SKYLINE SETS QUERIES FOR INCOMPLETE DATA

Mohammad Shamsul Arefin and Yasuhiko Morimoto

Graduate School of Engineering, Hiroshima University
Kagamiyama 1-7-1, Higashi-Hiroshima 739-8521, Japan
d105660@hiroshima-u.ac.jp, morimoto@mis.hiroshima-u.ac.jp

ABSTRACT

With the increase of data volume, advanced query operators, such as skyline queries, are necessary in order to help users to handle the huge amount of available data by identifying a set of interesting data objects. Skyline queries help us to filter unnecessary information efficiently and provide us clues for various decision making tasks.

Most of the existing skyline algorithms cannot preserve individual's privacy and are not well suited for data with outliers and frequently updated data. Considering these issues, earlier we have proposed skyline sets queries from databases where all dimensions are available for all data items and considered an efficient algorithm for computing convex skyline sets. In this paper, we use that idea for skyline sets queries for incomplete data and propose a method, namely, RBSSQ. RBSSQ method uses a replacement-based approach and is applicable to the databases having any number of missing dimensions in the database objects. We have conducted several experiments in terms of computational cost and found that our proposed method can efficiently compute skyline sets from data items with missing values.

KEYWORDS

Skyline sets, touching oracle, incomplete data, atomic point.

1. INTRODUCTION

Skyline queries retrieve a set of skyline objects so that a user can choose promising objects or eliminate unnecessary objects. Given a *k*-dimensional database *DB*, an object *p* is said to be in skyline of *DB* if there is no object *q* in *DB* such that *q* is better than *p* in all *k*-dimensions. If there exists such an object *q*, then we say that *p* is dominated by *q* or *q* dominates *p*. Consider the example of Figure 1. The table in Figure 1 is a list of five records of a sensor device, each of which contains two numerical attributes "Temperature" and "Humidity". In the list, the best choice usually comes from the skyline, i.e., one of $\{r_1, r_3, r_4\}$ (See Figure 1 (b)). A number of efficient algorithms for computing skyline have been reported in the literature [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. All of these works have several limitations.

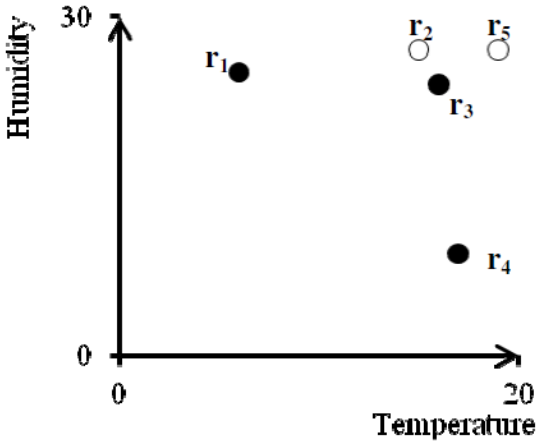
First, they consider that there is no missing value in the data. Such an assumption of completeness is not practical in many cases. For example, consider a movie rating application where each user rates movies from thousands of movies. It is highly unlikely that every single user will rate all movies. Instead, a user will rate only the movies that interest her / him. As a result, each movie will be represented as a *D*-dimensional point with several blank (i.e., missing) dimensions. Another example is from the hotel application where some hotels may not disclose some of their

DOI : 10.5121/ijcsit.2012.4506

properties. These undisclosed properties are represented as incomplete entries within the hotel multi-dimensional point representation.

Second, they are not robust in case of databases with outliers, though the existence of outliers is a common problem and it is very much important to minimize the effect of outliers in the

ID	Temperature	Humidity
r_1	6	25
r_2	15	27
r_3	16	24
r_4	17	9
r_5	19	27



(a) Sensor data (b) Skyline

Figure 1. Sensor data and corresponding skyline

computation. In different field of computing i.e. ubiquitous computing and sensor computing, we need to collect data via various sensor devices. In such situations, there are strong possibilities that some data may be outside of the general distribution of the data. Use of any conventional skyline query algorithm in such a data set may retrieve some outliers as skyline. Taking decision based on such skyline results can create serious problems in decision-making.

Again, consider the example of Figure 1. The table in Figure 1 is a list of five records of a sensor device, each of which contains two numerical attributes “Temperature” and “Humidity”. From the data of Figure 1(a), we can see that there are two records r_1 and r_4 have wrong values, which we call outliers, in their attributes “Temperature” and “Humidity”, respectively. If we perform a conventional skyline query, it will return r_1, r_3, r_4 (see Fig. 1 (b)) as skylines. Note that in the result two records r_1 and r_4 have been included. As these two records have outliers in their attributes, taking decision based on either r_1 or r_4 will create a great problem.

Third, conventional skyline query algorithms are not stable in case of update operation. Since skyline takes time to compute, we usually use precompute skyline results to answer a query quickly. If a record in a database is updated, we have to recompute the skyline results. However, if records in a database are frequently updated, we cannot prepare the precomputed results in time.

Consider the snapshot of a frequently updated database at a moment as shown in Figure 2. From this example, we can find that a conventional skyline query returns $\{h_1, h_3, h_4\}$ (see Figure (b)) as a skyline result. After a time interval, consider the snapshot of the same database as in Figure 3. From Figure 3 we can easily find that the result of any conventional skyline query is h_2, h_3 and h_5, which is different from the result before update. We can see that precomputed skyline $\{h_1, h_3, h_4\}$ is meaningless for users who are interested in the cheapest price unless it is recomputed. This problem has been an important research issue of skyline query.

Fourth, there is no consideration about the issue of individual's privacy in these works. However, recently preserving individual's privacy becomes an important issue in data management. In a database, it might be necessary to hide individual information to preserve privacy. People often do not want to disclose their records' values during the computation procedure. In such a privacy aware environment, we cannot use conventional skyline queries.

ID	Price	Distance
h_1	3	8
h_2	5	4
h_3	4	3
h_4	9	2
h_5	7	3

(a) Hotels data befo

(b) Skyline before update

Figure 2. Hotels data and corresponding skyline before update

ID	Price	Distance
h_1	6	8
h_2	5	4
h_3	4	5
h_4	9	3
h_5	7	2

(a) Hotels data after update

(b) Skyline after update

Figure 3. Hotels data and corresponding skyline after update

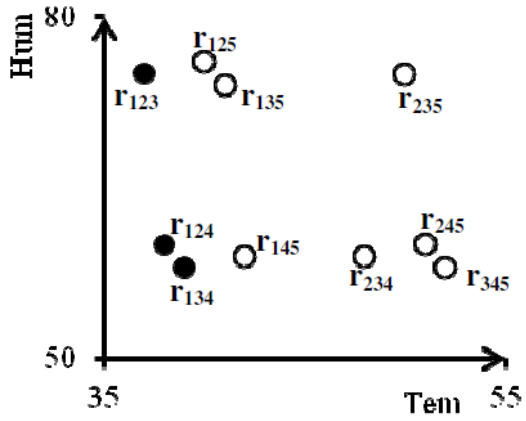
To overcome the first limitation, Khalefa et al. [11], developed several algorithms to compute skyline queries from data with missing values and proposed three different algorithms for computing skyline. However, none of their algorithm is robust against data with outliers. In addition, in their work there is no consideration about preserving the privacy of individual's privacy.

In [12], for the first time, we have introduced the concept of skyline sets queries that is robust to a certain level for databases with outliers and for frequently updated databases and can preserve individual's privacy.

Figure 4(a) is a list of 3-sets, in which all of the combinations of three records are listed. In Figure 2(a), "ID" denotes a set of three records and the attributes "Tem" and "Hum" represent the sums of temperature and humidity of three records of Figure 1(a), respectively. In the example, r_{123} denotes a set of three records r_1, r_2, and r_3. "Tem" and "Hum" of r_{123} are the sums of the "Temperature" and "Humidity" of records in the set, respectively. The skyline of the combinations of three records are $\{r_{123}, r_{124}, r_{134}\}$ as shown Figure 4(b). Note that in temperature sensitive situations if anyone takes decision based on the result of conventional skyline queries as shown in Figure 1, she / he will choose r_1 and the decision will be a wrong one as the value

ID	Tem	Hum
r_{123}	37	75
r_{124}	38	60
r_{125}	40	76
r_{134}	39	58
r_{135}	41	74
r_{145}	42	59
r_{234}	48	59
r_{235}	50	75
r_{245}	51	60
r_{345}	52	58

(a) 3-sets of sensor data

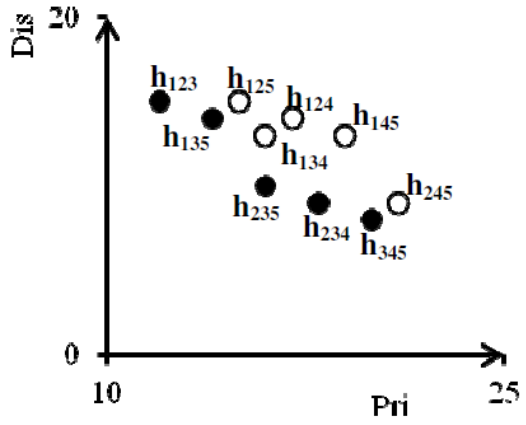


(b) Skyline of 3-sets

Figure 4. Three sets of sensor data and corresponding skyline 3-sets

ID	Pri	Dis
h_{123}	12	15
h_{124}	17	14
h_{125}	15	15
h_{134}	16	13
h_{135}	14	14
h_{145}	19	13
h_{234}	18	9
h_{235}	16	10
h_{245}	21	9
h_{345}	20	8

(a) Sets of three hotels before update



(b) Skyline of 3 hotels before update

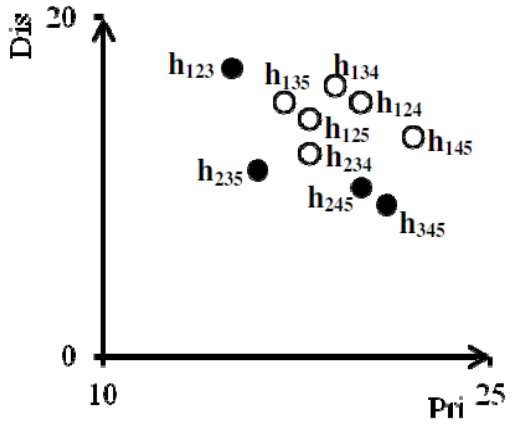
Figure 5. Three sets of hotels and corresponding skyline 3-sets before update

"6" in "Temperature" attribute is an outlier. On the other hand, it is observed that if the decision is taken based on the result of skyline sets queries, we can see that the chosen value is around "12" and the decision is much more accurate. In this way, using skyline set queries we can reduce the effect of outliers at a certain level.

If we consider the update situation and skyline sets queries with s = 3 from our examples of Figure 2 and Figure 3, we can find that the results are $\{h_{123}, h_{135}, h_{235}, h_{234}, h_{345}\}$ and $\{h_{123}, h_{235}, h_{245}, h_{345}\}$ as shown in Figure 5(b) and 6(b), respectively. From the results of skyline sets of Figure 5 and Figure 6, we can see that the result is almost same before and update operation. That means skyline sets queries are more robust against update than conventional skyline queries.
Note that skyline sets queries can preserve individual's privacy as there is no discloser of individual record's values.

ID	Pri	Dis
h_{123}	15	17
h_{124}	20	15
h_{125}	18	14
h_{134}	19	16
h_{135}	17	15
h_{145}	22	13
h_{234}	18	12
h_{235}	16	11
h_{245}	21	9
h_{345}	20	10

(a) Sets of three hotels after update



(b) Skyline of 3 hotels after update

Figure 6. Three sets of hotels and corresponding skyline 3-sets after update

Later, we have extended the idea of skyline sets queries for spatio-temporal databases [13] and for distributed databases [14, 15]. However, none of our work considers the issue of incomplete data i.e. databases with missing values.

In this paper, for the first time, we propose a Replacement Based Skyline Sets Queries (RBSSQ) method for computing skyline sets from the databases with missing values. Our proposed approach can efficiently calculate skyline sets from the databases with missing values. At the same time it can protect privacy of individual's.

The remainder of this paper is organized as follows. Section 2 provides a brief review of related works on skyline queries. Section 3 gives the preliminary ideas about different related topics. In Sections 4, we detail the skyline sets queries from databases with missing values. Section 5 presents the experimental results. Finally, we conclude and sketch future research directions in Section 6.

2. RELATED WORKS

Borzonyi et al. [1] first introduced the skyline operator into database systems and proposed Block Nested Loop (BNL), Divide-and-Conquer (D&C), and B-tree based algorithms. As a variant of

BNL, Chomicki et al. [2] improved BNL algorithm with the help of a sort-Filter-Skyline (SFS) algorithm. In SFS, data needs to be pre-sorted using a monotone scoring function, which can simplify the selection of skyline objects. Tan et al. [3] proposed two progressive algorithms: Bitmap and Index. The bitmap algorithm represents points in bit vectors and performs bit-wise operations. On the other hand, the index approach uses data transformation and B+-tree indexing. Kossmann et al. [4] proposed a Nearest Neighbor (NN) method. It selects skyline points by recursively invoking R*-tree based depth-first NN search over different data portions. Papadias et al. [5] proposed a Branch-and-Bound Skyline (BBS) method based on the best-first nearest neighbor algorithm. Godfrey et al. [6] provided a comprehensive analysis of previous skyline algorithms without indexing supports and proposed a new hybrid method with improvement. All of the above works consider a sole database for skyline computation.

Due to the development of network infrastructure, parallel and distributed skyline query processing also attracts reasonable considerations. In [7], Wu et al. first address the problem of parallelizing skyline queries over a shared-nothing architecture. They provided two mechanisms: recursive region partitioning and dynamic region encoding for the execution of skyline queries. In their approach, a server starts the skyline computation on its data after receiving the results of other servers based on the partial order. It causes a waiting delay of the servers. In our work, such problem is localized within the cluster. Sungwoo et al. [8] introduced two parallel skyline algorithms in multicore architectures. The first one is a parallel version of BBS algorithm. The second one is known as *pskyline*, which is based on skeletal parallel programming [16]. Gao et al. [9] proposed parallel computation of skyline queries in multi-disk environment using parallel R-trees. The core of their scheme is to visit more entries simultaneously and to enable effective pruning strategies. Cui et al. [10] introduced skyline queries in large-scale distributed environments without the assumption of any overlay structures and propose *PaDSkyline* algorithm. *PaDSkyline* is an algorithm that significantly reduces the response time by performing parallel processing over site groups produced by a partition algorithm. Within each group, it locally optimizes the query processing. It also improves the network transmission efficiency by performing early reduction of skyline candidates.

None of the above works considers the situation of missing data values. Also, none of them is robust against databases with outliers and frequently updated databases and there is no consideration about individuals privacy.

In [11], Khalefa et al. first considers the case of skyline computation from databases with missing values. At first, they defined new domination rule for incomplete data. Then they proposed three algorithms: "*Replacement*", "*Bucket*", and "*ISkyline*" for skyline queries from data with missing values. However, none of them is robust against databases with outliers and frequently updated databases. Moreover, there is no protection mechanism against individual's privacy in this work.

Our works in [12], [13], [14], and [15] can provide enough protection of individual's privacy. They are also robust against data with outliers and frequently updated databases. However, none of them considers the situation of databases with missing values.

In this paper, we introduce RBSSQ method that is a simple and efficient method to calculate skyline sets queries from databases with missing values.

3. PRELIMINARIES

We consider a database DB having k attributes and n records. Let a_1, a_2, …, a_k be the k attributes of *DB*. Without loss of generality, we assume that smaller value in each attribute is better.

3.1. Skyline Queries

Let p and q be objects in DB. Let $p.a_l$ and $q.a_l$ be the l-th attribute values of p and q respectively, where $1 \le l \le k$. An object p is said to dominate another object q, if for all k attributes $p.a_l \le q.a_l$ for $1 \le l \le k$ and on at least one attribute $p.a_j < q.a_j$ for $1 \le j \le k$. The skyline is a set of objects, which are not dominated by any other object in DB.

3.2. Skyline Sets Queries

Let s-set be a set of s objects. We assume s is a relatively small number such that $2 \le s \le 10$. Let

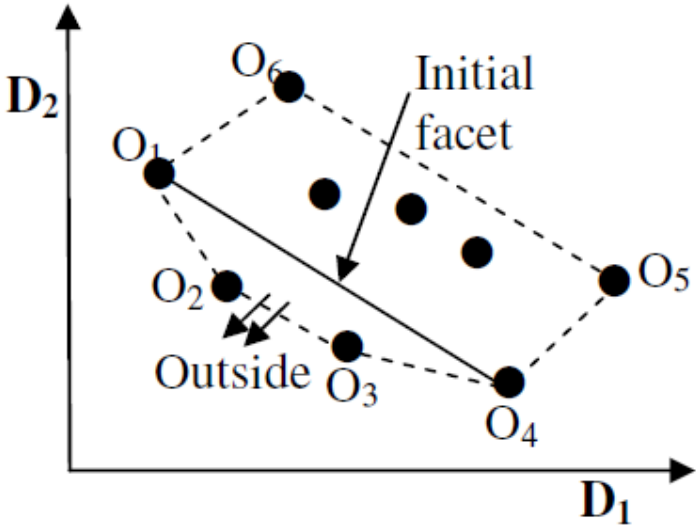


Figure 7. Convex hull and convex skyline

S be the set of all s-sets in DB. Note that the number of s-sets in DB is ${}_nC_s = \dfrac{n!}{s!(n-s)!}$ be the number of s-sets, which we denote by $|S|$. We denote $p.a_l$ as the l-th attribute value of a record p in S. An s-set $p \in S$ is said to dominate another s-set $q \in S$, denoted as $p \le q$, if $p.a_r \le q.a_r$, $1 \le r \le k$ for all k attributes and $p.a_t < q.a_t$, $1 \le t \le k$ for at least one attribute. We call such p as dominant s-set and q as dominated s-set between p and q.

An s-set $p \in S$ is said to be a skyline s-set if p is not dominated by any other s-set in S.

3.3. Convex Skyline

We can consider a record in S to be a point in k-dimensional vector space. Convex hull for the set of S points is the minimum convex solid that encloses all of the objects of S. The dotted line polygon of Figure 7 is an example of convex hull in two-dimensional space. In the Figure 7, O_1 and O_4 are the objects that have the minimum values of attribute in D_1 and D_2, respectively. Notice that such objects must be in the convex hull. We call the line between O_1 and O_4 "the initial facet". Among all objects in the convex hull, objects that lie outside of the initial facet are skyline objects and we call such objects "convex skyline objects". In k-dimensional space, we compute such initial hyperplane surrounded by k objects as the initial facet. Then, we compute convex skyline objects that lie in the convex hull and outside the initial facet.

The definition of convex skyline sets problem can be simplified as follows:

Given a natural number *s*, find all *s*-sets those lay in both the convex hull and the skyline of *S*.

In this paper, we have introduced the methodology for computing such skyline sets from data with missing values.

4. SKYLINE SETS QUERIES FOR INCOMPLETE DATA

4.1. Problem Formulation

We assume a database *DB* has a view table whose schema has following columns: *ID*, a_1, a_2,..., a_k, where *ID* is the primary key attribute and a_j (j = 1,...,k) are *k*-dimensional numerical attributes. Some attribute values of each object may be unknown or missed. Table 1 is an example of such a database that contains ten two dimensional records with some records have no values in either a_1 or a_2. We assume that in Table 1, the domain values in both attributes a_1 and a_2 are between 1 and 9 and smaller values in each attribute are better. We need to compute skyline sets from such a database.

Table 1. Database with missing values

ID	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_1	o_{10}
a_1	--	1	2	--	6	1	--	--	3	--
a_2	1	3	--	3	2	--	1	1	1	1

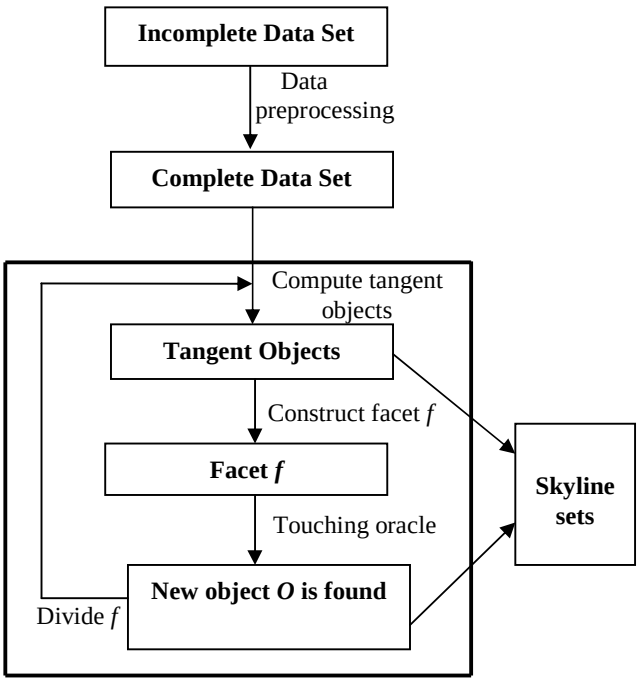


Figure 8. System architecture of RBSSQ method

4.2. Replacement Based Sets Skyline Computation Method

Our proposed Replacement Based Sets Skyline Queries (RBSSQ) method consists of two phases: data preprocessing phase and skyline sets computation phase. Figure 8 shows overall methodology of RBSSQ.

4.2.1 Data Preprocessing

In data preprocessing phase, we first search the database for obtaining missing values of the records in each attribute. Then, we replace the missing values of the records in an attribute with a value outside the domain values. The choice of such a value for an attribute depends on the nature of the data in that attribute. If smaller values are better for an attribute, we shall replace missing values in that attribute with a value larger than the domain value and vice versa. As for example, we replaced the missing values of Table 1 in each attribute with 10 as shown in Table 2. From the data in Table 2, we can see that now there is no missing value. Figure 9 shows the algorithm for such replacement.

Table 2. Database with no missing values

ID	$\mathbf{o_1}$	$\mathbf{o_2}$	$\mathbf{o_3}$	$\mathbf{o_4}$	$\mathbf{o_5}$	$\mathbf{o_6}$	$\mathbf{o_7}$	$\mathbf{o_8}$	$\mathbf{o_1}$	$\mathbf{o_{10}}$
$\mathbf{a_1}$	**10**	1	2	**10**	6	1	**10**	**10**	3	**10**
	1	3	**10**	3	2	**10**	1	1	1	1

Algorithm *Data Preprocessing*

Input: Incomplete Data Set *S*, value $value_j$ that will replace missing values in attribute *j*. Here, $j = 1$ to k
Output: Complete Data Set *T*
1: **begin**
2. **repeat**
3: read point p_i, $1 \leq i \leq n$ from input *S*
4: **for** each p_i **do**
5: check every dimension *j*, $j = 1$ to k
6: replace each missing value with $value_j$.
7: **end for**
8: until end of input *S*
9: **end**

Figure 9: Algorithm for replacement of missing values

Table 3. Inner products with tangent lines

o	$\mathbf{o_1}$	$\mathbf{o_2}$	$\mathbf{o_3}$	$\mathbf{o_4}$	$\mathbf{o_5}$	$\mathbf{o_6}$	$\mathbf{o_7}$	$\mathbf{o_8}$	$\mathbf{o_1}$	$\mathbf{o_{10}}$
$\mathbf{(\Theta_1, o)}$	-10	-1	-2	-10	-6	-1	-10	-10	-3	-10
$\mathbf{(\Theta_2, o)}$	-1	-3	-10	-3	-2	-10	-1	-1	-1	-1

4.2.2. Skyline Sets Computation

Skyline sets computation module computes skyline sets from the processed data. It uses the same concept as our previous work [12]. It utilizes touching oracle function to compute a point on the convex hull without generating all *s*-sets. It computes the tangent point of the convex hull of *S* and a (*k*-1)-dimensional hyperplane directly from *DB*.

From the information of Table 2, we can see that there are ten records in the database *DB*. Each of the ten records can be represented as a two-dimensional point and we call such a point as an atomic point *o*.

Assume there is a (*k* -1)-dimensional hyperplane (which is a line if *k* = 2), whose normal vector is Θ_1 = (-1, 0) in the two-dimensional space. In order to find the tangent point with the 1-dimensional hyperplane (line) and the convex hull without precomputing all points in *S*, we compute (Θ_1, *o*), i.e., inner products of the normal vector and each atomic point as in the second column of Table 3. We choose the top three inner products, i.e., {o_2, o_3, o_6}. Those top three inner products composes the tangent point (4, 23), which is the 3-set, o_{236}. Similarly, for a line with Θ_2 = (0, -1), we can find o_{178} is the top three in the inner products of (Θ_2, *o*). It composes the tangent point (23, 3), which is the tangent point of the convex hull and the 1-dimensional hyperplane (line) whose normal vector is Θ_2.

Like this way, we can compute a tangent point, which is a point on the convex hull, by giving the normal vector of a tangent line. In *k*-dimensional case, we can find a tangent point with a tangent (*k*-1)-dimensional hyperplane by giving the normal vector of the tangent (*k* - 1)-dimensional hyperplane.

The touching oracle function chooses top s inner products from n atomic points in *DB*. Since *s* is negligible small constant, we can compute the tangent point by scanning *n* atomic points only once, which is *O*(*n*).

Next, we need to compute all convex skyline sets using touching oracle function.

Table 4. Convex skyline 3-sets

ID	**P_1**	**P_2**	**P_3**	**P_4**	**P_5**
a_1	4	23	10	5	14
a_2	23	3	6	14	5

Touching oracle function first computes initial *k* tangent points with initial *k* normal vectors. These *k* tangent points construct the initial facet. In the example above, we have initial two tangent points: we have P_1 = (4, 23) with the normal vector Θ_1 = (-1, 0) and we have P_2 = (23, 3) with the normal vector Θ_2 = (0, -1). Using the facet containing the two initial points, we can compute the normal vector of the facet as $\Theta_{1,2}$ =(-(23-3), (4-23)) = (-20, -19), which directs outside of the facet. Using this normal vector, we can find new tangent point o_{258}, which is (10, 6). The new tangent point expands the initial facet into two facets, which are the facet surrounded by P_1 = (4, 23) and (10, 6) and the facet surrounded by (10, 6) and P_2 = (23, 3).

We recursively compute tangent objects for each of the expanded facet. If we find new object outside the facet, we expand the facet further. We continually adopt the recursive operation while we can find new tangent object outside the facet. Finally, we can find all convex skyline *s*-sets. We can apply this recursive operation for higher *k*-dimensional space.

In the *k*-dimensional case, new tangent object, which is found by the touching oracle, divides the initial facet into *k* facets. In *k*- dimensional case, the normal vector of each facet can be computed as follows:

Assume we have a facet surrounded by *k* points $P1 = (p1_1, p1_2,..., p1_k)$, $P2 = (p2_1, p2_2,..., p2_k)$, ..., $Pk = (pk_1, pk_2, ..., pk_k)$. We can calculate (*k* -1) vectors like *V*1, *V*2,....,*V*(*k* -1). Then, the normal

vector of the facet that directs outside can be computed as the expansion of the following determinant.

$$V1 \otimes ... \otimes V(k-1) = \begin{vmatrix} e1 & e2 & ... & ek \\ v1_1 & v1_2 & ... & v1_k \\ ... & ... & ... & ... \\ v(k-1)_1 & v(k-1)_2 & ... & v(k-1)_k \end{vmatrix}$$

If P is found outside of the facet, then the k new facets are as (P, $P2$,..., Pk - 1, Pk), ($P1$, P,..., Pk - 1, Pk),...,($P1$, $P2$, ..., Pk - 1, P). The normal vectors of these k facets are (($P2$ - P) ⊗ ... ⊗ (Pk - 1 - P) ⊗ (Pk - P)), ((P - $P1$) ⊗ ... ⊗ (Pk – 1 - $P1$) ⊗ (Pk - P)),...., (($P2$ - $P1$) ⊗ ... ⊗ (Pk - 1 -$P1$) ⊗ (P -$P1$)).

Replacing k with any value, we can obtain the normal vector calculation procedure for that dimensional data.

Using above computation procedure we obtain following points as shown in Table 4 as skyline 3-sets.

Note that in our result of Table 4 some of the skyline 3-sets contain inserted value in an attribute. We do not need any postprocessing of such s-sets to remove the inserted value. This is because if someone wants to hide some attributes' values of some records they should not get priority in those attributes. From the result of Table 4, anyone can take his decision. As for example, if anyone prefers some think related to attribute a_1, she / he can choose P_1 and can find that the average value is around 1. Similarly, if someone is interested in a_2, she / he can easily choose P_2 and can find that it is the best choice corresponds to a_2. In this way, skyline sets queries can help users to take their proper decision from incomplete data without disclosing individual's records values.

4.2.3. Complexity Analysis

The complexity of the proposed RBSSQ technique involves the computation cost of preprocessing and convex skyline sets calculation. If we have k-dimensional database with n objects in the database, preprocessing complexity is $O(n * k)$. As $k << n$, ignoring k provides preprocessing complexity $O(n)$. As stated earlier, skyline sets computation requires at most $O(n)$ time. Therefore, overall computation complexity of our proposed RBSSQ method is $O(n)$.

5. IMPLEMENTATION AND EXPERIMENTS

This section experimentally evaluates the performance of the proposed algorithms. We conduct a series of experiments to evaluate the performance of our method using synthetic data. We evaluated the efficiency and the scalability of the proposed methods in term of the cardinality and sets size of the datasets. We conduct simulation experiments in a machine having Intel core i5 processor, 2.3 GHz CPU, and 4 GB main memory. Each experiment is repeated five times and the average is taken.

We generated incomplete synthetic data with different percentage of missing values, different size and different dimensions.

The generation of the synthetic datasets is controlled by the parameters "p", "k", and "size", where "p" is the percentage of missing values, "k" is the number of attributes, "size" is the total

size of data sets. We restrict dimensionality within five, because from the users' point of view, we consider that higher dimension will increase the decision complexity and the result of skyline becomes useless in practice. Similar to others skyline related works in the literature, we employ the elapsed time as the performance metric.

Figure 10 shows the effect of set size. Here, we use 100k data and 30% missing values. From Figure 10, it is found that our proposed method becomes gradually slow if "s" increases. This is because when "s" becomes large the number of retrieved sets also increases. So, we can say that our algorithm can perform well even if "s" becomes large.

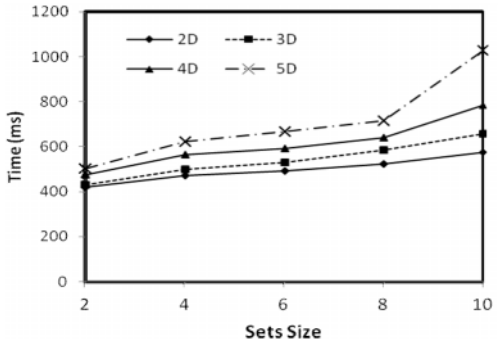


Figure 10. Time varying sets size

Next, we evaluate the effect of dataset size. We used synthetic datasets with cardinality 10k, 25k, 50k, 75k, and 100k. In this experiment, we fix "s" to 10 and missing data fixed to 30%. Figure 11 shows the result. We observe that the response time increases if the dataset size increases. We also observe that it gradually increases if the dimension increases. Similar to the previous experiments, the total elapsed time increases if dataset size and dimension increase.

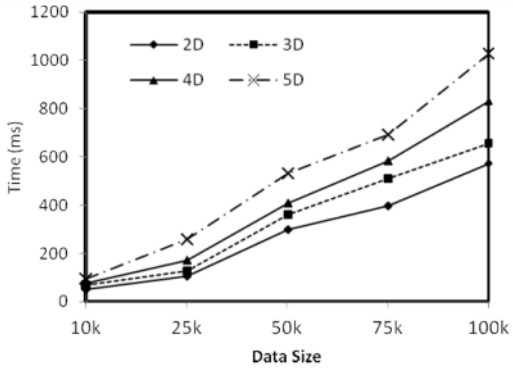


Figure 10. Time varying data size

Finally, we evaluate the performance with percentage of missing data in the data set. The missing data varies from 10% to 40% with data set fix to 100k and s fix to 10. Experimental result is shown in Figure 12. From the experiment, we find that for the same dimension there is almost no performance degradation with different percentage of missing attributes. However, among the dimensions there is a difference in the performance. From this experiment, we can say that our algorithm is an efficient one.

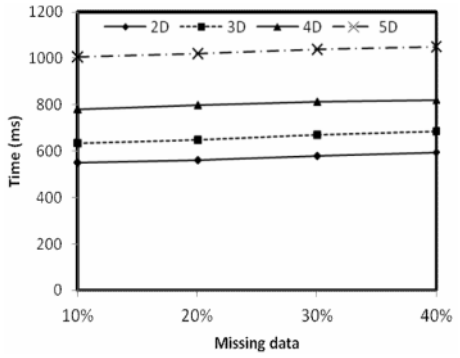


Figure 10. Time varying percentage of missing values

As a short summary, our performance evaluations on synthetic datasets indicate that the proposed algorithm is efficient and scalable regarding data size. To the best of our knowledge, there is no previous work that focuses on the skyline sets queries problem in case for incomplete data.

6. CONCLUSION

In this paper, we have addressed the problem of skyline sets queries from databases with missing values and proposed a simple and efficient method to compute skyline sets from such databases while preserving individual's privacy. Experimental evaluation using synthetic datasets demonstrates that the proposed method is meaningful and is scalable enough to handle large and high dimensional datasets. As more data intensive applications emerge and most data misses some attributes' values in some dimensions, the proposed method can be helpful for decision making in such situations. Our current work assumes a centralized environment. Future works will study on distributed environment and incremental update handling. Another interesting topic is the fast retrieval of approximate sets skyline, i.e., s-sets that do not necessarily belong to the skyline but are very close. In future, we hope to consider such issues.

ACKNOWLEDGEMENT

This work was partially supported by KAKENHI (19500123). Mohammad Shamsul Arefin is supported by the scholarship of MEXT Japan.

REFERENCES

[1] S. Borzonyi, D. Kossmann, and K. Stocker, "The skyline operator," in Proc. of 17th International Conference on Data Engineering, pp. 421-430, 2001.
[2] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in Proc. of 19th International Conference on Data Engineering, pp. 717-719, 2003.
[3] K.L. Tan, P.K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in Proc. of 27th International Conference on Very Large Data Bases, pp. 301-310, 2001.
[4] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in Proc. of 28th International Conference on Very Large Data Bases, pp. 275-286, 2002.
[5] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in Proc. of ACM SIGMOD Conference, pp. 467-478, 2003.
[6] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in Proc. of 31st International Conference on Very Large Data Bases, pp. 229-240, 2005.

[7] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi, "Parallelizing skyline queries for scalable distribution," in Proc. of 10th International Conference on Extending Database Technology, pp. 112-130, 2006.
[8] S. Park, T. Kim, J. Park, J. Kim, and H. Im, "Parallel skyline computation on multicore architectures," in Proc. of 25th International Conference on Data Engineering, pp. 760-771, 2009.
[9] Y. Gao, G. Chen, L. Chen, and C. Chen, "Parallelizing progressive computation for skyline queries in multi-disk environment," in Proc. of International Conference of Database and Expert Systems Applications (DEXA), pp. 697-706, 2006.
[10] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou, "Parallel distributed processing of constrained skyline queries by filtering," in Proc. of 24th International Conference on Data Engineering, pp. 546-555, 2008.
[11] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline query processing for incomplete data", in Proc. of 24th International Conference on Data Engineering, pp. 556-565, 2008.
[12] M. A. Siddique and Y. Morimoto. "Algorithm for computing convex skyline objectsets on numerical atabases", in IEICE Trans. on Information and Systems, vol. 10, pp.2709-2716, 2010.
[13] Y. Morimoto and M. A. Siddique, "Skyline sets query and its extension to spatio-temporal databases", in Lecture Notes in Computer Science, vol. 5999, pp. 317-329, 2010.
[14] Y. Morimoto, M. S. Arefin, and M. A. Siddique, "Agent-based anonymous skyline set computation in cloud databases", in Int. J. Computational Science and Engineering, Vol. v, no. 1, pp. 73-81, 2012.
[15] M. S. Arefin, and Y. Morimoto, "Privacy Aware Parallel Computation of Skyline Sets Queries from Distributed Databases", in Proc. of Second International Conference on Networking and Computing, pp. 186-192, 2011.
[16] F. A. Rabhi and S. Gorlatch, "Patterns and skeletons for parallel and distributed computing," Springer, 2003.

Authors

Mohammad Shamsul Arefin received his B.Sc. Engineering in Computer Science and Engineering from Khulna University, Khulna, Bangladesh in 2002, and completed his M.Sc. Engineering in Computer Science and Engineering in 2008 from Bangladesh University of Engineering and Technology (BUET), Bangladesh. Now he is a PhD candidate at Hiroshima University with support of the scholarship of MEXT, Japan.

He is a member of Institution of Engineers Bangladesh (IEB) and currently working as an Assistant Professor in the Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, Chittagong, Bangladesh. His research interest includes privacy preserving data mining, multilingual data management, semantic web, and object oriented system development.

Yasuhiko Morimoto is an Associate Professor at Hiroshima Un iversity. He received B.E., M.E., and Ph.D. from Hiroshima University in 1989, 1991, and 2002, respectively. From 1991 to 2002, he had been with IBM Tokyo Research Laboratory where he worked for data mining project and multimedia database project. Since 2002, he has been with Hiroshima University. His current research interests include data mining, machine learning, geographic information system, and privacy preserving information retrieval.