IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 12, 2019, accepted November 23, 2019, date of publication December 6, 2019, date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2958202

Optimizing Skyline Query Processing in Incomplete Data

YONIS GULZAR[1], **ALI A. ALWAN**[2], **AND SHERZOD TURAEV**[3]

[1]Department of Management Information Systems, College of Business Administration, King Faisal University, Al-Ahsa 31982, Saudi Arabia
[2]Department Computer Science, Kulliyyah of Information and Communication Technology, International Islamic University Malaysia, Selangor 35100, Malaysia
[3]Department of Computer Science and Software Engineering, College of Information Technology, United Arab Emirates University, Al Ain 15551, United Arab Emirates

Corresponding authors: Yonis Gulzar (ygulzar@kfu.edu.sa) and Ali A. Alwan (aliamer@iium.edu.my)

This work was supported in part by the Deanship of Scientific Research in King Faisal University, Saudi Arabia, under Program 186304.

ABSTRACT Given the significance of skyline queries, they are incorporated in various modern applications including personalized recommendation systems as well as decision-making and decision-support systems. Skyline queries are used to identify superior data items in the database. Most of the previously proposed skyline algorithms work on a complete database where the data are always present (non-missing). However, in many contemporary real-world databases, particularly those databases with large cardinality and high dimensionality, such assumption is not necessarily valid. Hence, missing data pose new challenges if the processing skyline queries cannot easily apply those methods that are designed for complete data. This is due to the fact that imperfect data cause the loss of the *transitivity property* of the skyline method and *cyclic dominance*. This paper presents a framework called *Optimized Incomplete Skyline (OIS)* which utilizes a technique that simplifies the skyline process on a database with missing data and helps prune the data items before performing the skyline process. The proposed strategy assures that the number of the domination tests is significantly reduced. A set of experiments has been accomplished using both real and synthetic datasets aimed at validating the performance of the framework. The experiment results confirm that the *OIS* framework is indeed superior and steadily outperforms the current approaches in terms of the number of domination tests required to retrieve the skylines.

INDEX TERMS Algorithms, incomplete data, database, preference queries, query processing, skylines, skyline queries.

I. INTRODUCTION

Skyline queries are a standout amongst the most driving, unequalled and much of the time utilized preferences queries in database systems. Skyline query is a very beneficial tool that is being utilized in various applications including multi-criteria decision-making systems [3], crowd-sourcing databases [4]–[7], cloud databases [8], [9], and decision support system [10]. In most of these applications, the user may need to combine various contradictory criteria to recommend a strategic decision. Skyline queries seek to prefer one data item (p) over another data item (q) if p is as good as q in at least one dimension and better than q in all dimensions. For instance, assume there are two different data items p_1 and p_2 with the same dimensions. We say p_1 is one of the superior skyline set (S) if p_1 is as good as p_2 in at least one dimension and superior to p_2 in all other dimensions [8], [11]–[21]. Based on the concept of dominance in skyline queries, the data items present in the skyline set are always the best in respect to any monotone ranking function. Since the predominant notion is natural and straightforward, users can easily formulate the skylines. The following example of a hotel finder database demonstrates how the skyline process works. In the given scenario, a hotel database consists of 10 hotels (a to j), and each hotel has the two dimensions of price and distance. We assume a person is looking for a hotel that is the nearest to the workshop venue and the lowest in price.

Figure 1 denotes the visual representation of a hotel finder database example with a two dimensional space (price and distance). The x-axis denotes the distance from the workshop venue to the hotel, while the y-axis indicates the price per night for each hotel.

From Figure 1 it can be seen that hotels a and d have similar distance value; however, the price of hotel d is lower than that of hotel a, so hotel d dominates a. Similarly, hotels d

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

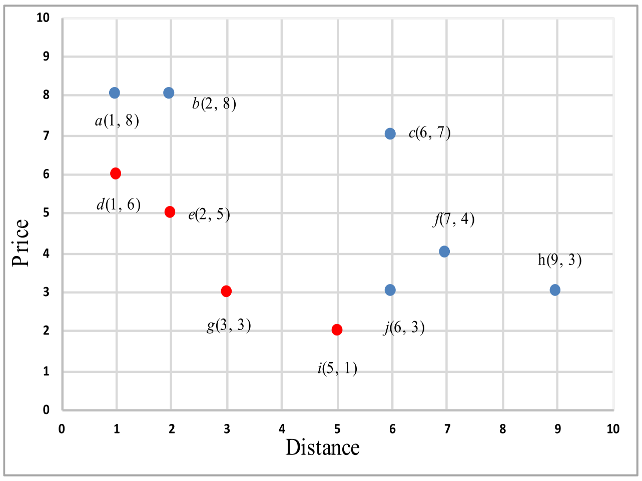


FIGURE 1. **Skylines of the hotel-finder database.**

and e dominate b and c since they are superior to hotels b and c in either one or both dimensions. Hotels $g$and i dominate hotels j, f, h, and c. Considering the completed comparison process, we conclude that hotels d, e, g, and i are the superior hotels since they are not dominated by any other hotel in the database. Hence, they can be retrieved as skylines of the hotel finder database sample.

A great number of algorithms have been proposed with the intention of evaluating skyline queries in complete databases [21]–[32]. Among the most notable skyline algorithms are Divide-and-Conquer (D&C), Block Nested-Loop (BNL) [24], Bitmap and Index [25], Nearest Neighbor (NN) [26], Sort Filter Skyline (SFS) [27], Branch and Bound Skyline (BBS) [28], Linear Elimination Sort Skyline (LESS) [22], Sort and Limit Skyline algorithm (SaLSa) [29], ZSearch [30], and OSPS [31]. Most of these skyline algorithms are tailored to identify the skylines while minimizing the searching space and avoid scanning the entire database. These algorithms are designed based on the assumption that the database is always complete if the values of all attributes are present during the skyline evaluation process. However, the assumption of data completeness cannot always be true, particularly for a database that received data from uncertain resources such as crowd and sensors. It is impractical to directly apply skyline algorithms coined for a database with complete data on an incomplete database since incomplete data influence the processing of skyline queries, cause the loss of the transitivity property of skyline and give rise to the issue of *cyclic dominance*. Consequently, this results in the infinite dominance test; in the most extreme case no data item can be retrieved as skyline.

The following scenario is used to illustrate the abovementioned problems. Assumed are three data items a, b and c with missing values with the three dimensions of d_1, d_2, and d_3. The detailed values of these data items are as follows: $a(2, *, 5)$, $b(3, 2, *)$, $c(*, 4, 3)$. The symbol (*) indicates that the value of that particular dimension is missing. To evaluate skyline queries on this kind of data items with the purpose to identify skylines means that the data items have to be compared with each other. Based on the given example, only the dimensions with known values on the common subspace are considered in the pairwise comparison. Comparing a with b (smaller is better) shows that a dominates b in one dimension (d_1). Similarly, comparing b with c indicates that b is better than c in the second dimension (d_2). Thus, c is dominated by b, however, a does not dominate c. According to the transitivity property of the skyline technique, if a dominates b and b dominates c, then a should also dominate c. However, this is not the case in our running database example with incomplete data. We can conclude that a is not better than c, while a dominates b and b dominates c. Hence, the transitivity property of the skyline technique does not hold in incomplete databases [11], [12], [14], [16], [17], [44].

Cyclic Dominance: When processing skyline queries on a database with incomplete data, there is a high possibility to encounter the issue of *cyclic dominance*. This happens due to the loss of the transitivity property of the skyline technique caused by the incompleteness of the data. Therefore, some data items may be incomparable with each other and thus, no data item is considered as skyline [11], [12], [14], [16], [17], [44]. In our running database example, the data item a dominates b and b dominates c while c dominates a. In this case, it seems that all data items are dominated by one another and hence, no skylines are introduced. Thus, an efficient solution is needed to resolve the above issues when running skyline queries in such incomplete databases. Any attempted solution should also consider the issue of lessening the number of domination tests between data items for identifying the skylines.

The following points summarize the contributions of this paper:

- We discuss the problem of identifying skylines in a database with incomplete data and explain the need for a more efficient solution.
- We conduct a comprehensive review of the leading studies conducted on skyline queries in database systems. This covers previous approaches designed for complete and incomplete databases. The review examines and highlights the strengths and the weaknesses of each approach.
- We propose a new skyline framework called *Optimized Incomplete Skylines* (*OIS*) that efficiently answers skyline queries in an incomplete database.
- We develop a new filtering scheme that helps optimize the skyline process of eliminating unwanted data items before applying the skyline technique.
- We provide a comprehensive time complexity analysis for all algorithms incorporated in the proposed approach.
- We evaluate the efficiency and the effectiveness of the proposed solution through several experiments using both real and synthetic datasets. The experiments demonstrate the effectiveness of the filtering scheme and the performance of the proposed solution.

The remainder of the paper is organized as follows: In Section II the previous works related to this research are discussed. The basic definitions and notations, which are subsequently used are set out in Section III. The proposed framework *OIS* is detailed in Section IV, while the experimental result is explained in Section V, followed by the conclusion as presented in Section VI.

II. RELATED WORK

Numerous skyline algorithms have been proposed in the research literature, each centred around enhancing the effectiveness and the execution time of the skyline process. The quantity of domination tests constitutes the most important factor influencing the execution time of the skyline process. In this regard, most proposed approaches focus on reducing the number of domination tests to the lowest possible minimum.

The first study that addressed the issue of computing skylines in relational databases was that of Borzsonyi *et al.* [24]. His research team proposed the two algorithms BNL and D&C for complete databases. Many algorithms have been proposed subsequent to BNL and D&C that sort or partition the initial dataset. Several algorithms have been proposed to improve upon BNL such as SFS [27], LESS [22], SaLSa [29], and ZSearch [30]. These algorithms are used to sort and rearrange the data items before eliminating those data items that are unlikely to be part of the skyline process. SFS sorts the entire database in non-ascending order to eliminate non-skyline data items early, whereas LESS combines the advantages of SFS and BNL by rearranging the data items. SalSa uses another function that keeps dominant data items on top of the list, prunes other data items and decreases domination tests. Besides, several other algorithms such as NN [26], BBS [28], INDEX [25], OSPS [31], BSkyTree [32] have been proposed that apply the D&C concept by dividing the dataset into small partitions before retrieving the local skylines from all partitions and combining all local skylines into the final skyline as last outputs.

Furthermore, several skyline algorithms have been proposed for incomplete databases. The first skyline algorithm designed for an incomplete database has been proposed by Khalefa *et al.* [11] whose team have proposed two algorithms, namely Bucket and Iskyline. The Bucket algorithm divides the dataset into different buckets before retrieving the skylines of each bucket. The skylines of the buckets are then further combined to identify the final skylines of the database. This algorithm has been optimized with the help of two optimization techniques to form the Iskyline algorithm. These optimization techniques result in reducing the number of pairwise comparisons needed to retrieve the skylines. RSSSQ [33] uses the same concept as used by Khalefa *et al.* [11] by replacing the missing values with numbers that are larger than the domain value to avoid losing the transitivity property of skyline and the issue of cyclic dominance. Miao *et al.* [34] proposed three algorithms known as Baseline, Virtual Point (VP) and k-iSkyband (kISB). On the downside, the Baseline algorithm requires a large number of pairwise comparisons to retrieve the skylines and ignores the correction of data items in the buckets. The VP algorithm overcomes the issue of data item correction. Lastly, kISB further optimizes VP by reducing the domination test process and eliminating redundant data storage.

The work in [12] have also highlighted the issue of processing skyline queries in incomplete datasets and proposed the SIDS approach by sorting to create the skylines. The idea of SIDS is as follows: Firstly, the data items are sorted in lists in descending order based on each dimension before comparing the data items of each sorted list with each other. This process is carried out in a round-robin fashion for each list. The dominated data items of each list are eliminated immediately while non-dominated data items are retained. The process count of every non-dominated data item is accumulated; if the process count is equal to the non-missing dimensions of that particular data item, it is considered as a skyline. Several real and synthetic datasets have been used to evaluate the performance of SIDS. However, the SIDS approach requires access to the lists in a sequential order. Therefore, it has to wait for the results of all the lists before being able to move to the next phase. Thus, increasing the number of lists may result into a lower performance of the skyline process and delays generating the skylines and making them available to the end user.

The research published in [35] has discussed the issue of processing skyline queries in incomplete data by proposing the Incomplete Data Frequent Skyline approach (IDFS). IDFS adopts the top-k frequent skyline technique as proposed by [1] that aims at controlling the size of the skyline results. IDFS relies on utilizing the concept of top-k to derive superior skylines based on the fractional skyline frequency of each data item. It is used to identify superior skylines for a database with missing values. Some experimental results have been reported that demonstrate the efficiency of IDFS using real and synthetic datasets. However, IDFS may not perform well and the performance is highly degraded if the examined space for determining the frequent skylines data items is very large.

Another recent approach has employed the SIDS approach as introduced in [15], which focuses on improving the SIDS approach by using a unique way of filtering the data items before applying the skyline technique. In addition, Zhang *et al.* [36] have proposed a framework called Compared One by One (COBO) that implements the ISSA technique to compute skylines in the database. ISSA executes the process in two phases: Firstly, it utilizes the idea of the bucket as proposed by Khalefa *et al.* [11] to eliminate dominated data before applying skyline process. Secondly, it uses an aggregate function to add all non-missing dimensions of the remaining data items in each bucket. Then the data items are sorted in ascending order according to the aggregate function in order to compute the final skylines. Miao *et al.* [37] have proposed the Improved Bitmap Index Guided algorithm (IBIG) to process top-k dominating queries for incomplete data. Top-k dominating queries combine the features of skyline queries and top-k queries. It first divides the initial dataset based on their corresponding bitmap representation and uses the binning strategy technique to optimize the data item size in the process of identifying the k-dominating skylines. The reported experimental results demonstrate that IBIG constitutes an effective technique for pruning a large number of unwanted data items that improve the skyline process in a database with incomplete data.

Furthermore, the work introduced by Alwan *et al.* [14] has also highlighted the issue of processing skyline queries in incomplete databases. They proposed the Incoskyline algorithm which works as follows: Firstly, the initial dataset is divided into subsets where each subset contains data items of the same namespace. Then two groups are constructed for each subset that contain only best data items. Next, the local skylines of each group are determined by comparing the data items contained in the groups with each other. To find the final skylines, virtual data items are generated from the local skylines of each subset and used to compare with local skylines of each subset instead of comparing all local skylines with each other. Lee *et al.* [16] proposed the Bucket and Sorting-based Bucket Skyline algorithm (SOBA) that uses the same technique of dividing the dataset into subsets based on the namespace of data items as in Iskyline [11]. For optimization, SOBA uses the same technique as used in ISSA[36] to compute the local skylines of each subset. Unlike ISSA, SOBA rearranges the data items in non-ascending order of subsets to find the final skylines. Similar to other approaches SOBA also identifies the local skylines of each subset, yet compares the data items with one another without pruning the dataset by eliminating the dominated data items before applying the skyline technique. Thus, this approach leads to many unwanted pairwise comparisons among data items during the skyline process.

Miao *et al.* [38] have proposed three algorithms (Baseline, DAG, and BIB) for processing k-dominant skyline queries on incomplete data (IKDS). Here, k-dominant skylines form the subset of superior data items among the skylines data items. In other words, k-dominant skylines return the limited number of data items that are not dominated by other data items in the database. Unlike other skylines queries that return all non-dominated data items of the database, DAG aims to minimize the searching space, whereas BIB employs the bitmap index for incomplete data and fast bitwise operations to check the k-dominance relationship under several heuristic techniques.

Zhang *et al.* [39] have proposed the PISkyline algorithm for processing probabilistic skylines in incomplete data, which returns those data items that have high probability to be part of the skylines set. Filtration techniques are used to prune dominated data items at the early stage and use two optimization techniques to accelerate the probabilistic skyline computation process.

To the best of our knowledge, the most recent work that has tackled the issue of processing skyline queries on incomplete databases has been contributed by Wang *et al.* [40] who have introduced the Skyline Preference Query approach (SPQ) that uses the same strategy of sorting data items in descending order for each dimension similar to SIDS [14]. However, SPQ divides the initial dataset into two distinct subsets based on the priority of dimensions chosen by the user meaning that the first subset has higher priority than the second subset. The local skylines of the first subset are computed using the SIDS technique [12], while the local skylines of the second subset are computed by dividing the second subset further into smaller subsets based on the bitmap representation of the data items. Lastly, the local skylines of the first subset present in SSRS are compared with the local skylines of the second subset present in RSRS to retrieve the final skylines. Although SQP has improved upon the SIDS algorithm, SPQ creates multiple pre-processed lists similar to SIDS.

Furthermore, SQP as well as SIDS algorithms are designed based on the data structure used in [2], [41], with the restriction of efficiently handling incomplete data. Table 1 summarises the skyline techniques presented in the section and includes the skyline techniques for complete and incomplete data. The table shows the approach, the year, the computation space, the database characteristic, the technique, the access method, the missing rate, the number of dimensions, and the dataset type. The definition of computation space denotes whether the skyline process needs to carry out a full scan with exhaustive search on the entire database or a partial scan for the database to determine the skylines. Here, the database characteristics indicate whether the database is complete (no missing values exist in the database) or the database has missing values in one or more dimensions.

III. DEFINITIONS

In this section, several definitions and annotations are provided related to the skylines queries in incomplete databases. These definitions and notations are important to be clarified as part of our proposed approach. Table 2 summarizes the symbols used throughout the paper, the relevant terms being explained below. Our approach has been developed in the context of an incomplete relational database, D. A relation of the database D is denoted by $R\ (d_1, d_2 \ldots, d_m)$ where R is the name of the relation with m-arity and $d = (d_1, d_2 \ldots, d_m)$ is the set of dimensions.

Definition 1, Skyline: This technique retrieves the skyline data items S in such a way that any skyline data item is not dominated by any other data item in the database.

Definition 2, Dominance on Complete Database: Given two data items p_i and $p_j \in D$ database with complete data d dimensions, p_i dominates p_j (denoted by $p_i \succ p_j$) if (and only if) the following two conditions hold:

1. $\forall d_k \in d, p_i.d_k \geq p_j.d_k$ **and**
2. $\exists d_l, \in d, p_i.d_l > p_j.d_l$

Definition 3, Skyline Queries on Complete Database: Select a data item p_i from the set of D complete database if (and only if) p_i is as good as p_j (where $i \neq j$) in all dimensions (attributes) and superior to p_j in at least one dimension.

TABLE 1. Summary of the previous approaches for skyline queries in database systems.

Approach, year	Computatio n space	Database characteristic	Technique	Access method	Missing rate (%)	No. of Dimensions	Dataset type
BNL and D&C, 2001 [24]	Full space	Complete	Sorting, clustering	Non-index	0	2-10	Synthetic
BITMAP, INDEX, 2001 [25]	Full space	Complete	Clustering, sorting	Index	0	2-10	Synthetic
NN, 2002 [26]	Full space	Complete	Clustering, sorting	Index	0	2	Synthetic
BBS, 2003 [28]	Partial	Complete	Clustering, sorting	Index	0	2-5	Synthetic
SFS, 2003 [27]	Full space	Complete	Sorting	Non-index	0	10	Synthetic
LESS,2005 [22]	Full space	Complete	Sorting	Non-index	0	2-7	Synthetic
SaLSA, 2006 [29]	Partial	Complete	Sorting	Non-index	0	2-6	Real, synthetic
Iskyline, 2008 [11]	Full space	Incomplete	Clustering	Non-index	10-90	2-2650	Real, synthetic
OSPS, 2009 [31]	Full space	Complete	Sorting, clustering	Index	0	2-22	Real
ZSearch, 2010 [30]	Full space	Complete	Sorting	Index	0	4-16	Real, synthetic
RBSSQ, 2012 [33]	Full space	Incomplete	Clustering	Index	10-40	5	Synthetic
SIDS, 2013[12]	Full space	Incomplete	Sorting	Index	10-90	2-6040	Real, synthetic
IDFS, 2013 [35]	Full space	Incomplete	Soring	Index	50	10-25	Real, synthetic
BSkyTree,2014 [32]	Full space	Complete	Sorting, clustering	Index	0	4-16	Real, synthetic
Baseline,VP,kISB, 2014 [42]	Full space	Incomplete	Clustering	Non-index	ç	5-2560	Real, synthetic
Framework, 2016 [15]	Full space	Incomplete	Sorting	Index	<50	3-18	Real, synthetic
COBO, 2016 [36]	Full space	Incomplete	Sorting	Index	25-50	5-6040	Real, synthetic
Incoskyline, 2016 [14]	Full space	Incomplete	Clustering	Non-index	<50	3-21	Real, synthetic
IBIG, 2016 [37]	Full space	Incomplete	Clustering	Index	40	5-25	Real, synthetic
SOBA, 2016 [16]	Full space	Incomplete	Clustering	Non-index	<50	10-706	Real, synthetic
Baseline, DAG, and BIB, 2016 [38]	Full space	Incomplete	Clustering	Index,	40-61	60-100	Real, synthetic
PISkyline, 2017 [39]	Full space	Incomplete	Sorting, clustering	Non-index	20	6-10	Real, synthetic
SPQ, 2017 [40]	Partial	Incomplete	Sorting	Index	<20	2-16	Real, synthetic
Model, 2018[20]	Full space	Incomplete	Sorting	Index	<50	3-18	Real, synthetic
D-SKY, 2018[43]	Full space	Incomplete	Sorting, clustering	Index	<50	3-18	Real, synthetic
ICS, 2019 [9]	Full space	Incomplete	Sorting	Index	<50	3-18	Real, synthetic
SCSA, 2019[44]	Full space	Incomplete	Sorting, clustering	Index	<50	3-18	Real, synthetic

TABLE 2. Symbols and description.

Notation	Definition
D	A Dataset containing data items (a_1, a_2, …, a_n)
R	A relation
d	Dimensions
a, p	Data items
C	Clusters created based on bitmap representation of data items
n	Number of data items
AL	Array List, it contains data items of same bitmap representation
id	Primary key of each data item, a_i
u	Array List i contain id of data items of d_i
CL	Candidate List, it contains data items after filtration
th	Threshold set by user to filter unwanted data items
S	Skylines

We use *Sskyline* to denote the set of skyline data items, $Sskyline = (p_i \forall p_i, p_j \in D, p_i \succ p_j)$.

Definition 4, Incomplete Database: Given a database D $(R_1, R_2, \ldots, R_n)$, where R_i is a relation denoted by $R_i\,(d_1, d_2, \ldots, d_m)$, D is said to be incomplete if (and only if) it contains at least a data item p_j with missing values in one or more dimensions d_k (attributes); otherwise, it is complete.

Definition 5, Dominance on Incomplete Database: Given two data items p_i and $p_j \in D$ incomplete database with d dimensions, p_i dominates p_j (denoted by $p_i \succ p_j$) if (and only if) the following three conditions hold:

1. The values of d_k and $d_l \in d$ for p_i and p_j must be non-missing **and**
2. $\forall d_k \in d, p_i.d_k \geq p_j.d_k$ **and**
3. $\exists d_l, \in d, p_i.d_l > p_j.d_l$

Definition 6, Skyline Queries on Incomplete Database: Select a data item p_i from the set of D incomplete database if (and only if) p_i is as good as p_j (where $i \neq j$) in all common non-missing dimensions and strictly better than p_j in at least one common non-missing dimension. We use *IncoDskyline* to denote the set of skyline data items on an incomplete database, $IncoDskyline = (p_i \forall p_i, p_j \in D, p_i \succ p_j)$.

Definition 5, Comparable: Let the data items a_i and $a_j \in R$, a_i and a_j are comparable (denoted by $a_i \varepsilon a_j$) if (and only if) they have no missing values in at least one identical dimension; otherwise ai is incomparable to a_j (denoted by $a_i \varepsilon / a_j$).

IV. THE PROPOSED FRAMEWORK

In this section, the components of the proposed framework (*Optimized Incomplete Skyline*) or OIS for processing skyline queries in a database with incomplete data are explained. The proposed framework consists of four components, namely: i) clustering, ii) sorting and filtering, iii) local skylines identifier, and iv) skylines identifier. The main purpose of the framework is to simplify the skyline process in a database with incomplete data. The simplification of the skyline process aims at reducing the number of domination tests between the data items when generating the skylines. Figure 2 illustrates the components and the process flow of the *OIS* framework.

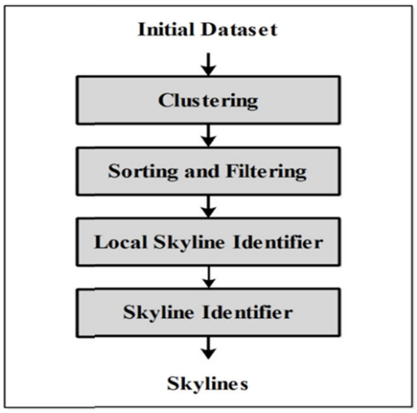


FIGURE 2. The proposed framework.

***Algorithm* 1**

Input: initial incomplete database, $D = \{a_1, a_2, a_3, ..., a_n\}$
Output: A set of clusters, $C = \{C_1, C_2, ..., C_m\}$

```
1. foreach data item $a_i$ in $D$ do
2.     if the bitmap representation of $a_i$ == the bitmap
       representation of an existing cluster $C_j$ then
3.         insert $a_i$ into $C_j$
4.     else
5.         create a new cluster, $C_k$
6.         insert $a_i$ into $C_k$
7.     end
8. end
```

FIGURE 3. Algorithm of clustering.

To ensure a better understanding of how the proposed framework works, a running example of incomplete database is used as demonstrated in Figure 4. The database example comprises of 40 data items with four dimensions. The symbol (*) is used to denote that the value of the corresponding dimension is not available (missing).

A. CLUSTERING

This component is used to aggregate all data items in a cluster taking into consideration the common comparable dimensions (shared namespace). Clustering is considered as an important step as it facilitates the process of pairwise comparisons and assures that the *transitivity property* of the skyline is sustained. Most importantly, clustering assists in preventing the issue of *cyclic dominance*.

The initial database with incomplete data is scanned and divided into smaller subsets (clusters) based on the bitmap representation of the data items. The bitmap representation pattern represents data items as a set of binary bits (0 or 1). If the value of the dimension exists, it is represented by 1; otherwise, it is represented as 0.

Figure 3 illustrates the steps of the clustering algorithm as adopted from [14]. Each data item is read in sequence (step 1); if the bitmap representation of a data item matches the bitmap representation of any of the existing clusters (step 2), then the data item is inserted into the corresponding cluster (step 3). However, if it does not match, a new cluster is created (step 5) followed by inserting the data item into the created cluster (step 6). This process is repeated over the entire dataset D, and the number of clusters generated is based on the corresponding bitmap representation patterns of the data items.

TIME COMPLEXITY ANALYSIS OF ALGORITHM 1

Let $|D|$ denote the size of data base D, i.e., the number of data items in D, and let $|a_i|$ denote the dimensionality of a data item a_i. Since $D = \{a_1, a_2, \ldots, a_n\}$ and each a_i is d-dimensional, $|D| = n$ and $|a_i| = d$. Similarly, we denote the size of C (the number of the clusters in C) and size of C_i (the number of data items in C_i) by $|C|$ and $|C_i|$, respectively. Since the clusters are created according to missing dimensions, the number of clusters

$$1 \leq |C| \leq 2^d$$

Though the dimensionality d is much less than the number of data items n, $d \ll n$, in practice, for a suitably larger d, the number 2^d may be a large number. Therefore, when analyzing this algorithm and the following algorithms, we need to take into consideration the effect of 2^d to the time complexity of the algorithms. As the bitmap representations of a data item a_i and a cluster C_j are order of d, each bitmap comparison takes a constant time. The total number of bitmap comparisons, BC, depends on the number data items and the number of clusters. Thus,

$$t \times n \leq BC\ (n, d) \leq t \times n \times 2^d$$

where $t > 0$ is a constant number.

The construction of each cluster C_j takes a constant time, since it is initially an empty list, and the total number of the constructions, CC, is

$$1 \leq CC\ (d) \leq 2^d$$

The insertion of a data item a_i into a cluster C_j takes $\mathrm{O}\ (d)$ time (a constant time). The total number of insertions, INS, is an order of n that is

$$INS\ (n) = \mathrm{O}\ (n)\,.$$

Let $T_1\ (n, d)$ denote the running time of ***Algorithm 1***. Then,

$$T_1\ (n, d) = BC\ (n, d) + CC\ (d) + INS\ (n)$$

In the best case:

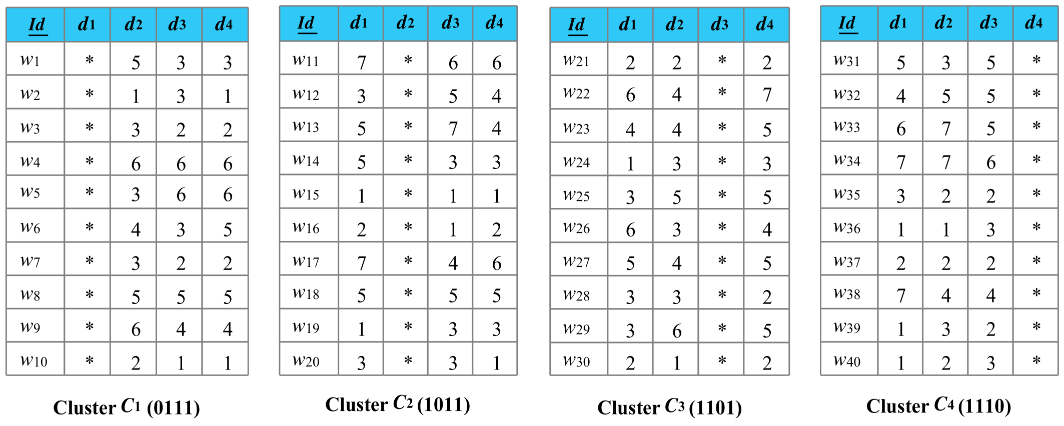
$$T_{1,best}\ (n, d) = \mathrm{O}\ (n) + O\ (1) + \mathrm{O}\ (n) = \mathrm{O}\ (n)$$

In the worst case:

$$\begin{aligned} T_{1,worst}\ (n, d) &= \mathrm{O}\left(n \times 2^d\right) + \mathrm{O}\left(2^d\right) + \mathrm{O}\ (n) \\ &= \mathrm{O}\left(n \times 2^d\right) \end{aligned}$$

Thus,

$$\mathrm{O}\ (n) \leq T_1\ (n, d) \leq O\left(n \times 2^d\right)$$



Id	d_1	d_2	d_3	d_4
w_1	*	5	3	3
w_2	*	1	3	1
w_3	*	3	2	2
w_4	*	6	6	6
w_5	*	3	6	6
w_6	*	4	3	5
w_7	*	3	2	2
w_8	*	5	5	5
w_9	*	6	4	4
w_{10}	*	2	1	1

Cluster C_1 (0111)

Id	d_1	d_2	d_3	d_4
w_{11}	7	*	6	6
w_{12}	3	*	5	4
w_{13}	5	*	7	4
w_{14}	5	*	3	3
w_{15}	1	*	1	1
w_{16}	2	*	1	2
w_{17}	7	*	4	6
w_{18}	5	*	5	5
w_{19}	1	*	3	3
w_{20}	3	*	3	1

Cluster C_2 (1011)

Id	d_1	d_2	d_3	d_4
w_{21}	2	2	*	2
w_{22}	6	4	*	7
w_{23}	4	4	*	5
w_{24}	1	3	*	3
w_{25}	3	5	*	5
w_{26}	6	3	*	4
w_{27}	5	4	*	5
w_{28}	3	3	*	2
w_{29}	3	6	*	5
w_{30}	2	1	*	2

Cluster C_3 (1101)

Id	d_1	d_2	d_3	d_4
w_{31}	5	3	5	*
w_{32}	4	5	5	*
w_{33}	6	7	5	*
w_{34}	7	7	6	*
w_{35}	3	2	2	*
w_{36}	1	1	3	*
w_{37}	2	2	2	*
w_{38}	7	4	4	*
w_{39}	1	3	2	*
w_{40}	1	2	3	*

Cluster C_4 (1110)

FIGURE 4. **Clusters.**

According to our running database example, the initial database relation D is partitioned into four distinct clusters according to the constructed patterns of the bitmap representation. Notice that the bitmap representation pattern of the cluster C_1 is 0111 whereby all data items of the cluster have no values in the first dimension, d_1. Similarly, the bitmap representations of clusters C_2, C_3, and C_4 are 1011, 1101, and 1110 respectively as demonstrated in Figure 4.

B. SORTING AND FILTERING

This component is an essential component that contributes toward simplifying the skyline process in an incomplete database. The process relies on removing the dominated data items in each cluster before the skyline process. This component is very beneficial and helps reduce the domination tests and avoid many unnecessary pairwise comparisons, which in turn decreases the processing time of the skyline process. We incorporate a *threshold* value used to determine the desired skyline data items. For example, a user might be looking for the best action movies (skyline data items) having a rating of 3 and above on the MovieLens website. In such a case, we have to filter out the dataset before applying the skyline technique.

The skyline process performs comparisons at the dimension level. Thus, by exploiting the concept of the skyline technique, our approach uses the *threshold* on each dimension. In this way, there is a zero per cent chance of having dominated data items in a skyline set. It should be noted that the idea of using a *threshold* assists in identifying the superior skyline data items out of the total candidate skyline data items in the database. Applying the concept of *threshold* value will not result into a false negative or false positive. Thus, we can always guarantee that at the end of the process the return skyline data items are complete and correct. In our approach, prior to the filtration technique, the data items of all clusters are kept in a non-ascending order of each dimension with no missing values. Also, a set of arrays (AL) is constructed to store the*Ids* of data items instead of their dimension value. It is important to note here that sorting technique helps rearrange the data items in such a way that all the superior data items are placed on top of the constructed list, which makes the filtration process easier and quicker. For each dimension of an array list(s) (AL_i), we select only those data items that have the highest value. Notice that all clusters are independent of each other, which means that the selection process can be carried out simultaneously. This helps in accelerating the process. By selecting only those data items that have the best value in any of the non-missing dimensions, $u_1, u_2, \ldots u_n$ results into considering only those data items that have a high potential to be the skyline data items. This is due to the fact that the data items with the best value assist in dominating a large number of data items in that particular dimension. Eventually, the best data items of all clusters for each dimension are selected for the next process. At this point, the remaining data items are discarded and can be safely removed from further processes. The selected data items are stored in a new list called candidate list (CL_i) of each array list. It is important to note that the storage of the data items in a candidate list is carried out column-wise, and repeating the data items is not allowed. In that way, the candidate list(s) of each array list (clusters) contains only the filtered data item(s).

Figure 5 details the steps of the sorting and filtering algorithm. The input of the algorithm is a set of clusters that have been generated in the previous step, while the output of the algorithm is a set of candidate lists containing only the superior data items in each cluster. It should be noted that the data items present in different clusters ($C_i - C_n$) are sorted in non-ascending order based on each non-missing dimension (d_i) and their *id*s and are stored in array lists ($AL_i - AL_n$) (step 1-5). Then, a user given threshold value (th) is identified to denote the limit of the superior data items a user is looking for (step 6). The value of the first data item represented by its *Id*present in the first column (u_j) of AL_i is being checked to determine whether it is missing or not (step 8). If the value is missing, then the loop continues to the next column (u_{j+1}) of array list (AL_i) (step 9). The value of threshold, *th*is assigned to a counter variable, *SuperiorDataItems* (step 11).

```
Algorithm 2
Input: A Set of Cluster List, $C = (C_1, C_2, C_3, ..., C_m)$
Output: Candidate List $CL = (CL_1, CL_2, CL_3, …, CL_m)$
1.   foreach $C_i$ in $C$ do
2.   foreach $d_j$ of $C_i$ do
3.      sort DataItems in non-ascending order based on available values
          in $d_j$
4.      add the indices $id$ of the sorted DataItems based on $d_i$ to Array
          List $AL_i.u_j$
5.   end
6.   set $th$ = User Given threshold
7    foreach column $u_j$ in $AL_i$ do
8.      if value of $u_j.a(id_k)$ is missing then //  a is DataItem in $u_j$
9.          continue;
10.     endif
11.     set SuperiorDataItems = th
12.     foreach $id_k$ in $u_j$ do // id represents the id of a DataItem
13.         if SuperiorDataItems == 0 then
14.             break;
15.         else
16.             if DataItem $a$ of $id_k$ does not exist in $CL_i$ then
17.                 insert $a$ of $id_k$ into $CL_i$
18.             endif
19.             if value of DataItem $u_j.a(id_k)$ != value of $u_j.a(id_{k++})$ then
20.                 SuperiorDataItems = SuperiorDataItems -1
21.             endif
22.         endif
23.     end
24.  end
25. return Candidate list $CL_i$
```

FIGURE 5. Algorithm of sorting and filtering.

The variable *SuperiorDataItems*is checked; if it is equal to zero, then the inner loop (step 13) stops and the control moves to the outer loop (step 14). Otherwise, the data item of the corresponding id_k will be checked to determine whether it is present in the candidate list (CL_i) or not (step 16). If the data item does not exist in CL_i, then the data item will be added to the corresponding candidate list CL_i (step 17). If the value of the non-missing dimension u_j of the data item of id_k is not equal to the value of non-missing dimension u_j of the next data item id_{k++} (step 19), the value of *SuperiorDataItems*is decreased by 1 (step 20). This process continues to find all the superior data items in the non-missing dimension u_j (steps 12 -23) in each array list AL_i. Steps 7 to 24 are repeated to find the superior data items in all non-missing dimensions in each array list AL_i. The process ends by returning the candidate list(s) which contain the superior data items of each cluster (step 25).

TIME COMPLEXITY ANALYSIS OF ALGORITHM 2

Here, we calculate the running time of the sorting algorithm, $SORT$, for all clusters C_i in C in total. Since we can use any efficient comparison-based sorting algorithm, for each dimension d_j in each cluster C_i, the sorting takes $\mathrm{O}(|C_i| \times \log |C_i|)$ time, i.e.,

$$SORT\left(|C_i|, d_j\right) = \mathrm{O}(|C_i| \times \log |C_i|)$$

Then for all dimensions $1 \le d_j \le d$ in each cluster C_i, the sorting takes $\mathrm{O}(d \times |C_i| \times \log |C_i|)$ time, i.e.,

$$SORT(d, |C_i|) = \mathrm{O}(d \times |C_i| \times \log |C_i|)$$

For all clusters C_i in C,

$$SORT(n, d) = \sum_{i=1}^{|C|} \mathrm{O}(d \times |C_i| \times \log |C_i|)$$

Here, we have two extreme cases: when we have only one cluster, i.e., $|C| = 1$ and when we have all possible clusters, i.e. $|C| = 2^d$. Then

$$SORT_{best}(n, d) = \mathrm{O}(d \times n \times \log n),$$

And

$$SORT_{worst}(n, d) = \mathrm{O}\left(2^d \times n \times \log n\right)$$

The creation of an array list AL_i takes $\mathrm{O}(1)$ time, thus, the total amount of time, $CONSTAL$, spent to construct all AL_is is the order of the number of the clusters in C, which means

$$\mathrm{O}(1) \le CONSTAL(d) \le \mathrm{O}\left(2^d\right)$$

The total number of insertions, INS, of ids of the data items in each cluster C_j for each dimension d_i into AL_j is the order of the size of the cluster, that is,

$$INS\left(\left|C_j\right|, d\right) = \mathrm{O}\left(\left|C_j\right| \times d\right)$$

Regardless the number of the clusters, we insert all data items in D into the Array Lists AL_is, thus,

$$INSAL(n, d) = \mathrm{O}(n \times d)$$

Depending on the value of the threshold th and the nature of the data items in D, we may insert from $\mathrm{O}(1)$ to $\mathrm{O}(n)$ number of data items into each candidate list CL_i. The insertion of each data item a_i into the corresponding candidate list CL_j takes $\mathrm{O}(d)$ time. Then, the total number of the insertions

$$INSCL(n, d) = \mathrm{O}(n \times d).$$

Finally, the total amount of the running time of ***Algorithm* 2** is

$$T_2(n, d) = SORT(n, d) + CONSTAL(n, d) + INSAL(n, d) + INSCL(n, d)$$

In the best case:

$$T_{2,best}(n, d) = T_2(n) = \mathrm{O}(n \times \log n) + \mathrm{O}(1) + \mathrm{O}(n) + \mathrm{O}(n) = \mathrm{O}(n \times \log n)$$

In the worst case:

$$T_{2,worst}(n, d) = \mathrm{O}\left(2^d \times n \times \log n\right) + \mathrm{O}\left(2^d\right) + \mathrm{O}(n) + \mathrm{O}(n) = \mathrm{O}\left(2^d \times n \times \log n\right)$$

According to our running database example, the data items in clusters C_1, C_2, C_3 and C_4 are sorted in non-ascending order for each dimension. A set of 2D array list is constructed (AL_1, AL_2, AL_3, AL_4) and only *Id*'s of data items of cluster C_1, C_2, C_3 and C_4 are stored in array lists respectively.

AL_1

u_1	u_2	u_3	u_4
*	w_4	w_4	w_4
*	w_9	w_5	w_5
*	w_1	w_8	w_6
*	w_8	w_9	w_8
*	w_6	w_1	w_9
*	w_3	w_2	w_1
*	w_5	w_6	w_3
*	w_7	w_3	w_7
*	w_{10}	w_7	w_1
*	w_2	w_{10}	w_{10}

AL_2

u_1	u_2	u_3	u_4
w_{11}	*	w_{13}	w_{11}
w_{17}	*	w_{11}	w_{17}
w_{13}	*	w_{12}	w_{18}
w_{14}	*	w_{18}	w_{12}
w_{18}	*	w_{17}	w_{13}
w_{12}	*	w_{14}	w_{14}
w_{20}	*	w_{19}	w_{19}
w_{16}	*	w_{20}	w_{16}
w_{15}	*	w_{15}	w_{15}
w_{19}	*	w_{16}	w_{20}

AL_3

u_1	u_2	u_3	u_4
w_{22}	w_{29}	*	w_{22}
w_{26}	w_{25}	*	w_{23}
w_{27}	w_{22}	*	w_{25}
w_{23}	w_{23}	*	w_{27}
w_{25}	w_{27}	*	w_{29}
w_{28}	w_{24}	*	w_{26}
w_{29}	w_{26}	*	w_{24}
w_{21}	w_{28}	*	w_{21}
w_{30}	w_{21}	*	w_{28}
w_{24}	w_{30}	*	w_{30}

AL_4

u_1	u_2	u_3	u_4
w_{34}	w_{33}	w_{34}	*
w_{38}	w_{34}	w_{31}	*
w_{33}	w_{32}	w_{32}	*
w_{31}	w_{38}	w_{33}	*
w_{32}	w_{31}	w_{38}	*
w_{35}	w_{39}	w_{36}	*
w_{37}	w_{35}	w_{40}	*
w_{36}	w_{37}	w_{35}	*
w_{39}	w_{40}	w_{37}	*
w_{40}	w_{36}	w_{39}	*

FIGURE 6. List of sorted arrays.

Figure 6 shows the list of constructed arrays of the running database example.

The filtration process starts by choosing only those data items that have highest value(s) in each dimension(u_i) of array lists (AL_i). In order to start the filtration process, a threshold (th) is needed to set according to users' preferences. In this example, we set $th = 2$ which means we choose the two top-valued data items from each dimension. Hence, from AL_1 in dimension $d_2(u_2) w_4$, w_9, w_1, and w_8 have been selected as the data items among top two values in d_2. Similarly, w_4, w_5, w_8 have been chosen from $u_{3,}$ and w_4, w_5, w_6, w_8 have been nominated from u_4. However, none of the data items have been selected from u_1 since all data items have a missing value in d_1. After selecting the superior data items, they are placed in a candidate list (CL_1) for further processing. It is important to note that the selected data items cannot be repeated. In other words, once a data item has been added to the candidate list it will not be added again later.

The process of filtration is carried out in parallel and simultaneously for all array lists. Hence, from AL_2 the superior data items w_{11}, w_{17}, w_{13}, w_{14}, and w_{18} are selected and stored in CL_2. Similarly, the candidate list CL_3 contains data items w_{22}, w_{26}, w_{27}, w_{29}, w_{25}, and w_{23}. Lastly, the candidate list, CL_4 contains five data items, w_{34}, w_{38}, w_{33}, w_{32} and w_{31} as shown in Figure 7.

CL_1

Id	d_1	d_2	d_3	d_4
w_4	*	6	6	6
w_9	*	6	4	4
w_1	*	5	3	3
w_8	*	5	5	5
w_5	*	3	6	6
w_6	*	4	3	5

CL_2

Id	d_1	d_2	d_3	d_4
w_{11}	7	*	6	6
w_{17}	7	*	4	6
w_{13}	5	*	7	4
w_{14}	5	*	3	3
w_{18}	5	*	5	5

CL_3

Id	d_1	d_2	d_3	d_4
w_{22}	6	4	*	7
w_{26}	6	3	*	4
w_{27}	5	4	*	5
w_{29}	3	6	*	5
w_{25}	3	5	*	5
w_{23}	4	4	*	5

CL_4

Id	d_1	d_2	d_3	d_4
w_{34}	7	7	6	*
w_{38}	7	4	4	*
w_{33}	6	7	5	*
w_{32}	4	5	5	*
w_{31}	5	3	5	*

FIGURE 7. Candidate lists of each cluster.

As shown in the running database example, four data items have been removed from cluster 1 (0111), which is 40% of data items reduced before applying the skyline technique. Similarly, from cluster 2 (1011) five data items have been removed which amounts to 50% reduction in the number of data items. The reduction in the amount of data will significantly decrease the number of domination tests required during the skyline process. Likewise, in cluster 3 (1101) and cluster 4 (1110), we have successfully managed to reduce up to 40% and 50% of data items respectively before applying the skyline technique. Therefore, we conclude that our sorting and filtering processes have successfully simplified the skyline process and eliminated up to 18 data items, which represents around 45% of the total size of the initial database.

C. LOCAL SKYLINE IDENTIFIER

This component is used to determine the local skyline data items of each candidate list. Since the candidate lists are independent from each other, the process of identifying the local skyline data items can be performed simultaneously. The data items present in one candidate list, CL_i are compared against each other to compute the local skyline data items of the candidate list, CL_i. This process is conducted concurrently on data items present in other candidate lists ($CL_{i+1}, \ldots, CL_n$). The main purpose here is to speed up the skyline process by running multiple processes in parallel. Pairwise comparisons between data items result in the elimination of the dominated data items in each candidate list, and only the non-dominated data items will be considered for the next stage. It must be noted here that the issue of *cyclic dominance* and loss of the *transitivity property* of the skyline technique will no longer occur even with missing values since the data items of each candidate list have similar bitmap representation, while those dimensions with missing value have no impact on skyline process. Performing pairwise comparison between the data

```
Algorithm 3
Input: A Set of Candidate List, $CL = (a_1, a_2, a_3, ..., a_m)$
Output: Local Skylines of each Candidate List $CL$
1.  foreach $a_j$ of $CL_i$ do
2.      $P = CL_i.a_j$
3.      $IsDominated$ = false
4.      for (int $k = j+1$; $k <$ length($CL_i$); $k$++)
5.          $Cw = CL_i.a_k$
6.          $DomValue$ = Compare ($P$, $Cw$) // Algorithm3
7.          if $DomValue == 1$ then
8.              remove $Cw$ from $CL_i$
9.          elseif $DomValue == 2$ then
10.             $IsDominated$ = true
11.         end
12.     end
13.     if $IsDominated$ then
14.         remove $P$ from $CL_i$
15.     end
16. end
17. return Local skylines of each Cluster ($CL_i$)
```

FIGURE 8. Algorithm for identifying local skyline data items.

items of each candidate list before aggregating them in one single list helps avoid redundant pairwise comparisons and reduces the number of data items to be considered in the next step. If a data item of one candidate list is dominated and removed by another data item in the same candidate list, the dominated data item will not be included in the skyline result. Thus, its timely removal allows us to reduce the processing time of the skyline query.

Figure 8 details the steps of the algorithm for processing local skyline data items for each candidate list (CL). The algorithm starts with selecting the first data item a_j of CL_i for processing (step 1). Then, the data item a_j is assigned to a pointer variable, P (step 2). In step 5 the second data item of the CL_i is assigned to another pointer variable, Cw. In step 6 a *compare* method is called to compare P with Cw, and P, Cw are passed as parameters. If the method returns 1, that indicates P has dominated Cw (step 7) and Cw is removed from cluster CL_i (step 8). However, if the method returns 2, then it is clear that Cw dominates P (step 9), and the Boolean variable *IsDominated* is set to TRUE. It is very important to note that P is not eliminated immediately if it has been dominated by Cw since P has a high potential to dominate other data items in the candidate list. Thus, without losing the generality of the skylines, P is compared with the rest of remaining data items in CL_i, and steps from 4 to 13 are repeated. At the end of the first iteration *IsDominated* is checked to determine whether P has been dominated by any data item or not; if so (step 13), then P is removed from CL_i (Step 14). This process continues until all remaining data items of CL_i have been compared with each other (steps 1 to 16). At the end of the process each candidate list ($CL_1 - CL_n$) contains the local skyline data items that are not dominated by any data item in that particular candidate list. It is important to note that the local skyline identifier algorithm processes all candidate lists ($CL_1 - CL_n$) simultaneously as all candidate lists are independent from each other. Hence, this step speeds up the process of computing skyline and generates the local skyline data items of all clusters at the same time.

```
Algorithm 4
Input: Two data item $P$ and $Cw$
Output: An integer value, (0, 1, or 2)
1.  if data item $P \succ C$ then
2.      return 1
3.  elseif data item $C \succ P$ then
4.      return 2
5.  else
6.      return 0
7.  end
```

FIGURE 9. The skyline algorithm.

1) TIME COMPLEXITY ANALYSIS OF ALGORITHM 3

Without loss of generality, we consider the case in where $|CL_i| = \mathrm{O}(n)$. if $|CL_i| = \mathrm{O}(1)$, then the all operations in the algorithm also take $\mathrm{O}(1)$ time.

First, the total number of the assignments of the data items to P is $|CL_i|$ which means

$$ASSIGNP\,(n) = \mathrm{O}\,(n)$$

Second, the total number of the assignments of the data items to Cw is

$$ASSIGNCW\,(n) = \sum_{j=1}^{|CL_i|-1} \sum_{k=j+1}^{|CL_i|} 1 = \sum_{j=1}^{|CL_i|} (|CL_i| - j) = \mathrm{O}\left(|CL_i|^2\right) = \mathrm{O}\left(n^2\right)$$

Similarly, each comparison of P with Cw takes $O\,(d)$ time, and the total number of comparisons is

$$COMPPCW\,(n, d) = \mathrm{O}\left(d \times n^2\right).$$

The removal of dominated Ps and Cws may take at most $\mathrm{O}\,(n)$ time. Thus, the total running time of ***Algorithm* 3** is

$$T_3\,(n, d) = ASSIGNP\,(n) + ASSIGNCW\,(n) + COMPPCW\,(n, d) = \mathrm{O}\left(d \times n^2\right)$$

Or simply,

$$T_3\,(n) = \mathrm{O}\left(n^2\right)$$

Figure 9 illustrates the detailed steps of the compare function algorithm that compares two data items and generates the result of the comparison. In step 1 the data items, P and Cw are compared with each other based on the comparable dimensions with non-missing values. If the data item P dominates Cw, then the return value is 1 (step 2). However, if Cw dominates P (step 3), then the return value is 2 (step 4) indicating that P is dominated by Cw. Otherwise, return 0 which denotes that neither P dominates Cw nor Cw dominates P (step 6).

2) TIME COMPLEXITY ANALYSIS OF ALGORITHM 4

The time complexity of ***Algorithm* 4** is very obvious. The comparison of two data items takes $\mathrm{O}\,(d)$ time.

Based on our running database example, the data items present in the candidate list, CL_1 are compared with

CL_1

Id	d_1	d_2	d_3	d_4
w_4	*	6	6	6

CL_2

Id	d_1	d_2	d_3	d_4
w_{11}	7	*	6	6
w_{13}	5	*	7	4

CL_3

Id	d_1	d_2	d_3	d_4
w_{22}	6	4	*	7
w_{29}	3	6	*	5

CL_4

Id	d_1	d_2	d_3	d_4
w_{34}	7	7	6	*

FIGURE 10. Local skyline data items of candidate lists.

each other. The process works as follows: The first data item, w_4 in CL_1 is read and compared with the next data item, w_9. If w_4 dominates w_9, then w_9 will be removed immediately from the candidate list, CL_1. However, if w_9 dominates w_4 then w_4 will only be removed from list at the end of the iteration as we believe that w_4 has a high potential to eliminate more data items in CL_1. Similarly, in the following iterations w_4 is compared with rest of the data items in $CL_1(w_1, w_8, w_5, w_6)$.

The data items dominated by w_4 will no longer remain in CL_1 and will be excluded from further processing. Then, the next remaining data item in CL_1 is selected, and in this case is w_9 and it will be compared with the remaining data items of CL_1. This process continues until all the remaining data items are compared with each other. At the end of the process the remaining data items in CL_1 will be reported as the local skyline data items of CL_1. This process is conducted in parallel for the other candidate lists, CL_2, CL_3, and CL_4. Figure 10 depicts the local skyline data items of the candidate lists of the database example. Notice that only one data item has been retrieved as local skyline from candidate list, CL_1. Similarly, from candidate list CL_2 only two data items have been selected as local skylines, from CL_3 and CL_4 only two and one data items have been chosen as local skylines. Hence, many dominated data items have been removed. This early pruning of the dominated data items greatly improves the skyline process and reduces the number of pairwise comparisons and reduces the cost of the skyline query.

D. SKYLINE IDENTIFIER

The purpose of this component is to return the final skyline data items of the entire database. This is achieved by comparing the reported local skyline data items of all candidate lists with each other to form the final skylines. The local skyline data items of each candidate list, CL_i are compared against the local skyline data items of other candidate lists, $CL_{i+1}, \ldots, CL_n$. This ensures that the local skyline data items of each candidate list are compared with the local skyline data items of other candidate lists to ascertain that the reported skyline data items are the global skyline data items of the entire database. This ensures that the output is always correct, and no data item has been left out. Due to the lack of some values resulting in different bitmap representation patterns, not all local skyline data items of the candidate lists are comparable with each other. Thus, certain issues such as *cyclic dominance* and losing *transitivity property* have to be considered. Therefore, we cannot simply remove dominated data items from the candidate list if they are being dominated by other comparable data items unless we have ensured that these removed data items are strictly worse than the reported skyline data items. Otherwise, the result might be invalid and produces an incorrect set of skyline data items. This issue must always be resolved when processing skyline queries with incomplete data. For instance, if a data item, p, dominates another data item, q, and q dominates r, it is not a necessity that p will also dominate r as each data item has dimensions with missing values. Therefore, the incompleteness of the data results in some data items being incomparable, and the transitivity property of the skyline technique is no longer guaranteed. Thus, we have to carefully check that each data item is a skyline over the entire database before reporting the result to the end user.

Algorithm 5

Input: Candidate ($CL_1 - CL_m$) lists of each cluster ($C_1, C_2, \ldots, C_m$)
Output: Final Skylines S

```
1.  foreach $CL_i$ do
2.    foreach DataItem $a_j$ of $CL_i$ do // $j = (1, 2, \ldots, n)$
3.      $P = CL_i.a_j$
4.      $IsDominated$ = false
5.      foreach $CL_{i++}$ do
6.        foreach DataItem $a_k$ of $CL_{++}$ do  // $k = (1, 2, \ldots, n)$
7.          $Cw = CL_{i++}.a_k$
8.          $DomValue$ = Compare $(P, Cw)$ // algorithm 3
9.          if $DomValue == 1$ then
10.             remove $Cw$ from $CL_{i++}$
11.         elseif $DomValue == 2$ then
12.             $IsDominated$ = true
13.         end
14.       end
15.     end
16.     if $IsDominated$ then
17.         remove $P$ from $CL_i$
18.     end
19.   end
20. end
21. return Final Skylines $S$
```

FIGURE 11. Algorithm for final skylines.

Figure 11 depicts the steps of the algorithm for processing local skyline data items for each candidate list (CL). In this algorithm local skyline the data items of each candidate list are taken as input to compute the final skyline data items as output. This is achieved by performing domination tests across the candidate lists. The process starts by assigning the first data item in the candidate list, CL_1 to a pointer variable, P (steps 1-3). Then the first data item of the next candidate list, CL_2 is assigned to a pointer variable, Cw (steps 6 and 7). These two data items are sent as parameters to a *compare* method (step 8). If P dominates Cw then Cw is eliminated immediately from the candidate list (CL_2) (step 10). However, if Cw dominates P then $IsDominated$ is set as *true* (step 12) and the process of comparing P with the remaining data items of all candidate lists ($CL_{i+1} - CL_n$) continues (steps 5 to 14 are repeated). If P has been dominated at the end of the first iteration by any other data item present in any candidate list (step 16), then $P$is removed from the candidate list (step 17). To compare each remaining data item of each candidate list ($CL_i - CL_n$) with one another, steps 1 to 20 are repeated. Thus, at the end of the process, the remaining data items contained in the candidate lists are considered as skyline data items since they have not been dominated by other data items in the database.

TIME COMPLEXITY ANALYSIS OF ALGORITHM 5

Since the analysis of Algorithm 5 is very similar to the analysis of Algorithm 3, we use the similar arguments to establish the running time of the algorithm.

For all CL_is and for all data items a_js in each CL_i, the total number of assignments to P is

$$ASSIGNP\,(n) = n - |CL_m| = \mathrm{O}\,(n)$$

The total number of the assignments of the data items to Cw is

$$\begin{aligned} &ASSIGNCW\,(n, d) \\ &= \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{k=1}^{|CL_j|} 1 = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} |CL_j| \\ &= \sum_{i=1}^{m-1} (|CL_{i+1}| + \cdots + |CL_m|) \\ &= 1 \times |CL_2| + 2 \times |CL_3| + \cdots + (m-1) \times |CL_m| \end{aligned}$$

Further, we need to consider the summation above with respect to two parameters n and d.
If for all $j = 2, 3, \ldots, m$, $\left|CL_j\right| = \mathrm{O}\,(n)$, then

$$\begin{aligned} ASSIGNCW\,(n, d) &= \mathrm{O}\,(n) \times (1 + 2 + \cdots + (m-1)) \\ &= \mathrm{O}\,(n) \times O\left(m^2\right) \end{aligned}$$

If for all $j = 2, 3, \ldots, m$, $\left|CL_j\right| = \mathrm{O}\,(1)$, then

$$\begin{aligned} ASSIGNCW\,(n, d) &= \mathrm{O}\,(1) \times (1 + 2 + \cdots + (m-1)) \\ &= \mathrm{O}\left(m^2\right) \end{aligned}$$

Since $1 \leq m \leq 2^d$, we have two possible extreme cases: In the best case, when m is a small constant, and in the first case, we have

$$ASSIGNCW\,(n) = \mathrm{O}\,(n)$$

And, in the second case,

$$ASSIGNCW\,(n) = \mathrm{O}\,(1)$$

In the worst case, when m is order of 2^d, in the first case,

$$ASSIGNCW\,(n) = \mathrm{O}\left(n \times 2^{2d}\right)$$

And, in the second case,

$$ASSIGNCW\,(n) = \mathrm{O}\left(2^{2d}\right)$$

Correspondingly, we execute the same number of $\mathrm{O}\,(d)$ comparisons for all assignments of Cw. Summarizing all cases, we obtain the following estimations for the running time of **Algorithm 5**, $T_5(n, d)$:

$$\mathrm{O}\,(1) \leq \mathrm{O}\left(2^{2d}\right) \leq T_5\,(n, d) \leq \mathrm{O}\,(n) \leq \mathrm{O}\left(n \times 2^{2d}\right)$$

From the database example run that has been used throughout the study, the data item present in first candidate list CL_1 is selected first and is compared against all local skyline data items of candidate lists, CL_2, CL_3, and CL_4. Comparing w_4 with w_{11} and w_{13} indicates that both data item of CL_2 are not worse than w_4 and will not be removed. Similarly, w_4 is compared with the local skyline data items of CL_3, w_{22} and w_{29}, and we notice that only w_{29} has been dominated by w_4 and is thus removed from the candidate list. At last w_4 is further compared with data item of CL_4, w_{34}. We notice that w_{34} dominates w_4. Therefore, at the end of the first iteration w_4 is removed from the candidate list CL_1. Once CL_1 is empty the data items of CL_2 are compared with the remaining data items of other candidate lists (CL_3 and CL_4). Hence, w_{11} is selected from CL_2 and compared with the remaining data items of CL_3 and CL_4 by comparing w_{11} with w_{22} of CL_3 and w_{34} of CL_4. Since, none of them are dominating each other, w_{13} is selected for processing in the next iteration and is compared with w_{22} and w_{34}. However, w_{22} dominates w_{13} and is thus removed from CL_2 at the end of the iteration. In the last iteration the data item(s) of CL_3, (w_{22}) are compared with the data item of CL_4, and the result of the comparison indicates that w_{34} dominates w_{22} which is thus removed from the list.

Id	d_1	d_2	d_3	d_4
w_{11}	7	*	6	6
w_{34}	7	7	6	*

FIGURE 12. **Final skyline data items.**

At this point, the process of identifying the final skyline has reached a stage where all available local skyline data items of each candidate list are compared with other candidate lists. Dominated data items are removed from the lists, and the remaining data items are reported as the final skyline data items of the entire database. Figure 12 illustrates that w_{11} and w_{34} are reported as the final skyline data items of the entire database.

We consider our proposed approach as significantly improving the performance of the skyline process through exploiting the filtration technique. Most importantly, the skyline process at the cluster level is performed in parallel, which results in the reduction of the domination test as well as safeguarding the *transitivity property* of skyline technique and avoids the issue of *cyclic dominance*. Rather than comparing all remaining data items present in each candidate list with each other, we compare only the local skyline data items of the candidate lists. Based on the database example considered throughout the study, we notice that the local

TABLE 3. The parameter setting of real and synthetic datasets in the experiments.

	Parameter Settings		
Dataset	**Dataset Size (KB)**	**No. of Dimensions (*d*)**	**No. of Dimensions with missing values (d')**
Independent	100, 200,300, 400, 500, 600	5-13	4-12
Correlated	100, 200,300, 400, 500, 600	5-13	4-12
NBA	40, 80, 120, 160, 200	6-18	5-17
CoIL 2000	50, 100, 150, 200, 250, 300	4-22	3-21

skyline identifier stage assists in eliminating up to 16 out of 22 data items, which translates into a 72% reduction of the initial database. Based on our detailed analysis, we conclude that each component of our proposed *OIS* framework plays a vital role in determining the skyline data items in a database with incomplete data.

V. EXPERIMENT EVALUATION

To evaluate the efficiency and measure the effectiveness of our proposed framework (OIS) of optimized skylines in incomplete data, several experiments have been conducted using various synthetic and real datasets. The proposed work has been contrasted with the five latest related works in the area of processing skyline queries in incomplete data, namely SCSA [44], SPQ [40], *Incoskyline* [14] and SIDS [12], *Iskyline* [11]. It has been contended that skyline computation is a CPU-rigorous process [1], [9], [10], [12]–[14], [24]. Therefore, the performance metric used in the experiments consists of the number of pairwise comparisons as it is considered as the most influenced parameter in skyline query processing [9], [10], [12]–[14]. The conducted experiments focused on measuring the number of pairwise comparison and processing time with respect to the number of dimensions and the size of the database (data cardinality). Two sets of experiments have been performed. In the first set, the number of dimensions has been varied and the database size fixed. This set of experiment has aimed to investigate the impact of the number of dimensions on the computation of skyline data items. Similarly, the second set of experiment has been designed by fixing the number of dimensions and varying the database size. This set of experiment has been used to examine the effect of the database size on the computing skyline data items with the presence of incomplete data. In all experiments we have assumed that the skyline query is submitted to retrieve the skyline data items with the maximum values of all non-missing dimensions. Moreover, we have also assumed that the user given threshold value controlling the number of superior skyline data items varies between 10 and 100. Two different datasets have been considered, namely synthetic and real datasets. For the synthetic datasets, two data have been randomly generated as a set of independent and correlated data. Furthermore, from the real dataset, NBA and CoIL 2000 insurance company have been used. Table 3 summarizes the

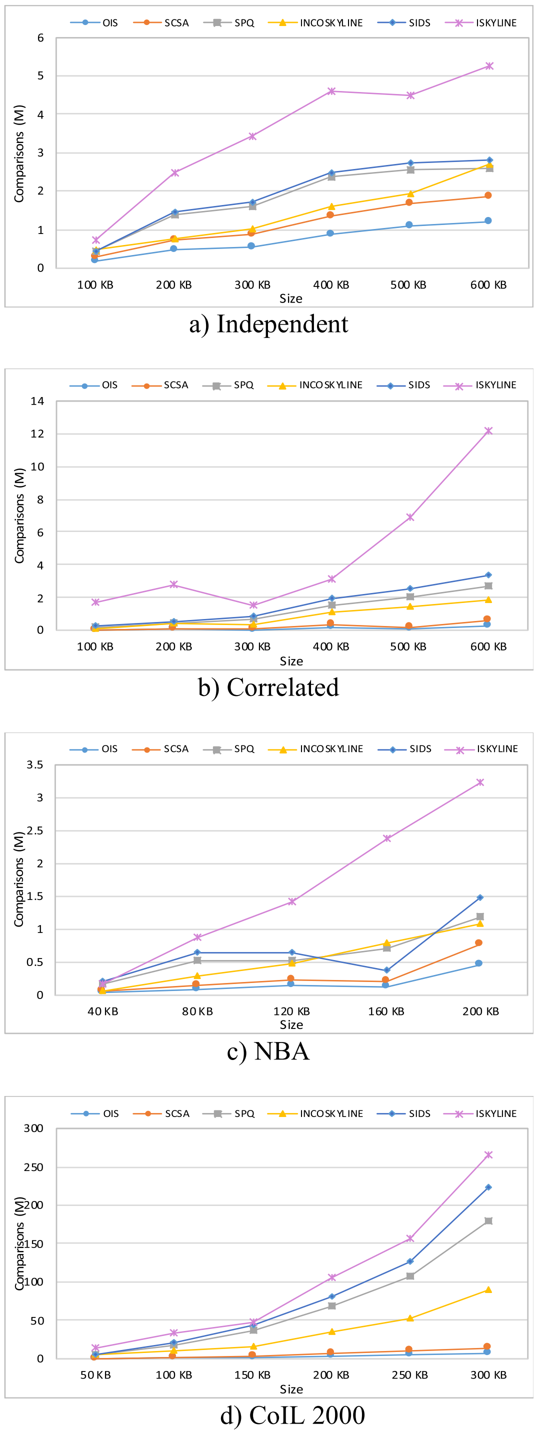


FIGURE 13. The effect of dataset size on the pairwise comparisons.

parameter setting of the experiments. The table outlines the range of the database size in KB, the total number of dimensions including the dimensions with missing and non-missing values. Lastly, the number of dimensions with missing values has also been reported.

A. DATA CARDINALITY

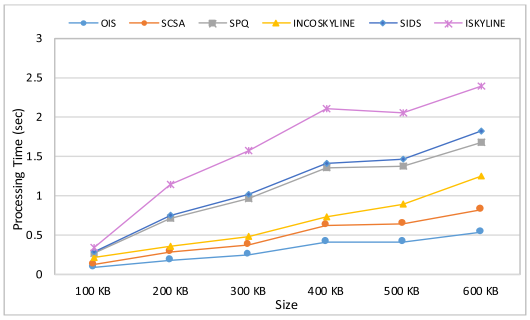
For this set of experiments, the database size has been varied and the number of dimensions fixed. Figure 13 illustrates the experimental results obtained for both datasets. Figure 13(a) shows the number of pairwise comparisons produced in our approach (*OIS*) and the other approaches including SCSA [44], SPQ [40], *Incoskyline* [14] SIDS [12], and *Iskyline* [11] for independent datasets. The number of dimensions has been fixed to eight dimensions including those dimensions with missing and non-missing values and the database size varying from 100KB to 600KB. Figure 13(b) shows the experiment results of the correlated synthetic dataset. In this experiment, the database size has varied between 100KB to 600KB, and the number of dimensions has been fixed to eight for both missing and non-missing values dimensions. Figure 13(c) depicts the results for the NBA real dataset. The database size has varied within the range of 40 to 200KB and the number of dimensions has been fixed to 17, including missing and non-missing values dimensions. Figure 13(d) details the results of the experiment on the CoIL 2000 insurance company dataset. In this experiment we have considered 21 dimensions for both missing and non-missing values, the dataset size varying between 50 to 300KB.

The obtained experiment results show that our approach (*OIS*) steadily outperforms the most recent approaches (SCSA, SPQ, SIDS, *Incoskyline* and *Isykline*) in all cases. The results also suggest that *Iskyline* performs the worst by producing the highest number of pairwise comparisons for all cases. Moreover, the result also shows that the number of pairwise comparisons generated by SCSA and *IncoSkyline* is slightly higher than that of OIS when the dataset size is less than 400KB. Lastly, the experiment results also indicate that SPQ performs slightly better than SIDS in all cases. We can also conclude that the database cardinality has marginal influence on the performance of our proposed solution. This improvement is due to the data sorting and pruning technique as proposed in our solution. Besides, the local skylines identifier component has also successfully removed many dominated data items prior to running the skyline operation, which in turn reduces the domination tests necessary to retrieve the final skyline data items of the database.

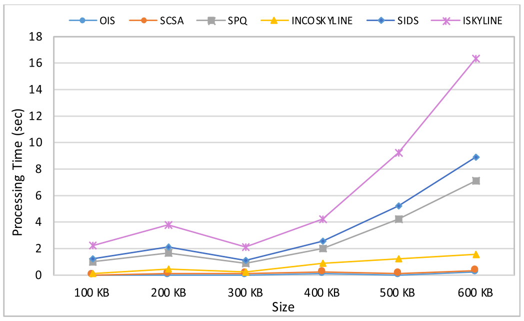
Figure 14(a), 14(b), 14(c), and 14(d) illustrate the amount of time required to process skyline queries for the independent, correlated, NBA, and CoIL 2000 insurance company datasets. It should be noted that the parameters setting of this set of experiments are similar to the set of experiments that are described above in Figure 13.

Figure 14(a) depicts the experimental results of the processing time of skyline queries in the incomplete database over the independent dataset.

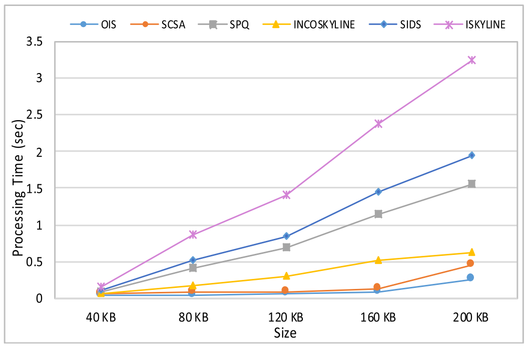
The experimental results suggest that our proposed strategy outperforms the previous strategies (SCSA, SPQ, SIDS, *Incoskyline* and *Isykline*) in all cases and requires less processing time. This significant reduction in the processing time is due to the sorting and filtering technique that eliminates many unwanted dominated data items before applying the skyline process. It is obvious that eliminating these dominated data items helps avoid many unnecessary pairwise



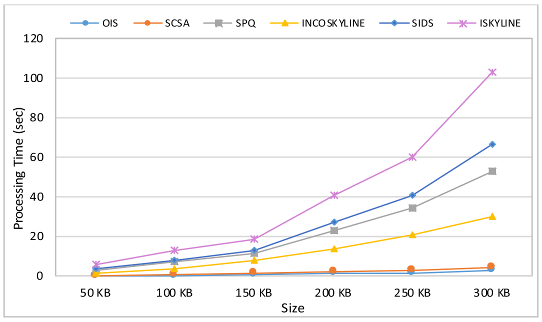
a) Independent



b) Correlated



c) NBA



d) CoIL 2000

FIGURE 14. The effect of dataset size on the processing time.

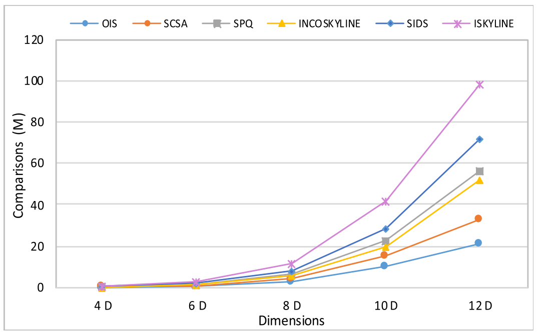
comparisons, which in turn decreases the processing time and keeps our approach scalable even if the dataset size increased. Figure 14(b) illustrates the experimental results of our proposed approach on the correlated synthetic dataset. The results show that our technique is superior to the other techniques (SCSA, SPQ, SIDS, *Incoskyline* and *Isykline*). The results also reveal that *ISkyline* is the worst performing of the five techniques, and its processing time dramatically increases with increased database size. Also, the results show that SPQ performs marginally better than SIDS in all cases. However, our approach has performed slightly better than *Incoskyline* when the dataset size is less than 400KB. This is due to the fact that *Incoskyline* also uses data filtration before applying skyline process. Nevertheless, our technique performs better when the database size is increased as it uses advanced filtration that further eliminates many dominated data items before running the skyline process.

The same behavior has also been observed in the other datasets. The performance of our approach is better than SCSA, SPQ, SIDS, *Incoskyline* and *ISkyline* when the dataset size is greater than 400KB. This is due to the fact that our approach successfully removes the dominated data items at the earliest possible stage before conducting pairwise comparisons among the data items to identify the skyline data items.

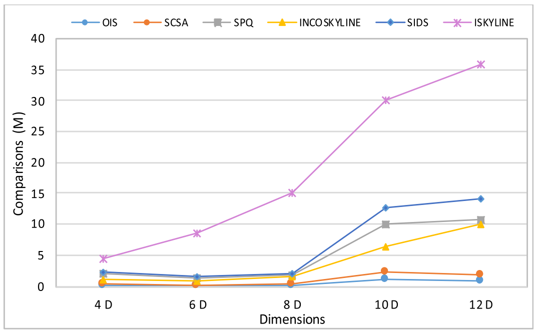
Figure 14(c) shows the results that have been obtained in the experiment on the NBA dataset. Here the processing time of skyline process is less than that of SCSA, SPQ, SIDS, *Incoskyline*, and *ISkyline* for different dataset sizes since many data items are excluded from pairwise comparison, resulting in less processing time. *ISkyline* and SIDS perform worse when the dataset size increases as it has to examine the whole data more than once meaning that the number of pairwise comparisons among the data items in the dataset increases. Figure 14(d) presents the processing time results for the skyline operation on the CoIL 2000 insurance company dataset. Based on the results we can conclude that our approach outperforms SCSA, SPQ, SIDS, *Incoskyline*, and *ISkyline* in all cases. We can also notice that the size of the dataset significantly influences the performance of the other approaches such as *ISkyline* and SIDS and leads to higher processing time when the dataset size is increased. However, our approach is only slightly influenced by the increment in the dataset size and maintains a reasonable processing time when the dataset size is increased. This is due to the same reasons already explained in the previous experiments.

B. DATA DIMENSIONALITY

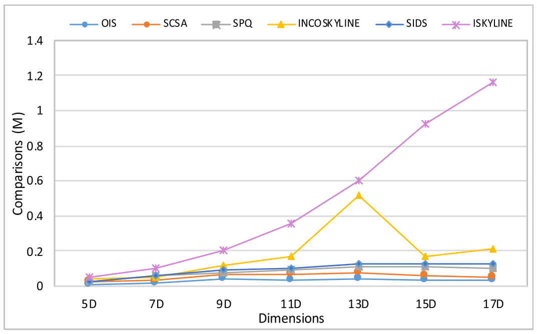
It has been argued that skyline queries are highly influenced by the number of dimensions that are involved in the skyline process. Thus, this set of experiments attempt to examine the impact of the number of dimensions on the pairwise comparisons process. Two types of datasets have been used, namely synthetic and real datasets, aiming at computing the total number of pairwise comparisons that need to be performed in order to identify the skyline data items. Moreover, the processing time of the skyline process for each approach has also been computed. In this set of experiments, the dataset size has been fixed and the number of dimensions varied. Figure 15(a), 15(b), 15(c), and 15(d) depict the experiment results for the independent, correlated, NBA, and CoIL 2000 insurance company datasets. Figure 15(a) describes the experimental results for the independent dataset by varying



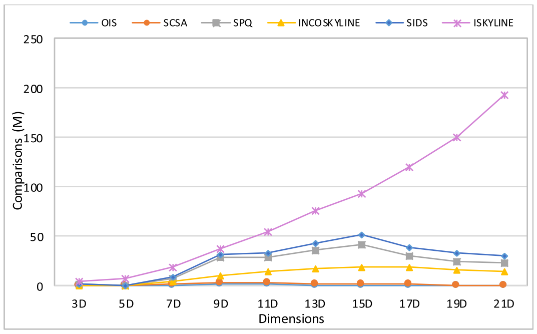
a) Independent



b) Correlated



c) NBA



d) CoIL 2000

FIGURE 15. The effect of number of dimensions on the pairwise comparisons.

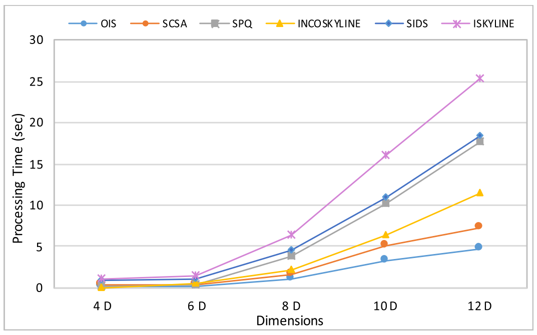
the number of dimensions between four and 12 and fixing the dataset size to 300KB. From the results it can be concluded that that our approach outperforms SCSA, SPQ, SIDS, *Incoskyline* and *Isykline* in all cases. Figure 15(b) shows the experiment results on the correlated dataset in which the dataset size has been fixed to 300KB and the number of dimensions varied between four and 12. In respect to the correlated dataset it seems that our approach is superior in comparison to SCSA, SPQ, SIDS, *Incoskyline* and *Isykline* in all cases. It is also clear that our approach performs slightly better than SCSA when the number of dimensions is less than eight. Nevertheless, the OIS approach marginally impacts when the number of dimensions increase and steadily outperforms the other approaches (SCSA, SPQ, SIDS, *Incoskyline* and *Isykline*) when the number of dimensions is more than eight. Figure 15(c) illustrates the experiment results of the NBA dataset where the database size has been fixed to 120KB and the number of dimensions varied between five and 17. The result shows that our approach steadily outperforms SCSA, SPQ, SIDS, *Incoskyline* and *Isykline* and that data dimensionality has no great effect on the performance of OIS due to the use of filtration which optimizes the process of identifying skyline data items in a database with incomplete data. Figure 15(d) depicts the results of the experiment on the real dataset, CoIL 2000, for an insurance company that shows the number of pairwise comparisons between data items to identify the skylines in an incomplete database. In this experiment we have aimed to test the approaches by varying the number of dimensions ranging from tree to 21 and fixing the dataset size to 150K. It can be observed that our approach is superior and outperforms the other approaches in all cases.

Given this set of experiments we can conclude that OIS is more scalable, and the number of dimensions has no significant impact on its performance. Furthermore, the performance of the other approaches (SCSA, SPQ, SIDS, *Incoskyline* and *Isykline*) diminishes when the number of dimensions increases. The result has also revealed that SIDS and *ISkyline* perform the worst in all cases as both are performed on the initial dataset without sorting or filtration to eliminate the dominated data items before applying the skyline process.

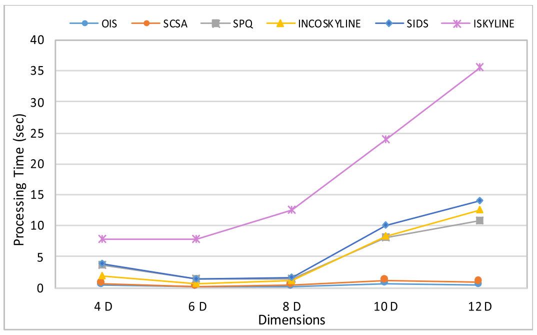
Figure 16(a), 16(b), 16(c), and 16(d) present the processing time of the skyline process produced in each approach to derive the skylines on the independent, correlated, NBA, and CoIL 2000 insurance company datasets. It should be noted here that the parameter setting of this set of experiments is same as that of the previous experiment.

The produced figures indicate that the *OIS* technique requires less time to carry out the skyline process on a database with incomplete data as most of the data items are eliminated from the skyline process at the early stage through data filtration. Also noticeable is that ISkyline performs the worst when the number of dimensions increases. In the NBA dataset (Figure 16(c)) our approach steadily outperforms SCSA, SPQ, SIDS, *Incoskyline* and *Isykline* when the number of dimensions increases. Nevertheless, OIS does not significantly improve when the number of dimensions is less than 11, and the improvement is slightly less compared to the correlated dataset.

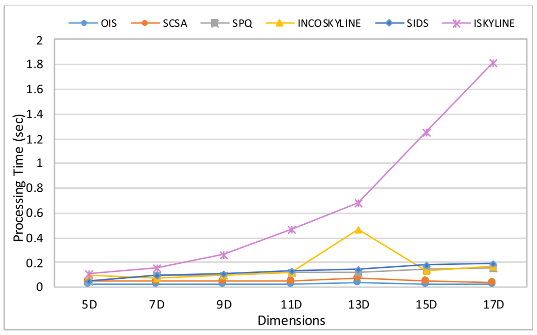
Moreover, in the CoIL 2000 insurance company dataset (Figure 16(d)), the OIS performance improves when the number of dimensions is greater than nine. Although our approach is more scalable and performs better when the number of dimensions is greater than 11, it performs the best when the range of values is high. This stands in stark contrast to SIDS which performs the worst in all cases.



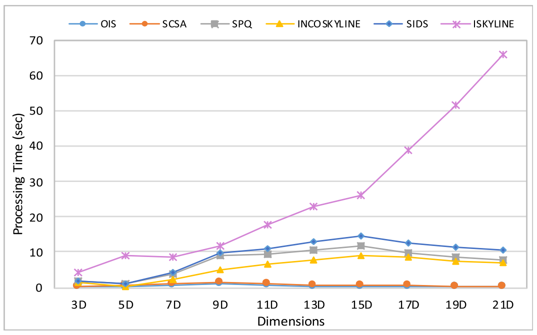
a) Independent



b) Correlated



c) NBA



d) CoIL 2000

FIGURE 16. **The effect of number of dimensions on the processing time.**

Based on the experiment results we have reported, we can conclude that our proposed technique steadily outperforms the closest recent techniques (SCSA, SPQ, SIDS, *Incoskyline* and *Isykline*) proposed for processing skyline queries in a database with incomplete data. In addition, the results of the experiments confirmed the effectiveness and the efficiency of our proposed framework in processing skyline queries in a database with incomplete data. The main cause of the superiority of our approach is the sorting and filtering process applied to the data items based on the non-missing values. This process aids in eliminating many unwanted data items before applying the skyline process in contrast to the other approaches which run the skyline process without filtering the initial data. The prior filtering of the initial data results in the significantly reduced number of pairwise comparisons of the data items. Moreover, applying the concept of parallel execution of data filtration accelerates the entire computing process and greatly reduces the overall processing time of the skyline operation.

VI. CONCLUSION

This paper discusses the problem of processing skyline queries in an incomplete database and proposes the Optimized Incomplete Skylines (*OIS*) framework to return the skyline data items from a database with incomplete data. The individual steps of each component in the proposed framework have been explained. *OIS* incorporates a particular optimization technique that helps eliminate the dominated data items before applying the skyline technique. This optimization approach results into the reduced number of pairwise comparisons between data items when identifying the skyline data items. Two sets of experiments have been carried out to compare the performance of *OIS* with the other recent related algorithms designed for processing skyline queries on incomplete data, namely SCSA, SPQ, SIDS, *Incoskyline* and *Isykline*. The results have confirmed that our approach significantly outperforms the previous approaches in terms of number pairwise comparisons and processing time.

REFERENCES

[1] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "On high dimensional Skylines," in *Advances in Database Technology*. Berlin, Germany: Springer, Mar. 2006, pp. 478–495.

[2] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant Skylines in high dimensional space," presented at the ACM SIGMOD Int. Conf. Manage. Data, Chicago, IL, USA, 2006.

[3] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous top-k dominating queries in subspaces," in *Proc. Panhellenic Conf. Informat.*, Samos, Greece, Aug. 2008, pp. 31–35.

[4] K. El Maarry, C. Lofi, and W.-T. Balke, "Crowdsourcing for Query processing on Web data: A case study on the Skyline operator," *J. Comput. Inf. Technol.*, vol. 23, pp. 43–60, Mar. 2015.

[5] C. Lofi, K. E. Maarry, and W.-T. Balke, "Skyline queries in crowd-enabled databases," presented at the 16th Int. Conf. Extending Database Technol., Genoa, Italy, 2013.

[6] J. Lee, D. Lee, and S.-W. Kim, "CrowdSky: Skyline computation with crowdsourcing," in *Proc. EDBT*, 2016, pp. 125–136.

[7] M. B. Swidan, A. A. Alwan, S. Turaev, and Y. Gulzar, "A model for processing Skyline queries in crowd-sourced databases," *Indonesian J. Electr. Eng. Comput. Sci.*, vol. 10, pp. 798–806, May 2018.

[8] Y. Gulzar, A. A. Alwan, N. Salleh, and I. F. A. Shaikhli, "Skyline Query processing for incomplete data in cloud environment," in *Proc. 6th Int. Conf. Comput. Informat.*, Kuala Lumpur, Malaysia, Apr. 2017, pp. 567–576.

[9] Y. Gulzar, A. A. A. Aljuboori, N. Salleh, and I. F. Al Shaikhli, "Identifying Skylines in cloud databases with incomplete data," *J. ICT*, vol. 18, pp. 19–34, Jan. 2019.

[10] M. L. Yiu and N. Mamoulis, "Efficient processing of top-k dominating queries on multi-dimensional data," presented at the 33rd Int. Conf. Very Large Data Bases, Vienna, Austria, 2007.

[11] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline Query processing for incomplete data," in *Proc. IEEE 24th Int. Conf. Data Eng.*, Cancun, Mexico, Apr. 2008, pp. 556–565.

[12] R. Bharuka and P. S. Kumar, "Finding Skylines for incomplete data," presented at the 24th Australas. Database Conf., vol. 137, Adelaide, Australia, 2013.

[13] A. A. Alwan, H. Ibrahim, and N. I. Udzir, "A framework for identifying Skylines over incomplete data," in *Proc. 3rd Int. Conf. Adv. Comput. Sci. Appl. Technol. (ACSAT)*, Dec. 2014, pp. 79–84.

[14] A. A. Alwan, H. Ibrahim, N. I. Udzir, and F. Sidi, "An efficient approach for processing Skyline queries in incomplete multidimensional database," *Arabian J. Sci. Eng.*, vol. 41, pp. 2927–2943, Aug. 2016.

[15] Y. Gulzar, A. A. Alwan, N. Salleh, I. F. A. Shaikhli, and S. I. M. Alvi, "A framework for evaluating Skyline queries over incomplete data," *Procedia Comput. Sci.*, vol. 94, pp. 191–198, Jan. 2016.

[16] J. Lee, H. Im, and G.-W. You, "Optimizing Skyline queries over incomplete data," *Inf. Sci.*, vol. 361, pp. 14–28, Sep. 2016.

[17] A. A. Alwan, H. Ibrahim, N. I. Udzir, and F. Sidi, "Processing Skyline queries in incomplete distributed databases," *J. Intell. Inf. Syst.*, vol. 48, pp. 399–420, Apr. 2017.

[18] G. Babanejad, H. Ibrahim, N. Z. Udzir, F. Sidi, and A. A. Alwan, "Deriving Skyline points over dynamic and incomplete databases," in *Proc. 6th Int. Conf. Comput. Informat.*, Kuala Lumpur, Malaysia, Apr. 2017, pp. 77–83.

[19] Y. Gulzar, A. A. Alwan, N. Salleh, and I. F. A. Shaikhli, "Processing skyline queries in incomplete database: Issues, challenges and future trends," *J. Comput. Sci.*, vol. 13, no. 11 pp. 647–658, 2017.

[20] Y. Gulzar, A. A. Alwan, N. Salleh, and I. F. Al Shaikhli, "A model for skyline Query processing in a partially complete database," *Adv. Sci. Lett.*, vol. 24, no. 2 pp. 1339–1343, 2018.

[21] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, Mar. 2005.

[22] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," presented at the 31st Int. Conf. Very Large Data Bases, Trondheim, Norway, 2005.

[23] Z. Huang, J. Zhang, and C. Tian, "Efficient processing of the skyline-CL Query," *Arabian J. Sci. Eng.*, vol. 41, pp. 2801–2811, Aug. 2016.

[24] S. Borzsony, D. Kossmann, and K. Stocker, "The Skyline operator," in *Proc. 17th Int. Conf. Data Eng.*, Cancun, Mexico, Apr. 2001, pp. 421–430.

[25] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proc. 27th Int. Conf. Very Large Data Bases (VLDB)*, Roma, Italy, 2001, pp. 301–310.

[26] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An Online algorithm for skyline queries," presented at the 28th Int. Conf. Very Large Data Bases, Hong Kong, 2002.

[27] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. 19th Int. Conf. Data Eng. (ICDE)*, Bangalore, India, 2003, pp. 717–719.

[28] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," presented at the ACM SIGMOD Int. Conf. Manage. Data, San Diego, CA, USA, 2003.

[29] I. Bartolini, P. Ciaccia, and M. Patella, "SaLSa: Computing the Skyline without scanning the whole sky," presented at the 15th ACM Int. Conf. Inf. Knowl. Manage., Arlington, VA, USA, 2006.

[30] K. C. Lee, W.-C. Lee, B. Zheng, H. Li, and Y. Tian, "Z-SKY: An efficient Skyline Query processing framework based on Z-order," *VLDB J.*, vol. 19, pp. 333–362, Jun. 2010.

[31] S. Zhang, N. Mamoulis, and D. W. Cheung, "Scalable Skyline computation using object-based space partitioning," presented at the ACM SIGMOD Int. Conf. Manage. Data, Providence, RI, USA, 2009.

[32] J. Lee and S.-W. Hwang, "Scalable skyline computation using a balanced pivot selection technique," *Inf. Syst.*, vol. 39, pp. 1–21, Apr. 2014.

[33] M. S. Arefin and Y. Morimoto, "Skyline sets queries for incomplete data," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, pp. 67–80, Oct. 2012.
[34] X. Miao, Y. Gao, L. Chen, G. Chen, Q. Li, and T. Jiang, "On efficient *K*-skyband Query processing over incomplete data," in *Proc. 18th Int. Conf. Database Syst. Adv. Appl.*, Wuhan, Chian, 2013, pp. 424–439.
[35] R. Bharuka and P. S. Kumar, "Finding superior Skyline points from incomplete data," presented at the 19th Int. Conf. Manage. Data, Ahmedabad, India, 2013.
[36] K. Zhang, H. Gao, H. Wang, and J. Li, "ISSA: Efficient Skyline computation for incomplete data," in *Database Systems for Advanced Applications*, Dallas, TX, USA, Apr. 2016, H. Gao, J. Kim, and Y. Sakurai, Eds. Cham, Switzerland: Springer, 2016, pp. 321–328.
[37] X. Miao, Y. Gao, B. Zheng, G. Chen, and H. Cui, "Top-*k* dominating queries on incomplete data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 252–266, Jan. 2016.
[38] X. Miao, Y. Gao, G. Chen, and T. Zhang, "*K*-dominant Skyline queries on incomplete data," *Inf. Sci.*, vols. 367–368, pp. 990–1011, Nov. 2016.
[39] K. Zhang, H. Gao, X. Han, Z. Cai, and J. Li, "Probabilistic skyline on incomplete data," presented at the ACM Conf. Inf. Knowl. Manage., Singapore, 2017.
[40] Y. Wang, Z. Shi, J. Wang, L. Sun, and B. Song, "Skyline preference Query based on massive and incomplete dataset," *IEEE Access*, vol. 5, pp. 3183–3192, 2017.
[41] W.-T. Balke, U. Güntzer, and J. X. Zheng, "Efficient distributed skylining for Web information systems," in *Advances in Database Technology—EDBT* (Lecture Notes in Computer Science), vol. 2992, E. Bertino *et al.*, Eds. Berlin, Germany: Springer, 2004.
[42] Y. Gao, X. Miao, H. Cui, G. Chen, and Q. Li, "Processing *K*-skyband, constrained skyline, and group-by skyline queries on incomplete data," *Expert Syst. Appl.*, vol. 41, no. 10, pp. 4959–4974, 2014.
[43] Y. Gulzar, A. A. Alwan, H. Ibrahim, and Q. Xin, "D-SKY: A framework for processing Skyline queries in a dynamic and incomplete database," presented at the 20th Int. Conf. Inf. Integr. Web-Based Appl. Services, Yogyakarta, Indonesia, 2018.
[44] Y. Gulzar, A. A. Alwan, R. M. Abdullah, Q. Xin, and M. B. Swidan, "SCSA: Evaluating Skyline queries in incomplete data," *Appl. Intell.*, vol. 49, pp. 1636–1657, May 2019.

YONIS GULZAR received the master's degree in computer science from Bangalore University, India, in 2013, and the Ph.D. degree in computer science from International Islamic University Malaysia, in 2018. He is currently an Assistant Professor with King Faisal University (KFU), Saudi Arabia. Before joining KFU, he was a part-time Lecturer, a Teaching Assistant, and a Research Assistant with the Department of Computer Science, International Islamic University, Malaysia. His research interests include preference queries, skyline queries, probabilistic and uncertain databases, query processing and optimization and management of incomplete data, data integration, location-based social networks (LBSN), recommendation systems, and data management in cloud computing.

ALI A. ALWAN received the master's and Ph.D. degrees in computer science from Universiti Putra Malaysia (UPM), Malaysia, in 2009 and 2013, respectively. He is currently an Assistant Professor with the Kulliyyah (Faculty) of Information and Communication Technology, International Islamic University Malaysia (IIUM), Malaysia. His research interests include preference queries, skyline queries, probabilistic and uncertain databases, query processing and optimization and management of incomplete data, data integration, location-based social networks (LBSN), recommendation systems, and data management in cloud computing.

SHERZOD TURAEV received the Ph.D. degree in computer science from University Rovira i Virgili, Spain, in 2010, and the Ph.D. degree in mathematics from the National University of Uzbekistan, in 2001. He was a Postgraduate Researcher with University Putra Malaysia, from 2009 to 2012, an Assistant Professor with the Faculty of Information and Communication Technology, International Islamic University Malaysia, from 2012 to 2018, and an Associate Professor with the Faculty of Natural Sciences and Engineering, International University of Sarajevo, Bosnia and Herzegovina, from 2018 to 2019. He is currently an Associate Professor with the College of Information Technology, UAE University. His research interests include graph theory, formal languages and automata, bio-computing, and information security. He is acting as an Editorial Board Member, a Programming Committee Member, and a Reviewer in many international journals, such as *Math Reviews*, *Theoretical Computer Science*, *Sains Malaysian*, *Applied Mathematics and Computational Intelligence*, and many international conferences.