dominated by existing skyline data points. Therefore, only the new data point p_i falls into the cell that is included in the skyline influence region, and the data point will become new skyline data. Then, it is necessary to perform skyline evaluation on only those data points that arrive in the skyline influence region. ■

Lemma 4: The skyline influence region IR is updated only when the skyline data set SL is updated.

Proof: To prove the correctness of this, we first show that, in the first case, the SL is updated when a new skyline data point p_s arrives. According to Lemma 3, a new skyline data point p_s is in the skyline influence region IR. Then, the arrival of new skyline point p_s will dominate the cells in the skyline influence region IR. In the other case, the SL is updated when an existing skyline data point p_s expires. The expiration of an existing skyline data point will result in a change of state of some dominated cells to un-dominated cells. The un-dominated cells will form a new skyline influence region. Therefore, the skyline influence region IR must be updated when the skyline data set SL is updated. However, skyline data points exist outside of the skyline influence region IR, and this contradicts Lemma 3. ■

3. Analysis

We observe that the number of comparisons is the most

important factor that affects the computation time to support continuous skyline query processing. For large dynamic data sets, the arrival of the data and the expiration of the data affects the skyline computation costs. Therefore, the objective of continuous skyline computation is how to filter out the unnecessary path length overhead.

Since the initial skyline computation time is obtained by executing skyline queries over an initial static data set, the computation costs can be considered as a constant value T_{int}. For the maintenance phase, the data arrival and expiration will require the re-computation of the current skyline. We assume that a skyline list (*SL*) is used to store the current skyline data, and we then use *s.SL* to denote the size of *SL*. Consider that the skyline size *s.SL* varies with time. Let *s.SL* be the maximum size, that is, *s.SL* is the largest number of skyline data entries at any time. The computation cost of processing the arrival of each single data point is determined by *s.SL*, that is, the performance is $O(s.SL)$. The continuous processing cost of the arriving data occurs as the new data enters the skyline influence region (*IR*). Similarly, *s.IR* is used to denote the area of the skyline influence region. We assume that the probability that the new data entering the skyline influence region of GICSC is P_G, which is determined by *s.IR*. A larger *s.IR* will result in a larger value of P_G and more computation cost. We assume that when there are *N* arriving data points, the computation cost of processing the arriving data of GICSC is the following (1):

$$f(N) = O(N \cdot P_G \cdot O(s.SL)). \quad (1)$$

However, for the proposed method, the probability of the new data entering the skyline influence region is P_M, which is determined by *s.IR*. Because the skyline influence region of the proposed method is created on multiple layer grids, the value of *s.IR* of the proposed method is much smaller than that of GICSC. Therefore, the probability P_M of the proposed method is smaller than P_G. As a result, we can conclude that $O(N \cdot P_M \cdot O(s.SL))$ is smaller than or equal to $O(N \cdot P_G \cdot O(s.SL))$.

The costs of processing single data expiration is primarily determined by the cost of re-evaluating the remaining data set. When the existing skyline data expire, the data points dominated by the expired data have the opportunity to become new skylines. The cost of processing the data points dominated by the expired skyline data is $O((s.D)/(s.CL) \cdot c)$, such that *s.D* is the size of the entire data set, *s.CL* is the number of cells, *s.D*/*s.CL* is the average size of data within each cell, and *c* is the number of cells that must be processed. $O(s.SL)$ is the cost of checking whether or not the expired point is a skyline point. We assume that there are *N* expired data points and the computation time of GICSC is the following (2):

$$f(N) = N \cdot \left(O(s.SL) + O\left(\frac{s.D}{s.CL} \cdot c \right) \right). \quad (2)$$

The proposed method utilizes multiple layer grids and the processing costs are computed by the following (3):

$$f(N) = N \cdot \left(\mathrm{O}(s.SL) + O\left(\frac{s.D}{(s.CL)^m} \cdot (n) + \frac{s.D}{s.CL} \cdot (c-n) \right) \right). \quad (3)$$

$O((s.D)/(s.CL)^m \cdot (n))$ is the processing cost over multiple layer grids, *n* is the number of cells that have multiple layer grids, and *m* is the number of layers. The processing costs of the proposed method and GICSC are similar over the uniform data set since the dynamic data is uniformly distributed within each cell. That is, $O((s.D)/(s.CL) \cdot c)$ is similar to $O((s.D)/(s.CL)^m \cdot (n) + (s.D)/(s.CL) \cdot (c-n))$. However, the advantage of the proposed method is obvious for skewed data sets.

V. Performance Evaluation

In this section, we evaluate the efficiency of the proposed multi-layer grids method. We compare it with the GICSC [14], [15] and the QSkycube [20], [21] methods. Our goal is to evaluate the performance of existing methods relative to the performance of the proposed method for continuous skyline queries over dynamic data sets.

1. Experimental Environments

There are three critical types of data distributions that stress the effectiveness of skyline query processing proposed in [1]. The correlated data demonstrate that a multi-dimension data tuple that is good in one dimension is also good in the other dimensions. By contrast, the anti-correlated data represent the data points that are good in one dimension and bad in one or all of the other dimensions. The independent data is that data that are distributed arbitrarily within the data space. The synthetic data sets used in the experiments consist of two parts, the initial static data part, and the dynamic data part. The initial static data sets have been generated as described in [1]. The size of the data set ranges from 1,000 to 250,000 and the number of the dimensions varies from 2 to 5. The dynamic data part is simulated by changing the attribute values of the initial static data. The existing data within the initial static data set are expired randomly and the new generated data are inserted to replace the expired data. The ratios of the expiring data and new incoming data are assigned values of 50% and 50%, respectively. The new data are generated according to the data types of the initial data itself. The experiments are executed by

computing the initial skyline data over the initial static dataset and continuously processing the dynamic data. The algorithm terminates when all of the data in the data set have been processed.

The real data set used in the experiments is the batting information of the hustle score from Lahman's baseball databases from the year of 1871 to 2009. The data set is dynamic since the information of each baseball player is updated when the new game is played. The data set contains 92,706 data objects.

For synthetic data sets, the continuous skyline query is defined to query data that is not continuously dominated by any other data and the small value of each data attribute is preferred in the dominancy comparison. For real data sets, the continuous skyline query is defined to evaluate excellent players with high scores during the baseball game and a high value of each dimension is preferred; the null value in the data set is set to 0.

2. Experimental Results

For grid based methods, the cell size is a key parameter that affects the performance. In the first experiment, we identify the effect of cell size. The number of the dimensions is 2 and the data size is 25,000. The number of cells at each dimension is varied from 1 to 100. Then, the cell size is varied from 1 to 1/100. The performance is measured by elapsed CPU time. Figure 6 shows that the number of cells in each dimension significantly affects the performance. The *x*-axis presents the number of cells at each dimension and the *y*-axis is the computation time. From these results, we can see that when the number of cells in each dimension is larger than a certain threshold (20), the execution time increases slowly. This is because when the number of cells in each dimension is small, the skyline influence region is coarse, and more irrelevant data points fall into the skyline influence region that require significant and unnecessary computations. With the increase in the number of the cells in each dimension, the skyline influence region becomes more accurate and more irrelevant data points can be filtered out, thus improving performance. However, a large number of additional cells are generated, and processing these cells increases the computation costs and memory requirements. In the following experiments, we set the default number of cells in each dimension to 8. Without loss of generality, the cell size of the GICSC is the same as that of the proposed method.

The skewed data are obtained by adjusting the data distribution of the data set within the entire data space. The skew rate is defined as the percentage of the data distribution region relative to the entire data space. In this experiment, the 2-dimension anti-correlated data type is used and the data size

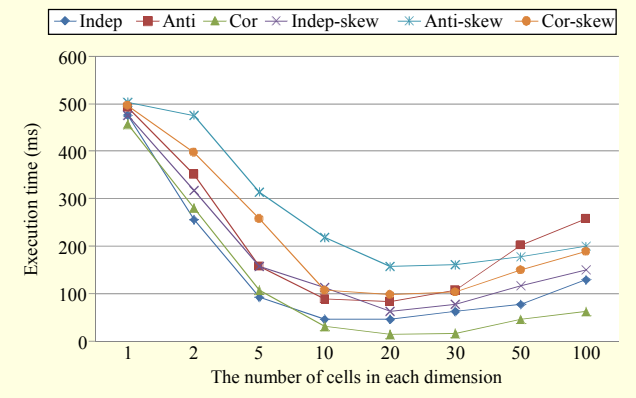

Fig. 6. Effect of the cell size.

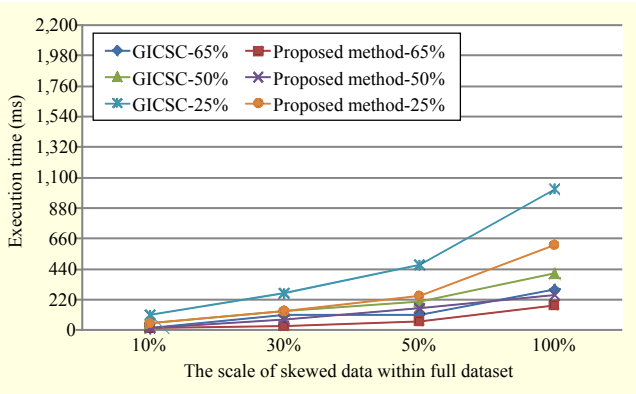

Fig. 7. Execution times according to different skew ratios.

is 100,000. The skew rates vary from 65% to 25%. The *x*-axis of the result in Fig. 7 is the scale of the skewed data. 10% represents a data set such that 10% of the data are skewed, and 90% of the data satisfy a uniform distribution. The ratios are varied from 10% to 100%. Since the multi-layer grids can filter out more un-promising data, the proposed method is more efficient than GICSC. In the following experiments, we use the skewed data set with a skew rate of 25%.

We examine the performance of GICSC, QSkycube, and the proposed method for continuous skyline computation over dynamic data sets. This experiment varies the size of the synthetic data sets from 1,000 to 250,000 and real data sets from 10,000 to 92,706. The dimension is 2. Figures 8(a), (b), and (c) show the execution times of the continuous skyline queries computation with respect to the three different synthetic data sets. From the results, we can observe that QSkyube is not suitable for dynamic data sets. It is designed for efficiently processing the subspace skyline computation when the data is not changed. For skewed data sets, the proposed method is more efficient than GICSC since the additional unnecessary data processing can be pruned. Figure 8(d) shows the result of continuous skyline queries over the real data set; the advantage of the proposed method is also obvious.

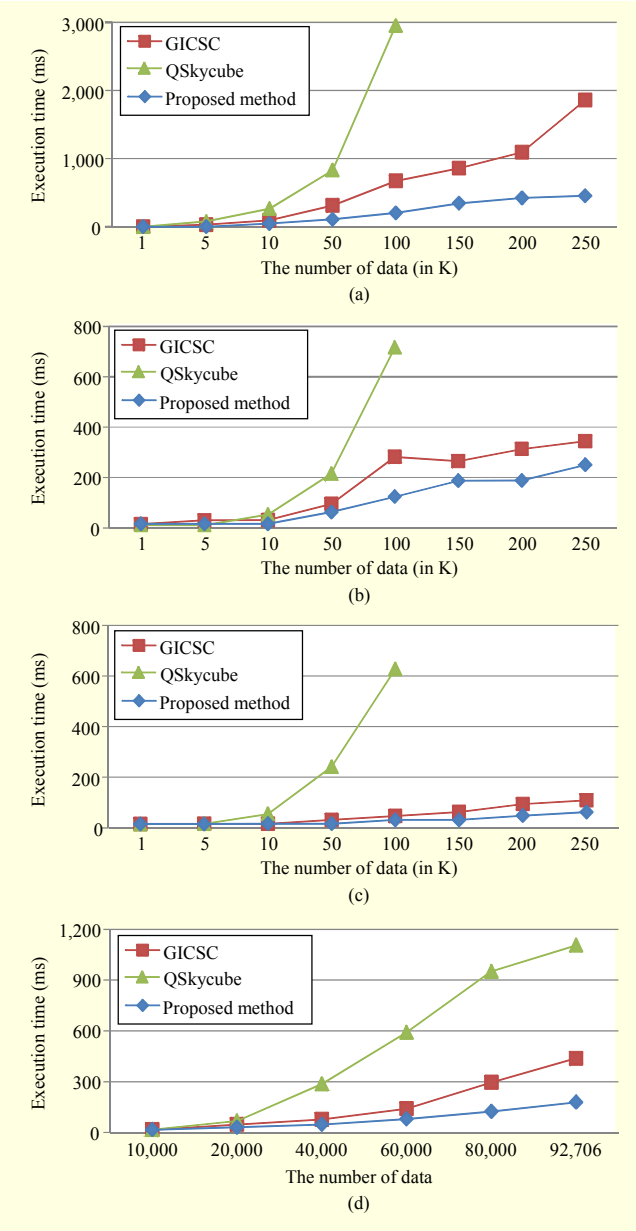

Fig. 8. Execution times according to data size: (a) anti-correlated data set, (b) independent data set, (c) correlated data set, and (d) real data set.

Figure 9 illustrates the execution time of the two methods according to the number of dimensions. The data sizes of a synthetic data set and a real data set are 100,000 and 92,706, respectively. The number of dimensions is varied from 2 to 5. Notice that the number of cells in the grid is exponential over the number of dimensions of the data set. That is, the number of cells increases sharply with the growth of the number of dimensions. As the experimental results show in Fig. 9, we find that the proposed method is faster than GICSC using all three data types. This is because the skyline influence region of the proposed method is more accurate than that of GICSC and

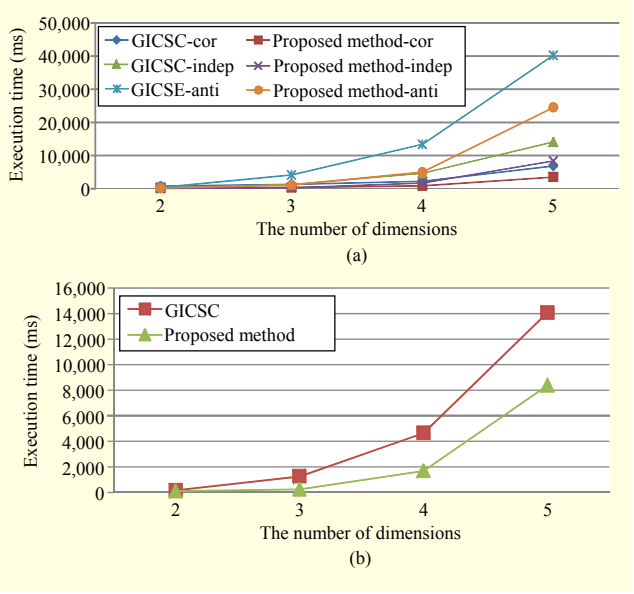

Fig. 9. Execution times according to the number of dimensions: (a) synthetic data set and (b) real data set.

more irrelevant data points can be filtered out. Like synthetic data sets, the results of real data sets show that the proposed method outperforms GICSC.

VI. Conclusions

In this paper, we propose adopting a multi-layer grid structure to filter out unnecessary computation while performing continuous skyline queries over large dynamic data sets. The proposed method includes two phases. In the first phase, the multi-layer grid structure and the corresponding skyline influence regions are obtained by processing the given initial data. In the second phase, the continuous skyline queries over dynamic data are executed only if the updated data are identified in skyline influence regions. The skyline result and skyline influence regions are updated according to the processing of the continuous skyline queries. Since more unnecessary computation can be filtered out by the proposed multi-layer grid structure, the computation cost of continuous skyline queries was reduced. The experimental results show that the proposed method outperforms existing methods in terms of synthetic data sets and real data sets. In the future, we plan to apply our method to some real applications.

Acknowledgments

This research was supported by NSFC (61602354), China Postdoctoral Science Foundation (2015M572526), the Fundamental Research Funds for the Central Universities the MSIP (Ministry of Science, ICT and Future Planning), Korea,

under the ITRC (Information Technology Research Center) support program (IITP-2016-H8501-16-1013, IITP-2016-H8601-16-1008) supervised by the IITP (Institute for Information & Communication Technology Promotion), and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2016R1A2B3007527).

References

[1] S. Börzönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," *Proc. Int. Conf. Data Eng.*, Heideberg, Germany, Apr. 2–6, 2001, pp. 421–430.

[2] K.L. Tan, P.K. Eng, and B.C. Ooi, "Efficient Progressive Skyline Computation," *Proc. VLDB*, Rome, Italy, Sept. 11–14, 2001, pp. 301–310.

[3] D. Papadias et al., "An Optimal and Progressive Algorithm for Skyline Queries," *Proc. SIGMOD Int. Conf. Manag. Data*, San Diego, CA, USA, June 9–12, 2003, pp. 467–478.

[4] D. Papadias et al., "Progressive Skyline Computation in Database System," *ACM J.*, vol. 30, no. 1, Mar. 2005, pp. 41–82.

[5] J. Chomicki et al., "Skyline with Presorting," *Proc. Int. Conf. Data Eng.*, Bangalore, India, Mar, 5–8, 2003, pp. 717–720.

[6] D. Kossmann, F. Ramsak, and S. Rost, "Shooting Stars in the Sky: an Online Algorithm for Skyline Queries," *Proc. VLDB*, Hong Kong, China, Aug. 20–23, 2002, pp. 275–286.

[7] C.Y. Chan et al., "Finding k-Dominant Skylines in High Dimensional Space," *Proc. SIGMOD Int. Conf. Manag. Data*, Chicago, IL, USA, June 27–29, 2006, pp. 503–514.

[8] X.M. Lin et al., "Stabbing the Sky: Efficient Skyline Computation over Sliding Windows," *Proc. Int. Conf. Data Eng.*, Tolkyo, Japan, Apr. 5–8, 2005, pp. 502–513.

[9] Y.F. Tao and D. Papadias, "Maintaining Sliding Window Skylines on Data Streams," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 3, Mar. 2006, pp. 377–391.

[10] Y.W. Lee, K.Y. Lee, and M.H. Kim, "Efficient Processing of Multiple Continuous Skyline Queries over a Data Stream," *Inform. Sci.*, vol. 221, Feb. 2013, pp. 316–337.

[11] H. Wang et al., "Efficient Processing of Continuous Skyline Query over Smarter Traffic Data Stream for Cloud Computing," *Discrete Dynamics Nature Soc.*, vol. 2013, 2013, pp. 1–10.

[12] Z.Y. Huang et al., "Continuous Skyline Queries for Moving Objects," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 12, Dec. 2006, pp. 1645–1658.

[13] M.W. Lee and S.W. Hwang, "Continuous Skylining on Volatile Moving Data," *Proc. Int. Conf. Data Eng.*, Shanghai, China, Mar. 29–Apr. 2, 2009, pp. 1568–1575.

[14] L. Tian et al., "Continuous Monitoring of Skyline Query over Highly Dynamic Moving Objects," *Proc. Int. Workshop Data Eng. Wireless Mobile Access*, Beijing, China, June 10, 2007, pp. 59–66.

[15] L. Tian et al., "Grid Index based Algorithm for Continuous Skyline Computation," *Chinese J. Comput.*, vol. 6, no. 6, June 2008, pp. 998–1012.

[16] H.T. Kung, F. Luccio, and F.P. Preparata, "On Finding the Maxima of a Set of Vectors," *J. ACM*, vol. 22, no. 4, Oct. 1975, pp. 469–476.

[17] J. Lee and S. Hwang, "QSkycube: Efficient Skycube Computation using Point-Based Space Partitioning," *Proc. VLDB Endowment*, vol. 4, no. 3, Dec. 2010, pp. 185–196.

[18] J. Lee and S. Hwang, "Toward Efficient Multidimensional Subspace Skyline Computation," *VLDB J.*, vol. 23, no. 1, Feb. 2014, pp. 129–145.

[19] M. Morse, J.M. Patel, and W.I. Grosky, "Efficient Continuous Skyline Computation," *Inform. Sci.*, vol. 177, no. 17, Sept. 2007, pp. 3411–3437.

[20] J. Lim et al., "A Continuous Reverse Skyline Query Processing Method in Moving Objects Environments," *Data Knowl. Eng.*, Vol. 104, July 2015, pp. 1–14.

[21] Y. Hsueh et al., "SkyEngine: Efficient Skyline Search Engine for Continuous Skyline Computations," *IEEE Conf. Data Eng.*, Hannover, Germany, Apr. 11–16, 2011, pp. 1316–1319.

[22] H. Li et al., "An Efficient Grid Method for Continuous Skyline Computation over Dynamic Data Set," *J. Contents*, vol. 6, no. 1, 2010, pp. 47–51.

[23] X.W. Wang and Y. Jia, "Grid-Bsed Probabilistic Skyline Retrieval on Distributed Uncertain Data," *Int. Conf. DASFAA*, Hong Kong, China, Apr. 22–25, 2011, pp. 538–547.

He Li is a professor at the School of Software, Xidian University, China. He received his MS and PhD degrees from the Department of Information and Communication Engineering of Chungbuk National University, Cheonju, Rep. of Korea. His main research interests include spatial data, social data, knowledge graph data, and graph data mining technologies.

Jaesoo Yoo is a professor in the Department of Information and Communication Engineering, Chungbuk National University, Cheonju, Rep. of Korea. He received his MS and PhD degrees in Computer Science from the Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea in 1991 and 1995, respectively. His research interests are database systems, storage management systems, sensor networks, distributed computing, and big data processing.