Applied Intelligence (2019) 49:1636–1657
https://doi.org/10.1007/s10489-018-1356-2

# SCSA: Evaluating skyline queries in incomplete data

Yonis Gulzar[1] · Ali A. Alwan[1] · Radhwan Mohamed Abdullah[2,3] · Qin Xin[4] · Marwa B. Swidan[1]

Published online: 7 December 2018


**Abstract**
Skyline queries have been extensively incorporated in various contemporary database applications. The list includes but is not limited to multi-criteria decision-making systems, decision support systems, and recommendation systems. Due to its great benefits and wide application range, many skyline algorithms have already been proposed in numerous data settings. Nonetheless, most researchers presume the completion of data meaning that all data item values are available. Since this assumption cannot be sustained in a large number of real-world database applications, the existing algorithms are rather inadequate to be directly applied on a database with incomplete data. In such cases, processing skyline queries on incomplete data incur exhaustive pairwise comparisons between data items, which may lead to loss of the transitivity property of the skyline technique. Losing the transitivity property may in turn give rise to the problem of *cyclic dominance*. In order to address these issues, we propose a new skyline algorithm called Sorting-based Cluster Skyline Algorithm (SCSA) that combines the sorting and partitioning techniques and simplifies the skyline computation on an incomplete dataset. These two techniques help boost the skyline process and avoid many unnecessary pairwise comparisons between data items to prune the dominated data items. The comprehensive experiments carried out on both synthetic and real-life datasets demonstrate the effectiveness and versatility of our approach as compared to the currently used approaches.

**Keywords** Skyline · Skyline queries · Incomplete data · Missing data · Preference queries · Query processing

## 1 Introduction

For the past decade, the interest in designing and developing more flexible query operators in database management systems has increased dramatically. These query operators have changed the way of retrieving data from the database. Preference queries return data items of the database if (and only if) they meet the user's given preferences. Skyline queries constitute one of the most practical and predominant types of preference queries. They were first introduced in database systems by [1]. Skyline queries return only those non-dominant data items from the database that best meet the user's given preferences in the submitted query [1–10]. For instance, someone is looking for a house in a specific area, and each house available on the market possesses different features (number of rooms, number of bathrooms, type of bedrooms, location, and distance from the workplace). Browsing a house rental website can be quite time-consuming and tedious as the interested user has to choose from among the many houses available and advertised on the website. In this scenario, submitting a skyline query will result in eliminating all those houses that do not fulfil the user's stated preferences and narrowing the results that best suit his or her *interest*.

In order to gain a better understanding of processing skylines queries in a given database, the following example of a hotel finder database will clarify how the skyline technique works. It is assumed that a researcher is looking for a hotel in proximity to his conference venue. He is looking for a hotel located closest to the conference venue and has been given the best ratings. Figure 1a represents the data in a relation form. This relation contains the details of ten hotels. The first attribute represents the hotel *ID*, the second attribute indicates the

✉ Ali A. Alwan
aliamer@iium.edu.my

[1] Department of Computer Science, Kulliyyah of Information and Communication Technology, International Islamic University Malaysia, 53100 Kuala Lumpur, Malaysia

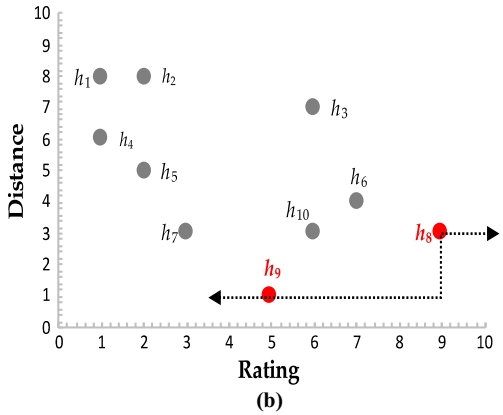
[2] Division of Basic Sciences, College of Agriculture and Forestry, University of Mosul, Mosul, Iraq

[3] Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Serdang, Malaysia

[4] Faculty of Science and Technology, University of Faroe Islands, Faroe Islands, Denmark

| Restaurant ID | Rating | Distance |
|---|---|---|
| $h_1$ | 1 | 8 |
| $h_2$ | 2 | 8 |
| $h_3$ | 6 | 7 |
| $h_4$ | 1 | 6 |
| $h_5$ | 2 | 5 |
| $h_6$ | 7 | 4 |
| $h_7$ | 3 | 3 |
| **$h_8$** | **9** | **3** |
| **$h_9$** | **5** | **1** |
| $h_{10}$ | 6 | 3 |

(a)




(b)

**Fig. 1** Skylines of hotel recommendation database

rating of the hotels, and lastly, the third attribute denotes the distance to the conference venue. Figure 1b illustrates the representation of the hotel-finder database example in 2-*D* space. Based on the definition of skyline queries and considering the existing constraints, it can be learned that hotels $h_1$, $h_2$, $h_4$, $h_5$, and $h_7$ are fully dominated by $h_9$ and $h_8$. This is due to the fact that $h_9$ and $h_8$ have better ratings than the other hotels ($h_1$, $h_2$, $h_4$, $h_5$, and $h_7$) and are similar in terms of distance. Hotels $h_3$, $h_6$, and $h_{10}$ are partially dominated by $h_9$ based on distance. However, these hotels are dominated by $h_8$ in both dimensions (better rating and shortest distance). The dominated hotels ($h_1$, $h_2$, $h_3$, $h_4$, $h_5$, $h_6$, $h_7$, $h_{10}$) are not considered as skylines. It can also be observed that hotel $h_9$ partially dominates $h_8$ in dimension three (shortest distance), and similarly, hotel $h_8$ partially dominates hotel $h_9$ in the second dimension (better rating). According to the skyline definition, these two hotels ($h_8$, and $h_9$) are the skylines of this hotel-finder database.

Due to their immense benefits, skyline queries are being widely utilized in many domains such as multi-criteria decision making, decision support [11–15], hotel recommender [16], restaurant finder [2, 16], temporal databases [17], crowdsourcing databases [18–20], and cloud databases [21]. Since the first introduction of a skyline operator into a database system by Borzsony et al. [1], researchers in the database community have been striving to improve the performance of skyline process by reducing the searching space and minimizing the number of pairwise comparisons, which in turn lowers the processing time of any skyline computation. Many skyline approaches have been proposed in the literature [11–13, 15, 22–24] concentrating on skyline issues over complete databases where values of all the dimensions are present in the database. The completeness of the data renders it easier to identify the skylines among all data items present in the database which fit the user's preference. However, in the real-world databases, which are mostly large and multidimensional databases, it is rather unlikely that the database is complete. In other words, values of some data items are not present (missing) in one or more dimensions. Thus, the assumption of data completeness in any given skyline process cannot be sustained.

The incompleteness of data in databases has given rise to many challenges in evaluating skyline queries and made it quite difficult to identify the skylines of the database. Due to the incompleteness of the data, the approaches proposed for complete databases are not recommended to be directly applied on incomplete databases. Incomplete data results in unnecessary pairwise comparisons that eventually lead to the cost of processing skyline queries being prohibitive. Most importantly, an incomplete database also means losing the transitivity property of the skyline, which in turn may lead to the problem of *cyclic dominance* [2]. Furthermore, in large incomplete databases, the size of skylines may increase due to the incompleteness of data in many dimensions and render many data items as incomparable. Besides, a high number of skylines does not necessarily provide any insight to the user and may not assist him or her in making the right choice [2].

In order to further clarify the issue of incomplete data and its impact on processing skyline queries, the following example is given. A house-rental website possesses a database to allow users to rent houses, and a customer wants to rent a house whose features should be as follows: 1) It should have the maximum number of bedrooms; 2) It should be closest to his workplace; and 3) the rent should be the lowest among the houses available in the database. Let it be assumed that the database contains three houses and that some dimensions are missing: $h_1$ (2, *, 600), $h_2$ (*, 5, 700) and $h_3$ (3, 6, *). The first dimension contains the number of rooms, the second dimension the distance (in *km*) from the workplace, and the third dimension the price of the rent. The symbol * denotes that the value of a particular dimension is not present (missing). In order to evaluate the skylines for this kind of incomplete dataset, we need to compare all houses with one other and identify the best house according to the user's preferences. When comparing $h_1$ with $h_2$ from the given dataset, we find that the $h_1$ dominates $h_2$ in the third dimension. When comparing $h_2$ with $h_3$, we find that $h_2$ is better than $h_3$ in the second dimension. In respect to the transitivity property where $h_1$ dominates $h_2$ and $h_2$ dominates $h_3$, so $h_1$ should dominate $h_3$. However, this is not the case in the given dataset. Instead, we find that $h_3$ dominates $h_1$. Hence, we lose the transitivity property and also encounter the issue of *cyclic*

*dominance*. In consequence, the process ends with no result as all houses are dominated by one other.

- **Transitivity:** Given $a_i$, $a_j$, $a_k \in R$, if $a_i \succ a_j$, and $a_j \succ a_k$, according to transitivity property $a_i \succ a_k$ holds for complete data. However, it may not hold for incomplete data.
- **Cyclic dominance:** Given $a_i$, $a_j$, $a_k \in R$, $a_i \succ a_j$, $a_j \succ a_k$, *and* , $a_k \succ a_i$ may hold over incomplete data.

The research work presented in this paper focuses on sorting the data items based on the available data in each dimension in descending order. This is followed by accumulating the *domination power* of each data item denoting the number of dominated data items by scanning the sorted lists. Subsequently, filtration is applied to begin with the initial pruning for the dominated data items before applying the skyline technique. Then, the remaining data items are partitioned in disjoint sets called clusters based on their *domination power*. These clusters are further divided into smaller groups, where each group contains data items with the same *bitmap representation*. Dividing the clusters into smaller groups simplifies the skyline process and helps sustain the transitivity property of the skyline technique. In addition, it also helps reduce the searching space and minimizes the domination tests. In order to accelerate the process, it is run parallel on each group to eliminate the unwanted data items effectively. Lastly, the approach ends by comparing the local cluster skylines and returns only non-dominated data items as the skylines.

The following points summarize the contributions of this paper:

- We re-introduce the problem of identifying skylines in a database with incomplete data and justify the need to form a new and more efficient solution.
- We conduct a comprehensive review examining the most notable work done in the area of skyline queries in database systems. This covers the previous approaches designed for complete and incomplete databases. The focus is given on examining and summarizing the strengths and the weaknesses of each approach.
- We propose a new skyline algorithm processing skyline queries on incomplete data called *Sorting-based Cluster Skylines Algorithm* (***SCSA***) that efficiently answers skyline queries in incomplete databases by exploiting the sorting and clustering in simplifying the skyline computation.
- We develop an innovative method that helps sort initial data items into distinct clusters based on the *domination power* of the data items.
- We incorporate the two optimization techniques of filtration and optimization that help reduce the number of pairwise comparisons between data items. Filtration prunes the initial incomplete database and eliminates the dominated data items before applying the skyline technique.
- We evaluate the efficiency and the effectiveness of the proposed solution through experiments using both real and synthetic datasets. These experiments demonstrate the effectiveness of the proposed solution.

The remainder of the paper is organized as follows: In Section 2, the previous works related to this research are reported and discussed. The basic definitions and notations used in the rest of the paper are set out in Section 3. The proposed approach is explained and illustrated in Section 4, and the experimental results are illustrated and explained in Section 5. The conclusion is described in Section 6.

## 2 Related work

Skyline queries were initially examined for maximal vector [25–27] or Pareto set computation [28, 29]. Borzsonyi et al. [1] was the first who presented the skyline operator in database management systems. Following this example, a number of works were proposed to improve the efficiency and performance of the skyline process in database management systems. Therefore, the proposed algorithms in the literature focus on reducing the searching space, minimizing the pairwise comparisons between the data items to declare the skylines, and reducing the execution time. In this section, we review and investigate the existing algorithms designed for complete databases as described in Subsection 2.1. Subsection 2.2 concentrates on examining the existing skyline solutions formed for incomplete databases.

### 2.1 Skyline queries for complete databases

Borzsonyi et al. [1] proposed the two algorithms BNL and D&C. Block Nested-Loop (BNL) reads data items one by one and compares them to the other data items in the window. The data items that are dominated will be removed from the window leaving the rest of data items in the window for next iterations. Divide and Conquer (D&C) divides the initial dataset into two main sections, computes the local skylines of both sections separately, and then identifies the final skylines by computing local skylines of both sections together. A numerous number of algorithms have been offered after BNL and D&C and applied two methods (*sorting* and *partitioning*) on initial data to evaluate skylines.

**Sorting approaches** The main goal of sorting is to reorganize the data items in such a way that the dominated data items will be pruned as soon as possible, which also reduces domination tests. As in BNL, the data items are accessed in the initial (input) state. In order to enhance the efficiency of BNL, other

algorithms like SFS [23], LESS [30] and SaLSa [31] have been proposed. These algorithms use the sorting concept and rearrange the data items to eliminate those data items at the early stages of the skyline process that possess the least potential. SFS sorts entire database in non-ascending order to eliminate non-skyline data items early, whereas LESS combines the advantages of SFS and BNL and reorders the data items. SalSa uses another function and keeps dominant data items on top that helps to prune more data items and reduces domination tests.

**Partitioning approaches** The main idea behind partitioning datasets is to increase the efficiency of the algorithm. It also reduces the domination tests between the data items. As stated earlier in the literature, D&C divides the dataset into two main subsets and then retrieves the skylines by recursively partitioning the subsets. Several other algorithms such as INDEX [11], NN [22], BBS [24], OSPS [32], ZSearch [33], BSkyTree [34] have been proposed based on the D&C concept by dividing the dataset into small partitions before retrieving the local skylines from all partitions, combining all local skylines and producing the final skyline.

## 2.2 Skyline queries for incomplete databases

While Borzsonyi et al. [1] first introduced skyline queries in complete data, Khalefa et al. [2] first proposed skyline queries for incomplete data. They proposed the *ISkyline* algorithm to identify skylines in incomplete data. In the *ISkyline* algorithm, he optimized his own proposed algorithm called the Bucket algorithm [2] using the two optimizing techniques *virtual points* and *shadow skylines*. Instead of comparing all data items of bucket $B_i$ with $B_{i++}$, virtual points $P_k$ are created for each bucket ($B_i$-$B_n$), and the virtual point of $B_i$ is used to compare the data items present in another group $B_{i++}$. The objective here is to reduce the maximum number of unwanted domination tests and prune unwanted data items. In order to identify the correct skylines, a set of data items represented as shadow skylines are used to dominate some data items across the buckets. This set of data items, however, is dominated by virtual points. Regardless of the enhanced Bucket algorithm, ISkyline's performance is affected by the order of initial data. Using the virtual point technique may produce incorrect results. It may be that the data item in bucket $B_{i++}$ should be in the final skyline, yet due to using the virtual point of bucket $B_i$ the data item is dominated or vice versa. Moreover, *ISkyline* involves *n* number of virtual points which are derived from the local skylines of the buckets to be placed on top of each bucket to prune the dominated local skylines. However, the number of virtual points increases when the number of local skylines increases, which in turn increases the number of pairwise comparisons. The performance of *ISkyline* is highly influenced by the number of virtual points.

RSSSQ [35] uses the same concept of Khalefa et al. [2] by replacing missing values with certain numbers that are larger than the domain value in order to avoid losing the transitivity property of the skyline and the issue of cyclic dominance. Miao et al. [36] proposed three algorithms known as baseline, virtual point (VP) and k-iSkyband (kISB). Their baseline algorithm requires a high number of pairwise comparisons to retrieve the skylines. In addition, the correction of data items in the buckets is ignored. 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.

Based on [2], Bharuka and Kumar [37] proposed the *SIDS* algorithm (*Sorting-based Incomplete Data Skyline*) to improve the ISkyline algorithm. SIDS uses the pre-sorting approach on input data to rearrange the position of data items. This rearrangement helps place those data items on the top of the list that are most likely to dominate other data items. SIDS uses the sorting technique of [13, 38] where values of each dimension are sorted in a descending order without counting the missing dimensions. Domination tests are done by selecting a data item *P* from each dimension according to the round-robin method and comparing it with the data items *C* present in the same dimension. The variable *ProcessedCount (pc)* keeps the record of domination tests of the data item *P*. Dominated data items are subsequently removed from the candidate set. If the value *pc* of *P* is equal to the number of non-missing dimensions of *P*, then *P* will be moved to *ResultSet*. When all candidate skylines (data items) are processed at least once, the domination process stops and the remaining data items in the candidate set are also moved to *ResultSet*. The *ResultSet* consists of the final skylines. However, in this SIDS approach, the lists have to be accessed in a sequential order, and the system has to receive the results of all lists before moving to the next phase. Thus, the increasing the number of lists may degrade the performance of the skyline process and delay generating the skylines for the end user. Moreover, SIDS is not fully efficient in handling all types of incomplete datasets as it is based on datasets used in [13, 38]. SIDS lacks any optimization that could simplify the process of identifying the skylines. It identify skylines by accessing each data item in a sequential order. This sequential access renders the process of pairwise comparisons very tedious and exhausting as many unnecessary pairwise comparisons are needed to eliminate the dominated data items.

[39] has discussed the issue of processing skyline queries in incomplete datasets by proposing an approach named *Incomplete Data Frequent Skyline* or IDFS. IDFS adopts the *top-k* frequent skyline technique suggested in [12] in order to control the size of the skyline results. It utilizes the concept of top-k to derive superior skylines based on the fractional skyline frequency of each data item, which enables it to identify the superior skylines for a database with missing values.

Experimental results have reportedly shown the efficiency of IDFS using real and synthetic datasets. However, its performance becomes highly degraded if the examined space for determining the frequent skylines data items is very large.

A number of alternative approaches have been proposed for processing skyline queries in incomplete data such as the COBO framework [40] and SOBA [41]. Based on the COBO framework, Zhang et al. [40] proposed an algorithm (ISSA) that identifies skylines in two phases. In the first phase the bucket technique from [2] is adopted to eliminate most of the dominated data items at an early stage. In the second phase a function is utilized that aggregates the values of non-missing dimensions of data items and sorts those data items in descending order to prune dominated data items. SOBA, on the other hand, uses the same approach of partitioning the data items into subsets based on the same namespace. For optimization of data items, SOBA uses a technique whereby data items are sorted in ascending order for each subset to identify local skylines. Unlike ISSA, SOBA compares data items with one another across buckets without considering to prune the dataset by eliminating the dominated data items before applying the skyline technique. This, however, necessitates many unwanted pairwise comparisons among data items during the skyline process.

To the best of our knowledge, the latest works on the issue of processing skyline queries on incomplete databases are authored by Alwan et al. [42] and Wang et al. [43]. Alwan et al. [42] has developed the *Incoskyline* algorithm for processing skyline queries in multidimensional and incomplete databases. *Incoskyline* has improved *ISkyline* by using a different way of reducing pairwise comparisons and pruning non-skyline data items as early as possible. In the *Incoskyline* algorithm, the initial dataset is divided into a different set of clusters $C_n$, each cluster containing data items of the same namespace (same bitmap representation). Each cluster is divided into two groups $C_i.G_1$, $C_i.G_2$. The first groups contain all the data items that possess the highest values in any dimension while the other group contains the data items with the second highest values. Subsequently, the local skylines are found by comparing the data items with one another in each group. After combining the local skylines of each group in one list, domination tests are carried out to determine the local skylines of each cluster ($C_i$-$C_n$). A virtual data item called *k-dom* is created by selecting the highest values of all dimensions of any data item in cluster $C_i$ and $C_{i+1}$. *k-dom* is used to compare data items of $C_{i+1}$ instead of comparing all data items of $C_i$ with $C_{i+1}$. Thus, for cluster $C_1$ *k-dom* is created from $C_2$ and $C_3$, for cluster $C_2$ *k-dom* is created from cluster $C_3$ and $C_4$. Likewise, for Cluster $C_n$ *k-dom* is created from cluster $C_{n-1}$ and $C_1$. Although *Incoskyline* has improved *Iskyline* in terms of execution time and pairwise comparisons, yet it lacks result accuracy due to its use of a virtual data item (*k-dom*), which is similar to *Iskyline* as explained earlier.

Wang et al. [43] has recently introduced the SPQ approach (*Skyline Preference Query*). SPQ adopts the SIDS [37] approach. However, it divides the initial data into two subsets based on preference. The first subset contains all those dimensions given high priority by the user. The SIDS approach is implemented on the first subset to retrieve the local skylines. However, for another subset the D&C [2] technique is implemented to divide the data items based on their bitmap representation and identify the local skylines. In order to retrieve the final skylines, the local skylines of the first subset are compared to the local skylines of the second subset. Although SPQ has improved SIDS, both approaches include the crestion of multiple arrays, each array being processed sequentially, which slows down the processing time. Furthermore, many unwanted pairwise comparisons are performed in order to identify the local skylines of each subset since filtration is not done to prune the unwanted data items as early as possible.

In clear contrast, our proposed algorithm (SCSA) uses a new hybrid approach that combines the power of sorting and partitioning to prune the unwanted data items before further processing. Unlike *SPQ* and *SIDS*, the proposed approach uses several optimization techniques that allow the pruning of the dominated data items before applying the skyline operation. This is achieved during the sorting and filtering and selecting superior local skylines phases. We attempt to identify those data items that are most likely to be contained in the final skyline result by identifying the *domination power* of each data item. Also, partitioning the data items based on their *domination power* value enhances the performance of our solution.

## 3 Definitions and notations

In this section, a number of definitions and notations are provided related to skylines queries in incomplete databases. These definitions and notations help clarify our proposed approach.

Table 1 summarizes the symbols used throughout the paper. These terms are further explained below. Our approach has been developed in the context of incomplete relational databases, $D$. A relation of the database $D$ is denoted by $R(d_1, d_2..., d_m)$ where $R$ is the name of the relation with $m$-arity and $d = (d_1, d_2, ..., d_m)$ is the set of dimensions.

- **Definition 1 Skyline:** The skyline technique retrieves the skyline $S$, in a way such that any skyline in $S$ is not dominated by any other data items in the database.
- **Definition 2 Dominance:** Given two data items $p_i$ and $p_j \in D$ database with d dimensions, $p_i$ dominates $p_j$ (the greater is better) (denoted by $p_i \succ p_j$) if (and only if) the

**Table 1** Symbols and description

| *D* | Dataset |
|---|---|
| *dt, a* | Data item (tuple) |
| *n* | Number of data items |
| *d* | Dimension |
| *dp* | Domination power |
| *m* | Total no. of dimensions |
| *u* | Non-missing dimensions |
| *C* | Cluster(s) created based on domination power |
| *G* | Group(s) created based on bitmap representation of *dt* |
| *CLS* | Cluster Local Skyline(s) |
| *SLS* | Superior Local Skyline(s) |
| *R* | Relation |
| *th* | Threshold to eliminate some data items based on *dp* |
| *dp-list* | Domination power list |

following condition holds: $\forall\ d_k \in d,\ p_i.d_k \geq p_j.d_k \wedge \exists d_l,\ \in d,\ p_i.d_l > p_j.d_l$.

- **Definition 3 Skyline Queries:** Select a data item $p_i$ from the set of $D$ database if (and only if) $p_i$ is as good as $p_j$ (where $i \neq j$) in all dimensions (attributes) and strictly better than $p_j$ in at least one dimension (attribute). 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 $D$ ($R_1$, $R_2$, ..., $R_n$), where $R_i$ is a relation denoted by $R_i$ ($d_1$, $d_2$, ..., $d_m$), $D$ 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 **Comparable:** Let the data items $a_i$ and $a_j \in R$, $a_i$ and $a_j$ be comparable (denoted by $a_i\ \varepsilon\ a_j$) if (and only if) they have no missing values in at least one identical dimension; otherwise $a_i$ is incomparable to $a_j$ (denoted by $a_i\ \varepsilon/\ a_j$).

## 4 SCSA algorithm

We have proposed a new efficient algorithm called *Sorting-based Cluster Skyline Algorithm* (SCSA) for deriving skylines in a database with incomplete data. The proposed algorithm consist of the five phases of Sorting and Filtering, Clustering and Grouping, Identifying Local Skylines, Selecting Superior Local Skylines, and Retrieving Final Skylines as illustrated in Fig. 2. These five phases are further explained in following subsections.

In order to illustrate the function of SCSA a sample run on an incomplete database has been conducted as demonstrated in Fig. 3. The database example contains 40 data items with seven dimensions in which some values are not present (marked as *).

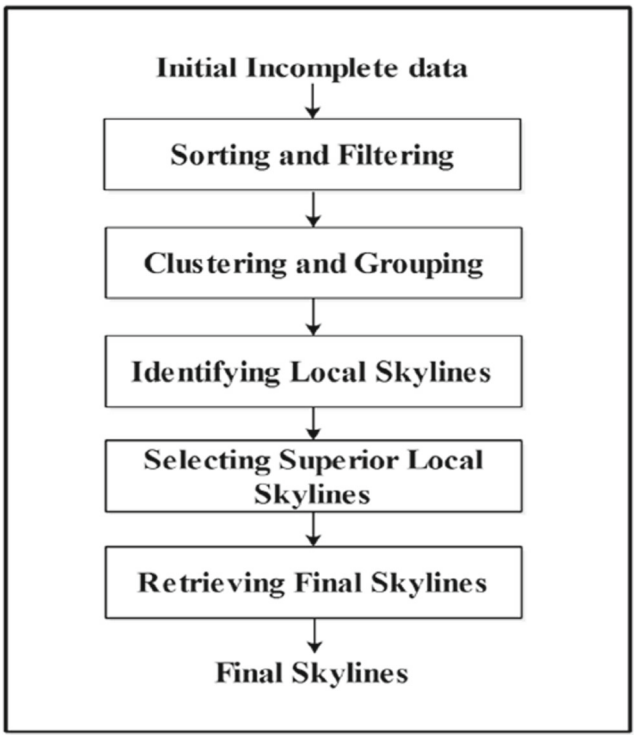



**Fig. 2** The phases of the proposed algorithm (SCSA)

### 4.1 Sorting and filtering

In this phase the data items of the initial dataset are sorted in descending order based on the values of each non-missing dimension before eliminating the dominated data items with a low value of *domination power* and discarding them. Data items with low *domination power* value are unlikely to contribute in forming the skyline results and thus, removing them before applying the skyline technique saves a large amount of unnecessary pairwise comparisons and reduces the overhead of the skyline computation process. The process starts by sorting the data items in each distinct list according to the values of each dimension. It is worth noticing here that only the *id* dimension of data items is stored in the lists, which are scanned in a round-robin fashion to count the *domination power* of each data item. This process continues until all the data items of the initial dataset have been read at least once. Those data items with a *domination power* value less than the user-defined *threshold* (*th*) are removed as they may be dominated by other data items in one single dimension. Hence, it is ascertained that the eliminated data items will not be part of the skyline result as they are most likely to be dominated by other data items with a *domination power* value greater than *th*. This filtration process helps simplify the skyline operation by reducing the number of pairwise comparisons between data items.

| ID | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| $m_0$ | * | 4 | 6 | 7 | 6 | 8 |
| $m_1$ | * | 6 | 2 | 7 | 6 | 1 |
| $m_2$ | 1 | * | 3 | 5 | 9 | * |
| $m_3$ | * | 5 | 7 | 9 | 1 | 1 |
| $m_4$ | * | 6 | 4 | * | 4 | 2 |
| $m_5$ | * | 2 | 4 | 4 | * | 8 |
| $m_6$ | * | 4 | * | 8 | 7 | 9 |
| $m_7$ | 7 | * | 7 | 4 | 3 | 6 |
| $m_8$ | 9 | * | 8 | 3 | * | 5 |
| $m_9$ | 9 | * | 6 | 6 | 7 | 3 |
| $m_{10}$ | 5 | * | 4 | 3 | 1 | 5 |
| $m_{11}$ | * | 4 | 2 | 4 | 4 | 5 |
| $m_{12}$ | 1 | * | 8 | 7 | * | 5 |
| $m_{13}$ | 5 | * | 5 | 4 | 4 | 4 |
| $m_{14}$ | 1 | 5 | * | 2 | * | 4 |
| $m_{15}$ | 3 | 5 | * | 5 | 4 | 6 |
| $m_{16}$ | 6 | 5 | * | * | 2 | 1 |
| $m_{17}$ | 4 | * | 5 | 4 | 3 | 6 |
| $m_{18}$ | 7 | * | 3 | 2 | 3 | 6 |
| $m_{19}$ | * | 7 | * | 4 | 2 | 5 |
| $m_{20}$ | 5 | 2 | * | 8 | 2 | 8 |
| $m_{21}$ | 7 | 1 | 7 | 9 | * | 7 |
| $m_{22}$ | * | 7 | 6 | * | 4 | 1 |
| $m_{23}$ | 5 | 5 | 2 | 6 | * | 5 |
| $m_{24}$ | 1 | * | 3 | * | 1 | 7 |
| $m_{25}$ | 2 | 4 | 3 | * | 3 | 8 |
| $m_{26}$ | 5 | 6 | 5 | 4 | * | 3 |
| $m_{27}$ | * | 5 | 3 | * | 2 | 5 |
| $m_{28}$ | 4 | * | 3 | 5 | * | 5 |
| $m_{29}$ | 4 | 8 | 6 | 9 | * | 8 |
| $m_{30}$ | * | 9 | 9 | 9 | * | 6 |
| $m_{31}$ | * | 9 | 4 | 8 | 4 | 8 |
| $m_{32}$ | 6 | 2 | 9 | 8 | * | 9 |
| $m_{33}$ | 3 | * | 3 | 7 | * | 5 |
| $m_{34}$ | 5 | 6 | 6 | 4 | * | 5 |
| $m_{35}$ | 2 | 6 | * | 4 | 1 | * |
| $m_{36}$ | 3 | * | 1 | 7 | 1 | * |
| $m_{37}$ | 4 | 6 | 6 | 5 | 6 | * |
| $m_{38}$ | 5 | 5 | 4 | 6 | 5 | * |
| $m_{39}$ | 7 | 8 | 4 | 5 | 5 | * |

**Fig. 3** An example of incomplete database

As part of the sample run, the initial dataset $D$ is sorted in descending order for each dimension $d_1 - d_6$. A set of lists $u_1 - u_6$ is constructed to store the sorted data items based on the corresponding dimension. Figure 4 depicts the sorted data items according to each dimension in the dataset. It can be observed that the six constructed lists $u_1 - u_6$ correspond with the number of dimensions $d_1 - d_6$ in the dataset $D$.

The data items of the constructed lists are scanned in a round-robin fashion to calculate the *domination power* for each data item. This process continues until all the data items of the dataset have been read at least once. The following equation demonstrates the formula of computing the *domination power* ($dp$) of the data item.

$$dp = \sum_{k=1}^{u} dt_i, \quad \text{iff } dt_i.k > dt_j.k$$

| $u_1$ | $u_2$ | $u_3$ | $u_4$ | $u_5$ | $u_6$ |
|---|---|---|---|---|---|
| $m_8$ | $m_{30}$ | $m_{30}$ | $m_3$ | $m_2$ | $m_6$ |
| $m_9$ | $m_{31}$ | $m_{32}$ | $m_{21}$ | $m_6$ | $m_{32}$ |
| $m_7$ | $m_{29}$ | $m_8$ | $m_{29}$ | $m_9$ | $m_0$ |
| $m_{18}$ | $m_{39}$ | $m_{12}$ | $m_{30}$ | $m_0$ | $m_5$ |
| $m_{21}$ | $m_{19}$ | $m_3$ | $m_6$ | $m_1$ | $m_{20}$ |
| $m_{39}$ | $m_{22}$ | $m_7$ | $m_{20}$ | $m_{37}$ | $m_{25}$ |
| $m_{16}$ | $m_1$ | $m_{21}$ | $m_{31}$ | $m_{38}$ | $m_{29}$ |
| $m_{32}$ | $m_4$ | $m_0$ | $m_{32}$ | $m_{39}$ | $m_{31}$ |
| $m_{10}$ | $m_{26}$ | $m_9$ | $m_0$ | $m_4$ | $m_{21}$ |
| $m_{13}$ | $m_{34}$ | $m_{22}$ | $m_1$ | $m_{11}$ | $m_{24}$ |
| $m_{20}$ | $m_{35}$ | $m_{29}$ | $m_{12}$ | $m_{13}$ | $m_7$ |
| $m_{23}$ | $m_{37}$ | $m_{34}$ | $m_{33}$ | $m_{15}$ | $m_{15}$ |
| $m_{26}$ | $m_3$ | $m_{37}$ | $m_{36}$ | $m_{22}$ | $m_{17}$ |
| $m_{34}$ | $m_{14}$ | $m_{13}$ | $m_9$ | $m_{31}$ | $m_{18}$ |
| $m_{38}$ | $m_{15}$ | $m_{17}$ | $m_{23}$ | $m_7$ | $m_{30}$ |
| $m_{17}$ | $m_{16}$ | $m_{26}$ | $m_{38}$ | $m_{17}$ | $m_8$ |
| $m_{28}$ | $m_{23}$ | $m_4$ | $m_2$ | $m_{18}$ | $m_{10}$ |
| $m_{29}$ | $m_{27}$ | $m_5$ | $m_{15}$ | $m_{25}$ | $m_{11}$ |
| $m_{37}$ | $m_{38}$ | $m_{10}$ | $m_{28}$ | $m_{16}$ | $m_{12}$ |
| $m_{15}$ | $m_0$ | $m_{31}$ | $m_{37}$ | $m_{19}$ | $m_{19}$ |
| $m_{33}$ | $m_6$ | $m_{38}$ | $m_{39}$ | $m_{20}$ | $m_{23}$ |
| $m_{36}$ | $m_{11}$ | $m_{39}$ | $m_5$ | $m_{27}$ | $m_{27}$ |
| $m_{25}$ | $m_{25}$ | $m_2$ | $m_7$ | $m_3$ | $m_{28}$ |
| $m_{35}$ | $m_5$ | $m_{18}$ | $m_{11}$ | $m_{10}$ | $m_{33}$ |
| $m_2$ | $m_{20}$ | $m_{24}$ | $m_{13}$ | $m_{24}$ | $m_{34}$ |
| $m_{12}$ | $m_{32}$ | $m_{25}$ | $m_{17}$ | $m_{35}$ | $m_{13}$ |
| $m_{14}$ | $m_{21}$ | $m_{27}$ | $m_{19}$ | $m_{36}$ | $m_{14}$ |
| $m_{24}$ | | $m_{28}$ | $m_{26}$ | | $m_9$ |
| | | $m_{33}$ | $m_{34}$ | | $m_{26}$ |
| | | $m_1$ | $m_{35}$ | | $m_4$ |
| | | $m_{11}$ | $m_8$ | | $m_1$ |
| | | $m_{23}$ | $m_{10}$ | | $m_3$ |
| | | $m_{36}$ | $m_{14}$ | | $m_{16}$ |
| | | | $m_{18}$ | | $m_{22}$ |

**Fig. 4** List of sorted arrays

As part of our experimental sample run of the database, the process works as follows: the first data item $m_8$ in list $u_1$ is read and its domination power $dp$ has increased by 1 which indicates the appearance of $m_8$. In the second iteration, the first data item $m_{30}$ in list $u_2$ is read and its $dp$ has also increased by 1. In the third iteration, the data item $m_{30}$ has been read again in list $u_3$ and its $dp$ value has further increased by 1 and becomes 2. This process continues until the reading of all data items of the dataset to compute their $dp$ is complete. Based on the example (Fig. 4), the process terminates at the 104th iteration where $m_{27}$ in list $u_2$ is read. The process has been terminated after reading $m_{27}$ as the termination conditions have become true (number of scanned data items = number of data items in the dataset).

The scanning process helps facilitate the filtration process by utilizing the *domination power* value of each data item. Figure 5 illustrates the scanned data items and their *domination power*.

During the filtration process, all data items with a *domination power* lower than the user-defined threshold ($th$) are eliminated from further processing as all data items with $dp < th$ are not likely to be part of the skyline result given that they are good only in one dimension. The main idea of filtering lies in exploiting the *domination power* value to further simplify the skyline process in incomplete databases with multiple

| Data item ID | Domination power (dp) |
|---|---|
| $m_8$ | 3 |
| $m_{30}$ | 4 |
| $m_3$ | 3 |
| $m_2$ | 2 |
| $m_6$ | 3 |
| $m_9$ | 4 |
| $m_{31}$ | 4 |
| $m_{32}$ | 4 |
| $m_{21}$ | 4 |
| $m_7$ | 4 |
| $m_{29}$ | 5 |
| $m_0$ | 4 |
| $m_{18}$ | 3 |
| $m_{39}$ | 3 |
| $m_{12}$ | 2 |
| $m_5$ | 1 |
| $m_{19}$ | 1 |
| $m_1$ | 3 |
| $m_{20}$ | 3 |
| $m_{22}$ | 3 |
| $m_{37}$ | 3 |
| $m_{25}$ | 1 |
| $m_{16}$ | 2 |
| $m_{38}$ | 3 |
| $m_4$ | 3 |
| $m_{10}$ | 2 |
| $m_{26}$ | 3 |
| $m_{13}$ | 3 |
| $m_{34}$ | 3 |
| $m_{11}$ | 1 |
| $m_{24}$ | 1 |
| $m_{35}$ | 1 |
| $m_{23}$ | 3 |
| $m_{33}$ | 1 |
| $m_{15}$ | 3 |
| $m_{36}$ | 1 |
| $m_{17}$ | 4 |
| $m_{14}$ | 1 |
| $m_{28}$ | 1 |
| $m_{27}$ | 1 |

**Fig. 5** Domination power of each data item

dimensions and large dataset size. These data items can thus be safely removed since the elimination of these data items will not affect the skyline results. We represent the *dp-list* set using the following formula.

$$dp{-}list = \{\forall dt_i : \text{iff } dp \text{ of } dti >= th\ \}.$$

where $i = 1, \ldots, n$ and $th$ is the minimum defined threshold value of $dp$.

For instance, if $th$ is set as 2, the data item $m_{25}$ has to be removed as its $dp < 2$. It is clear that $m_{25}$ will be dominated based on the comparable common non-missing dimension data items $m_{29}$, $m_{30}$, and $m_{32}$. Hence, $m_{25}$ should be removed before conducting unnecessary pairwise comparisons between data items. Similarly, the data items $m_5$, $m_{19}$, $m_{25}$, $m_{11}$, $m_{24}$, $m_{35}$, $m_{33}$, $m_{36}$, $m_{14}$, $m_{28}$, and $m_{27}$ have been removed since their $dp$ is <2. A sizeable number of data items has been removed prior to applying the skyline technique, which means that unnecessary pairwise comparisons have been avoided. Figure 6 depicts the remaining

| Data item ID | Domination power (dp) |
|---|---|
| $m_{29}$ | 5 |
| $m_{30}$ | 4 |
| $m_9$ | 4 |
| $m_{31}$ | 4 |
| $m_{32}$ | 4 |
| $m_{21}$ | 4 |
| $m_7$ | 4 |
| $m_0$ | 4 |
| $m_{17}$ | 4 |
| $m_8$ | 3 |
| $m_3$ | 3 |
| $m_6$ | 3 |
| $m_{18}$ | 3 |
| $m_{39}$ | 3 |
| $m_1$ | 3 |
| $m_{20}$ | 3 |
| $m_{22}$ | 3 |
| $m_{37}$ | 3 |
| $m_{38}$ | 3 |
| $m_4$ | 3 |
| $m_{26}$ | 3 |
| $m_{13}$ | 3 |
| $m_{34}$ | 3 |
| $m_{23}$ | 3 |
| $m_{15}$ | 3 |
| $m_2$ | 2 |
| $m_{12}$ | 2 |
| $m_{16}$ | 2 |
| $m_{10}$ | 2 |

**Fig. 6** Data items after filtration

data items sorted in descending order based on their *domination power*.

We argue that this sorting and filtering process simplifies the skyline process by eliminating unwanted data items before applying the skyline technique. In our running database example, 11 out of 40 data items are deleted before commencing with the skyline process, which translates into reducing 27% of the pairwise comparison process of the skyline. For the sake of simplicity and without losing generality, we assume that all data items with $dp$ value <2 must be removed from further processing. However, our approach can also accommodate cases where the user may chose a minimum $th$ value to be set to remove the dominated data items. For instance, in certain cases where the number of dimensions is large (meaning here more than eight dimensions) we can set the filtration condition to remove all data items with $dp < 3$ or 4. Therefore, a large number of dominated data items can be removed, which reduces the number of pairwise comparisons between data items in the skyline process.

Figure 7 illustrates the detail steps of sorting and filtering algorithm. In this algorithm, the data items of dataset $D$ are sorted in descending order for each dimension and stored in the constructed lists (steps 1–4). In step 5, a 2D array or *dp-list* is constructed to store the *ID* of all data items present in $D$ and their domination power value. A variable *AllDataItemsRead* is

***Algorithm* 1**

**Input:** *d*-dimensional Incomplete Dataset D
**Output:** A set of DataItems *dt* with their *domination power*, *dp-list*

```
1.  foreach dimension d_i in D do
2.          sort DataItems in descending order based on
            available values in d_i
3.          add the indices of the sorted DataItems dt based on d_i
            to List List.u_i
4.  end
5.  Construct 2D array called dp-list // dp-list stores the ID of
the DataItem dt and its domination power, dp
6.   set AllDataItemsRead = total number of DataItems in D
7.          DimCountDone= false
8.          foreach DataRow dr_i of List.u_i do
9.            if DimCountDone then
10.                   break
11.           end
12.           foreach Column u_j of dr_i do
13.                   if AllDataItemsRead > 0 then
14.                     if u_j read before then
15.                             Increment dp of u_j
16.                     else
17.                             add ID of u_j to dp-list
18.                             set dp of u_j=1
19.                             decrement AllDataItemsRead
20.                     end
21.                   else
22.                     DimCountDone = true
23.                     break
24.                   end
25.           end
26.         end
27.  foreach dataItem dt_i in dp-list do
28.         if dp of dt_i < 2 then
29.                   remove dt_i from dp-list
30.         end
31.  end
32.  return dp-list
```

**Fig. 7** The algorithm for sorting and filtering

initialized to denote the total number of data items in $D$ (step 6). In step 8 for each row $dr_i$ of *List* is selected. If *DimCountDone* is true the scanning process terminates (step 9), and if not the data item index of column $u_i$ of $dr_i$ is read (step 12). If *AllDataItemsRead* is greater than 0 (step 13), then $u_j$ is checked to determine whether it has been read before or not (step 14). If it has been read before, the *dp* of $u_j$ is incremented by 1 (step 15) and if not, $u_j$ is added to *dp-list* (step 17) and its *dp* is set to 1 (step 18) before *AllDataItemsRead* is decremented by 1 (step 19). If *AllDataItemsRead* is not greater than 0, *DimCountDone* is set as true (step 22) and the process is terminated (step 25). Steps 8–26 are repeated in order to read all the data item from *List* at least once.

The scanning process is executed in a round-robin fashion. By the end of the scanning process, the *dp-list* contains all the data items present in $D$ and their *domination power* value (*dp*). The *dp* indicates how many times a data item (*dt*) has been read. The maximum value of *dp* for each data item is equal to the number of non-missing dimensions for that particular data item. Steps 27–31 represent the filtration process, which starts by reading each data item present in *dp-list* (step 27). If *dp* of $dt_i$ is <2 (step 28), then it is removed from the *dp-list* (step 29). The process ends by returning the filtered *dp-list* (step 32).

## 4.2 Clustering and grouping

This phase aims at further simplifying the skyline process in a database with incomplete data by partitioning the data items into smaller clusters based on their *domination power* generated in the previous phase. The idea of clustering relies on grouping data items with similar *domination power* in one cluster. This is achieved by scanning the list of the remaining data items after filtration and placing the data items with similar *domination power* in one cluster. The number of created clusters equals to ($d$-$d$`- $dp_{min}$) where $d$ is the total number of dimensions in a database excluding the primary key dimension and $d$` is the number of dimensions with missing values. In addition, $dp_{min}$ denotes the minimum value of *dp* for those removed dominated data items. Distributing filtered data items into different clusters based on their *domination power* significantly reduces the number of pairwise comparisons necessary to generate the final skylines. This is due to the fact that clustering essentially applies the divide and conquer technique, which has been proven as an effective way of processing skyline queries in database systems [1, 2, 41–43].

Hence, this process helps reduce the number of data items to be compared with each other, which in turn helps avoid many unwanted pairwise comparisons without compromising the correctness of the skyline result.

The detail steps of clustering are further elaborated as follows. Firstly, the list of filtered data items is scanned to identify the highest domination power or *dp* value in the list before creating a new cluster $C_i$ that contains all data items with the highest *dp* value. Another new cluster $C_j$ may be created on the next highest *dp* value and contains the data items that have a value equal to the next highest *dp* value. This process continues until the *dp* value of the remaining data items is below the user-defined threshold value *th*. The following formula translates the process of creating clusters based on the domination power *dp* of the data items.

$$C_j = \{\forall dt_i : \text{iff } dp(dt_i) = hdp\ \}$$

Where *hdp* is the highest *dp* value in *dp-list*.

Data items in each cluster may have different *bitmap representation* which makes it very difficult to apply the skyline technique due to the incompatibility of the bitmap representation of data items in one cluster. This problem is caused by values missing in one or more dimension of data items, which makes it impractical to perform pairwise comparisons between data items that ensure that the transitivity property of skylines will always hold. Most importantly, it is inevitable to encounter the issue of cyclic dominance. Thus, these large clusters need to be further divided into smaller manageable groups to avoid the above-mentioned problems. The purpose of grouping is to create groups from each cluster based on the common bitmap representation of data items. We employed

the principle of *bitmap representation* that indicates the membership of the data items to the appropriate group. For instance, given a set of data items $a_1$, $a_2$, $a_3 \in$ a cluster $C_i$ we assume that each data item has three dimensions in total and one dimension with missing value $a_1$ (4, 5, *), $a_2$ (4, *, 9) and $a_3$ (1, 3, *). Based on the given data of item $a_1$, its bitmap representation is 110 where value 1 denotes that the value is present in that particular dimension while 0 indicates the value is missing for that particular dimension. Similarly, the bitmap representation of $a_2$ and $a_3$ is 101 and 110 respectively. Therefore, according to the bitmap representation of the above data items 2 groups, $G_1$ and $G_2$ must be created where $G_1$ contains $a_1$ and $a_3$ while $G_2$ contains $a_2$. Aggregating data items with the same *subspace* allow for smooth pairwise comparisons, thus sustaining the transitivity property of the skyline and excluding the problem of cyclic dominance [2, 4, 41, 42]. The following formula demonstrates the process of adding data items to their corresponding groups.

$$G_j = \{\forall dt_i : \text{iff } dt_i.bitmap = G_j.bitmap\}$$

Grouping also helps reduce the number of pairwise comparisons by limiting the number of data items to be compared against each other in one group rather than comparing the entire data items in one cluster. This simplifies the skyline process by decreasing the number of pairwise comparisons, which in turn reduces the execution time of the skyline process. We believe clustering and grouping significantly contributes to improving the performance of the skyline process by eliminating a large number of unnecessary pairwise comparisons.

Figure 8 demonstrates the result of the clustering process conducted on our running database example. From the figure it can be observed that four distinct clusters have been created ($C_1$, $C_2$, $C_3$, $C_4$). Cluster $C_1$ contains one data item $m_{29}$ with $dp = 5$, which means that $m_{29}$ has appeared five times in the list of sorted arrays. Likewise, clusters $C_2$, $C_3$, and $C_4$ contain data items with the next highest $dp$ values of 4, 3 and 2 respectively. Furthermore, the data items belonging to each cluster are further separated and distributed in distinct groups based on similar *bitmap representation*.

| Cluster $C_1$ $dp = 5$ | | Cluster $C_2$ $dp = 4$ | | Cluster $C_3$ $dp = 3$ | | Cluster $C_4$ $dp = 2$ | |
|---|---|---|---|---|---|---|---|
| $m_{29}$ | 5 | $m_{30}$ | 4 | $m_8$ | 3 | $m_2$ | 2 |
| | | $m_9$ | 4 | $m_3$ | 3 | $m_{12}$ | 2 |
| | | $m_{31}$ | 4 | $m_6$ | 3 | $m_{16}$ | 2 |
| | | $m_{32}$ | 4 | $m_{18}$ | 3 | $m_{10}$ | 2 |
| | | $m_{21}$ | 4 | $m_{39}$ | 3 | | |
| | | $m_7$ | 4 | $m_1$ | 3 | | |
| | | $m_0$ | 4 | $m_{20}$ | 3 | | |
| | | $m_{17}$ | 4 | $m_{22}$ | 3 | | |
| | | | | $m_{37}$ | 3 | | |
| | | | | $m_{38}$ | 3 | | |
| | | | | $m_4$ | 3 | | |
| | | | | $m_{26}$ | 3 | | |
| | | | | $m_{13}$ | 3 | | |
| | | | | $m_{34}$ | 3 | | |
| | | | | $m_{23}$ | 3 | | |
| | | | | $m_{15}$ | 3 | | |

**Fig. 8** Data items after clustering

Figure 9 depicts the output of the grouping process applied on the created clusters of our database sample run. It can be observed that only one group with the *bitmap representation* (111101) has been created from cluster $C_1$. Similarly, four groups ($G_1$, $G_2$, $G_3$, $G_4$) with different *bitmap representations* have been created based on the data items of cluster $C_2$. It should be obvious that creating smaller groups further simplifies the skyline process. This divide and conquer technique further lowers the number of data items to be compared against each other.

Figure 10 details the steps of employing the data clustering algorithm. The input of the algorithm includes the *dp-list* and the initial dataset with incomplete data, while the output consists of a list of distinct clusters. The algorithm works as follows: Initially, each data item $dt_i$ in the *dp-list* is read. If the $dp$ value of the data item $dt_i$ is equal to the $dp$ value of any created cluster $C_j$ (step 2), the data item is inserted in the corresponding cluster $C_j$ (step 3). A new cluster $C_k$ is created (step 5) and

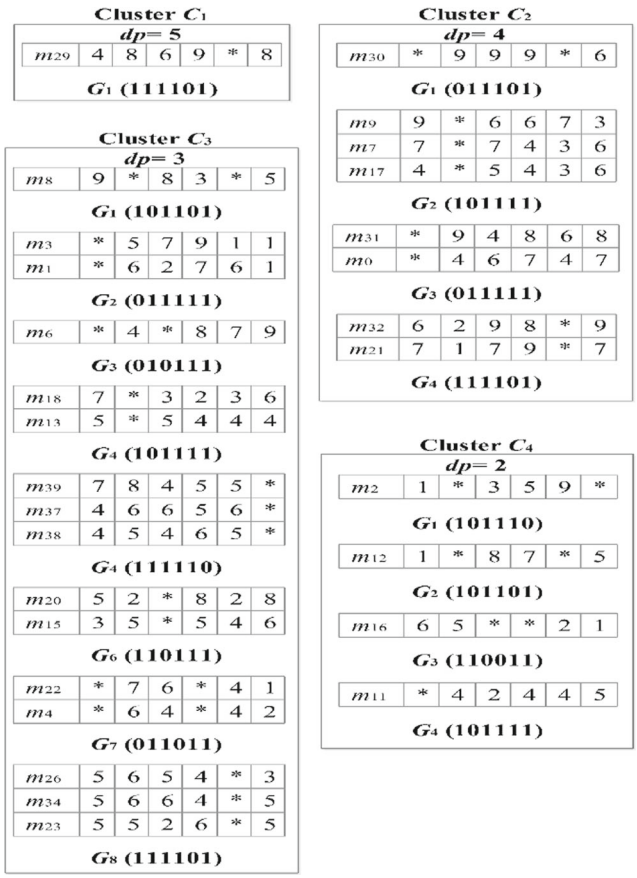



**Fig. 9** List of clusters divided into groups

***Algorithm* 2**

**Input:** *dp-list,* Initial Dataset D.
**Output:** List of Clusters based on *domination power (dp)*

```
1. foreach dataItem dti in dp-list do
2.     if dp of dti == dp of any existing cluster Cj then
3.         insert dti into Cj
4.     else
5.         create new cluster Cj
6.         insert dti into Cj
7.     end
8. end
9. return list of distinct clusters
```

**Fig. 10** The algorithm for clustering

the data item $dt_i$ inserted into cluster $C_k$ (step 6). This process continues until each data item in the *dp-list* is read and inserted into one of the created clusters. Finally, the algorithm returns a list of distinct clusters, each cluster containing data items with similar *dp* value (step 9).

Figure 11 clarifies the steps of creating groups from the constructed clusters. The algorithm input consists of a cluster containing data items with different bitmap representation whereas the output of the algorithm includes a list of distinct groups based on bitmap representation. Each data item $dt_i$ belonging to cluster $C_j$ is read (step 1). If the bitmap representation of the data item *dti* is equivalent to the bitmap representation of any previously created group $G_k$ (step 2) before inserting the data item into group $G_k$ (step 3). Else, create a new group $G_k$ (step 5) and insert the data item $dt_i$ into the corresponding group $G_k$ (step 6).

Eventually, a set of groups with distinct bitmap representation is returned (step 9). It should be noted that this process is running simultaneously on all clusters since they are independent of each other and creating groups from one cluster bears no effect on the other cluster relationships. Therefore, this process can be conducted in parallel, which speeds up the skyline process in a database with incomplete data.

We noticed that the idea of generating the *domination power* and clustering data items based on their *domination power* values is very beneficial. It significantly reduces the complexity of the skyline process in incomplete data by lowering the number of data items to be considered for pairwise comparison. Therefore, the number of pairwise comparisons that need to be conducted is further decreased, which in turn results in less processing time. Constructing groups based on *bitmap representation* also further simplifies the skyline process in incomplete data, which ensures that the transitivity property of the skyline technique is kept and that the problem of *cyclic dominance* issue is avoided.

***Algorithm* 3**

**Input:** A cluster $C_j$.
**Output:** List of Groups of cluster $C_j$ [ ($C_j.G_1$, … $Cj.G_m$)]

```
1. foreach DataItem dti of Cj do
2.     if bitmap representation of dti == bitmap
       representation of any existing group Gk of Cj then
3.         insert dti into Gk
4.     else
5.         create new group Gk
6.         insert dti to Gk
7.     end
8. end
9. return list of clusters of each cluster
```

**Fig. 11** The algorithm for grouping

## 4.3 Identifying local skylines

In this phase the local skylines of each cluster are identified and all the dominated data items removed and excluded from further processing. This helps reduce the number of pairwise comparisons that need to be made in order to identify the final skyline of the dataset. The process begins with identifying the skylines of each group that belong to each created cluster. This step is important as it ensures the smooth execution of the pairwise comparison process between data items since all group data items have similar bitmap representation. Thus, the issue of losing transitivity property and cyclic dominance can be avoided. The skylines of the groups are further compared with each other to derive the local skylines of the cluster. This process reduces the number of data items to be considered in the subsequent phases to that of the non-dominated data items. Hence, the complex skyline process is considerably simplified. Its process runs parallel whereby the local skylines of each cluster are generated simultaneously. Therefore, exploiting the parallel execution intensifies the skyline process and shortens the total execution time.

In our database sample running, data items present in each group of the constructed clusters are compared to each other in order to identify group skylines. In Fig. 9 is shown group $C_1.G_1$ of the cluster where $C_1$ contains only one data item $m_{29}$ and is thus considered as the skyline of group $C_1.G_1$. Similarly, there are the four groups $C_2.G_1$, $C_2.G_2$, $C_2.G_3$ and $C_2.G_4$ created from cluster $C_2$. The skylines of each group are identified by comparing all data items in one group to another. The dominated data items in each group are discarded and eliminated from further processing. In a similar fashion the group skylines of clusters $C_3$ and $C_4$ are identified. It is important to notice here that the process of identifying group skylines of all clusters are performed simultaneously. This is possible as all the data items in the groups are independent of each other. This parallel execution speeds up the process of identifying the group skylines of the clusters. These group skylines of the clusters are then compared to each other in order to determine the local skyline of each cluster. Figure 12 shows the local skylines of clusters for our database running sample. We can conclude that most dominated data items have been removed and excluded from further processing.

**Fig. 12** Local skylines of each cluster

**Cluster $C_1$**

| | | | | | | |
|---|---|---|---|---|---|---|
| $m_{29}$ | 4 | 8 | 6 | 9 | * | 8 |

**Cluster $C_2$**

| | | | | | | |
|---|---|---|---|---|---|---|
| $m_{30}$ | * | 9 | 9 | 9 | * | 6 |
| $m_{31}$ | * | 9 | 4 | 8 | 6 | 8 |
| $m_0$ | * | 4 | 6 | 7 | 4 | 8 |
| $m_{32}$ | 6 | 2 | 9 | 8 | * | 9 |
| $m_{21}$ | 7 | 1 | 7 | 9 | * | 7 |

**Cluster $C_3$**

| | | | | | | |
|---|---|---|---|---|---|---|
| $m_3$ | * | 5 | 7 | 9 | 1 | 1 |
| $m_1$ | * | 6 | 2 | 7 | 6 | 1 |
| $m_6$ | * | 4 | * | 8 | 7 | 9 |
| $m_{39}$ | 7 | 8 | 4 | 5 | 5 | * |
| $m_{37}$ | 4 | 6 | 6 | 5 | 6 | * |
| $m_{38}$ | 4 | 5 | 4 | 6 | 5 | * |
| $m_{34}$ | 5 | 6 | 6 | 4 | * | 5 |

**Cluster $C_4$**

| | | | | | | |
|---|---|---|---|---|---|---|
| $m_{12}$ | 1 | * | 8 | 7 | * | 5 |
| $m_{16}$ | 6 | 5 | * | * | 2 | 1 |

Figure 13 describes the detail steps of identifying the group skylines process. The algorithm input constitutes a set of data items in group $G_j$ while the output of the algorithm constitutes a set of group skylines. The algorithm works as follows: In step 1 each data item $dt_i$ of a group $G_j$ is read. A pointer *ptr* indicate a data item $dt_i$ (step 2). Each data item $dt_k$ is read where $k = i + 1$ in $G_j$ (step 4). A pointer *cw* points to $dt_k$ (step 5). The values of non-missing dimensions between *ptr* and *cw* are compared by calling algorithm 6 (step 6). If the comparison results return 1 (step 7), the data item pointed by *cw* from $G_j$ is removed (step 8) and if the comparison results return 2 (step 9), the set *IsDominated* is true (step 10). This process is repeated until all the data items of $G_j$ are compared to the data item pointed to by *ptr*. If a data item of *ptr* is dominated by any data item of *cw* (step 13), the data item pointed by *ptr* is removed from $G_j$ (step 14). This process continues until all the remaining data items of the group are compared to each other. The algorithm ends by returning the group skylines.

***Algorithm* 4**

**Input:** A group of data items $G_i$
**Output:** Group Skylines

1. **foreach** $dt_i$ of $G_j$ **do**
2. let a pointer $ptr = dt_i$ //*i =0, …, n, where n is the size of group*
3. *IsDominated* = false
4. **foreach** $dt_k$ of $G_j$ **do** // $k$=$i$+1, …, $n$
5. let a pointer $cw = dt_k$
6. *DomValue* = Compare *(ptr, cw)* //***Algorithm 6***
7. **if** *DomValue* == 1 **then**
8. remove *cw* from $G_j$
9. **else if** *DomValue* == 2 **then**
10. *IsDominated* = true
11. **end**
12. **end**
13. **if** *IsDominated* **then**
14. remove *p* from $G_j$
15. **end**
16. **end**
17. **return** group skylines

**Fig. 13** Algorithm for group skylines

Figure 14 details the steps of an algorithm that is used to identify the local cluster skylines. The input of the algorithm consists of a set of groups of a cluster, while the output consists of a set of local skylines of a cluster. The algorithm works as follows: In step 1 a group $G_i$ of clusters $C_j$ is selected. The data item $dt_k$ of $G_i$ is read (step 2). A pointer *ptr* denotes a data item $dt_k$ (step 3). Then, a new group $G_{j++}$ of $C_j$ is selected (step 5) before each data item $dt_l$ of group $G_{j++}$ is read (step 6) followed by setting a pointer *cw* that indicates the data item $dt_l$ (step 7). A pairwise comparison between data items pointed to by *ptr* and *cw* is conducted (step 8). If the result returned from the comparison is 1 (step 9), the data item pointed to by *cw* is removed from $G_{j++}$ (step 10) and if the result of the comparison is 2 (step 11), then *IsDominated* is true (step 12). This process is repeated until all data items of $G_{j++}$ are compared to the data item pointed to by *ptr* (steps 6–14). The process continues by comparing *ptr* with the data items of the other groups of $C_j$ (steps 5–15). If a data item pointed to by *cw* of any group dominates the data item pointed to by *ptr* (step 16), the data item pointed to by *ptr* is removed from $G_j$ (step 17). This process continues until all the remaining data items of group $G_j$ are compared to the data items of the other groups (steps 2–19). This process ends once all groups have been compared to each other and the skylines of the cluster have been identified (step 1–20). The process ends by returning the local skylines of the cluster (step 21).

Figure 15 presents the steps of the pairwise comparison algorithm. The input for the algorithm consists of two different data items while the output constitutes an integer value that denotes the result of the comparison. The process begins with reading the values of all dimensions of $dt_i$ (step 1). If the value of a dimension $d_k$ of $dt_i$ or $dt_j$ is missing or the value of $d_k$ of $dt_i$ and $dt_j$ is equal (step 2), it is continued by skipping the next statements and jump to step 1 (step 3) and if the value of $d_k$ of $dt_i$ is better than the value of $d_k$ of $dt_j$ (step 5), then *PDom* is true (step 6). However, if the value of $d_k$ of $dt_i$ is worse than the value of $d_k$ of $dt_j$ (step 7), then *CDom* is true (step 8). If *PDom* and *CDom* are true (step 9), then the return is 0 (step 10). This process continues until all dimensions of $dt_i$ and $dt_j$ are compared to each other (steps 1–13). If only *PDom* is true (step

***Algorithm* 5**

**Input:** Set of groups of a cluster, $C_j$ [$C_j.G_1$, $C_j.G_2$, …, $C_j.G_n$].
**Output:** Local skylines of a cluster, $CLS_j$

```
1.  foreach Group Gi of Cj do
2.    foreach dtk of Gi do
3.      let a pointer ptr= dtk
4.      IsDominated == false
5.      foreach Group Gi++ do
6.        for (int l= 0; l < n; l++)    //n = number of data
          items in Gi++
7.          let a pointer cw = dtl
8.          DomValue = Compare (ptr, cw) //Algorithm 6
9.            if DomValue == 1 then
10.                 remove cw from Gi++
11.           else if DomValue == 2 then
12.                 IsDominated = true
13.           end
14.       end
15.     end
16.     if IsDominated then
17.       remove ptr from Gi
18.     end
19.   end
20. end
21. return local skylines of a cluster
```

**Fig. 14** Algorithm for cluster local skylines

14), then the return is 1 (step 15). However, if only *CDom* is true (step 16), then the return is 2 (step 17).

We can conclude at this point that applying the skyline technique during the local skyline identification phase helps eliminate many dominated data items. As evident from our database example, only 15 data items out of the remaining 29 data items of the dataset are left for further processing. This represents up to 50% reduction in the dataset.

***Algorithm* 6**

**Input:** two data items ($dt_i$, $dt_j$)
**Output:** integer value

```
1.  foreach dk of dti do
2.      if dti.dk == 0 || dtj.dk == 0 || dti.dk == dtj.dk then // 0
        denotes missing value
3.        continue
4.      else
5.              if dti.dk > dtj.dk then
6.                      PDom = true
7.              else if dti.dk < dtj.dk then
8.                      CDom = true
9.              if PDom && CDom == true then
10.                     return 0;
11.             end
12.     end
13. end
14. if PDom == true then
15.     return 1
16. else if CDom == true then
17.     return 2
18. end
```

**Fig. 15** Comparison algorithm

## 4.4 Selecting superior local skylines

The main purpose of this phase is to further optimize the process of identifying the skylines in the incomplete database. This is achieved by eliminating some of the local cluster skylines produced during the previous phase before comparing them to the local skylines of other clusters. This step is necessary in order to avoid many unwanted pairwise comparisons while identifying the final skylines. Removing the local skylines of a cluster before comparing them to the local skylines of other clusters allows scanning the available values in all dimensions and retain only the data items with the highest value in any of non-missing dimensions. In other words, we first read the value of each data item in sequence by accessing only one dimension and mark each data item with the highest value in that particular dimension. This process continues by accessing the remaining non-missing dimensions belonging to each data item and marking the data items with the highest value. This means that all unmarked data item(s) are eventually removed from the local skyline of the cluster. This optimization step can greatly reduce the number of local skylines of the clusters that need to be considered in deriving the final skylines and also reduces the number of pairwise comparisons that need to be conducted in order to identify the final skylines. This process is carried out concurrently on all clusters and makes the skyline process more efficient. The following formula generalizes the process of identifying the superior local skyline of each cluster.

$$SLS(C_i) = \left\{ dt_i : max(dt_i.d_j) \right.$$
$$\left. whrer\ 0 \geq d_j \leq u \right\}$$

Figure 16 details the process of deriving superior skylines of clusters. The data items with shaded cells represent the superior skyline of clusters. Here, $m_{30}$, $m_{31}$, $m_{32}$, and $m_{21}$ constitute the superior skylines of cluster, $C_2$, and $m_0$ should be removed from $C_2$ since it has no highest value in any of its non-missing dimensions and will be dominated by the local skyline $m_{29}$ of cluster $C_1$. Therefore, unnecessary pairwise comparisons between $m_{29}$ and $m_0$ are avoided, which in turn accelerates the skyline process. Similarly, data items $m_1$, $m_{37}$, $m_{38}$, and $m_{34}$ are removed since their values in the non-missing dimensions are not the highest and will be dominated by the local skylines of the other clusters, $C_1$ and $C_2$.

Figure 17 illustrates the superior local skylines of each cluster. We can conclude that $m_{29}$ is the superior skyline of cluster $C_1$. Similarly, $m_{30}$, $m_{31}$, $m_{32}$ and $m_{21}$ are the superior skylines of cluster $C_2$, $m_3$, $m_6$, and $m_{39}$ are the superior skylines of $C_3$, and $m_{12}$ and $m_{16}$ are the superior skylines of cluster $C_4$.

**Fig. 16** The process of selecting superior local skylines

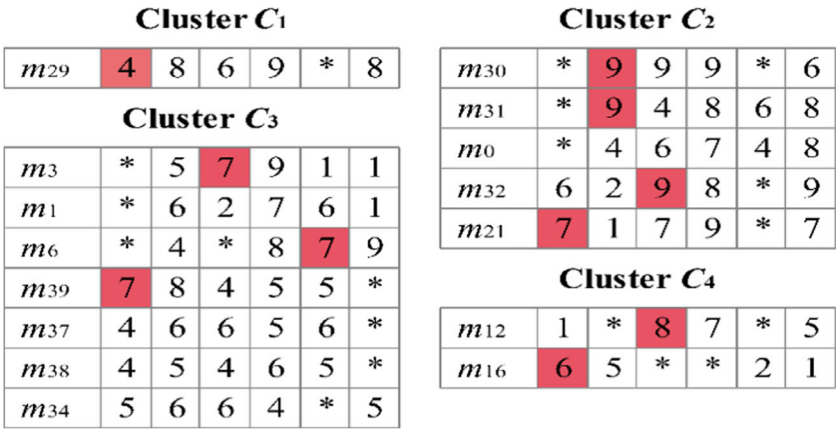



Figure 18 details the steps of selecting the superior local skylines of the cluster algorithm. The input of the algorithm includes the local skylines of a cluster while the output consists of a list of superior local skylines of a cluster. The algorithm starts by selecting a dimension *di* (step 4) followed by setting *d_max* as the highest available value in $d_i$ (step 5). Each value in dimension $d_i$ is read (step 6), and if the value is equal to *d_max* and the corresponding data item $dt_j$ is unmarked (step 8), the data item $dt_j$ is marked (step 9). This process continues until all the data items of the cluster $C_i$ are read. The unmarked data items are removed from the list of the cluster of the local skylines (step 14). Eventually, the remaining data items in *CLS* are retrieved as the superior local skyline of cluster $C_i$ (step 15).

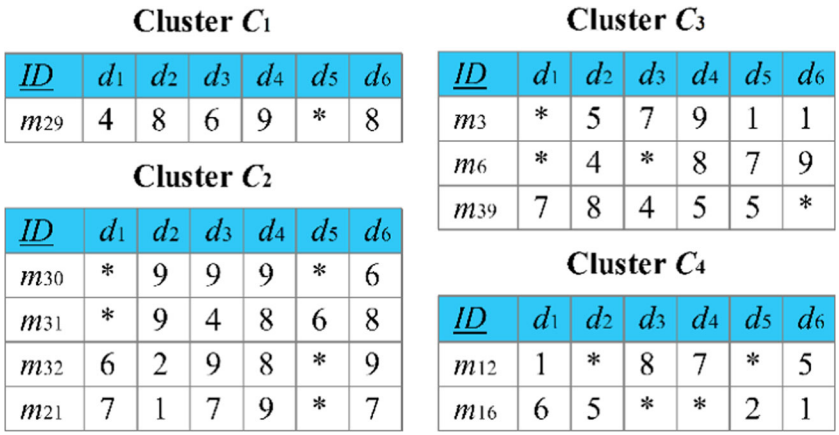
The phase of selecting the superior local skyline includes a new optimization technique that significantly reduces the number of data items to be considered in the skyline process. The idea is novel and attempts to exploit the maximum available value in any dimension when selecting the superior local skylines. Therefore, we believe that this optimization technique significantly simplifies the skyline process in incomplete datasets. From our database example it can be learned that out of 15 data items, five data items are eliminated from further processing. This represents a 33% reduction of the entire dataset.

### 4.5 Retrieving final skylines

In the final phase of our proposed approach to processing the skyline queries in incomplete data, we identify the global skylines of the entire dataset. In other words, we retrieve the final skylines that are not dominated by any data item in the whole dataset. This objective is accomplished by comparing the superior local skylines of each cluster to each other and determining the final skylines. The process of this phase is identical to the process of identifying the local skylines of each cluster as already described in Section 4.3. In order to derive the final skylines, algorithm 5 is used. The input consisting of the list of superior local skylines of each cluster, whereas the output consists of a set of final skylines, *final_S*.

In our database running sample, the superior local skylines of cluster $C_1$ are compared to the superior local skylines of $C_2$, $C_3$, and $C_4$. If the local skyline of cluster $C_1$ dominates the superior local skylines of $C_2$, $C_3$, and $C_4$, the superior local skylines are removed, and if the superior local skyline of $C_2$, $C_3$, and $C_4$ dominates the superior local skyline of $C_1$, it will be removed at the end of the iteration. Similarly, the superior local skylines of $C_2$ are compared to the remaining superior local skylines of $C_3$ and $C_4$. Finally, the superior local skylines of $C_3$ are compared to the remaining superior local skylines of $C_4$. Eventually, the remaining superior local skylines of all

**Fig. 17** Superior local skylines of each cluster



***Algorithm* 7**

**Input:** Cluster Local Skylines, $CLS = \{LS_1, \ldots, LS_n\}$, of $C_i$
**Output:** Superior Local Skylines, $SLS = \{SLS_1, \ldots, SLS_n\}$, of $C_i$

```
1.  let no_of_dim = 1
2.  while (no_of_dim != m) // m = total number of
        dimensions
3.  {
4.      select a dimension, di // i = 1, …, m
5.      let d_max = highest value in di
6.      for (j= 1, j< n , j++)  // n= total number of data items
          in Ci
7.        {
8.          if (dtj.di == d_max && dtj is unmarked ) then
9.              mark dtj
10.         end
11.       }
12.     no_of_dim++;
13.  }
14.  Remove all unmarked data items from CLS of Ci
15.  return SLS of Ci
```

**Fig. 18** Algorithm for identifying superior local skylines

clusters are reported as the final skylines, *final_S* of the dataset. Figure 19 depicts the final skylines of given dataset *D*.

## 5 Experimental environment

In order to evaluate the efficiency and the performance of our proposed approach (SCSA) developed for processing skyline queries in incomplete databases, we have compared our approach to the most recent skyline approaches designed for incomplete databases such as *SPQ* [43] *Incoskyline* [42], *SIDS* [37], and *Iskyline* [2].

All approaches considered in this research work have been implemented using C# programming language. A comprehensive and intensive set of experiments has been conducted on *i*3 1.6*GHz* PC with 3GB memory and Windows 7 32bit platform. Since it is argued that the skyline queries is a CPU exhaustive process [1, 2, 12, 13, 35, 37, 39, 42], the experiments of this research work involved the two performance metrics of the number of pairwise comparisons between the data items and the processing time. In all experiments these

| ID | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| $m_{29}$ | 4 | 8 | 6 | 9 | * | 8 |
| $m_{30}$ | * | 9 | 9 | 9 | * | 6 |
| $m_{31}$ | * | 9 | 4 | 8 | 6 | 8 |
| $m_{21}$ | 7 | 1 | 7 | 9 | * | 7 |
| $m_6$ | * | 4 | * | 8 | 7 | 9 |

**Fig. 19** Final skylines

two metrics are measured by varying the number of dimensions that belong to the database, the number of dimensions with incomplete data, and the total size of the database. Synthetic and real datasets are used in this paper. Two types of synthetic datasets are generated and chosen to run the experiments. First, an independent dataset is generated whose values in one dimension are unrelated to the values of other dimensions. The second synthetic dataset is a correlated dataset whose values in one dimension are influenced by the values of the other dimensions. Another three types of the real dataset are selected, which include NBA, MovieLens, and CoIL 2000 insurance company datasets. These datasets are more realistic and are frequently used by researchers in the area of processing skyline queries in complete and incomplete database systems [1, 2, 4, 12, 13, 15, 37, 39, 42]. Also, some of them suit our experimental conditions as they are initially incomplete (MovieLens and CoIL 2000 insurance company). We opted for the query statement when retrieving the skylines with the highest values for the sake of simplicity and without losing generality. Table 2 summarizes the range of parameter values for the synthetic and real datasets used to evaluate the proposed approach for handling skylines queries in incomplete databases presented in this paper.

### 5.1 Experimental results

This section highlights the experimental results performed on the synthetic and real datasets for our proposed approach of processing skyline queries in incomplete databases. In this section, we attempt to investigate the impact of database dimensionality (number of dimensions) and the influence of database cardinality (dataset size) on the process of pairwise comparison and the processing time for skyline evaluation. We argue that these are the most crucial parameters that influence the skyline query processing [1, 2, 12, 15, 24, 37, 42, 44–46].

#### 5.1.1 Effect of number of dimensions

It has been evidenced in the literature of skyline queries that the number of dimensions highly influences the skyline query process [2, 37, 42]. Therefore, the first set of experiments examine in particular the impact of the number of dimensions on the process of pairwise comparison in order to derive the skylines for a database with incomplete data. In this section, we illustrate the experimental results for the synthetic and real datasets used throughout the paper. Figure 20a and b illustrate the results for the independent and correlated synthetic dataset in which the number of dimensions vary from 4 to 12 and the dataset size is fixed to 300 KB. From the figures, it is obvious that our approach consistently outperforms the *SPQ*, *Incoskyline*, *SIDS*, and *Iskyline* for both types of synthetic datasets. Furthermore, Fig. 20c presents the experimental

**Table 2** The parameter settings of the synthetic and real datasets in the experiments for incomplete database

| Dataset Name | | Parameter settings | | |
|---|---|---|---|---|
| | | No. of dimensions (*d*) | No. of dimensions with missing values (*d`*) | Dataset Size (*KB*) |
| Synthetic | Independent | 5, 7, 9, 11, 13 | 4, 6,8,10, 12 | 100, 200, 300, 400, 500, 600 |
| | Correlated | 5, 7, 9, 11, 13 | 4, 6,8,10, 12 | 100, 200, 300, 400, 500, 600 |
| Real | NBA | 6, 8, 10, 12, 14, 16, 18 | 3, 5, 7, 9, 11, 13, 15 | 40, 80, 120, 160, 200 |
| | CoIL 2000 | 4, 6, 8, 10, 12, 14, 16, 18, 20, 22 | 3, 5, 7, 9, 11, 13, 15, 17, 19, 21 | 50, 100, 150, 200, 250, 300 |
| | MovieLens | 4 | 3 | 400, 800, 1200, 1600, 2000 |

results on the NBA real dataset depicting the number of pairwise comparisons between data items executed during the skyline process. For this set of experiments, the dataset size is fixed at 120 KB, while the number of dimensions varies between five to 17, including the dimensions with missing values.

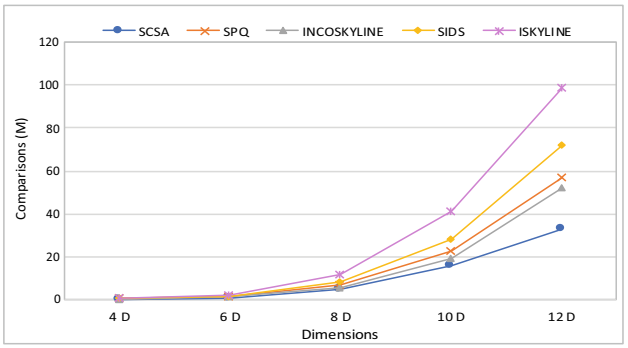
From the NBA dataset, we conclude that our approach is superior to that of *SPQ, Incoskyline*, *SIDS*, and *Iskyline*. We can also observe that the performance of our approach is better than *SPQ* and *SIDS* as the number of dimensions increases to more than 9. Figure 20d illustrates the experiment result on the CoIL 2000 insurance company real dataset. This set of experiments evaluates the existing approaches by varying the number of dimensions in the range of 3 to 21 and fixing the dataset size to 150 KB. We can observe that our approach is superior in all cases in terms of the number of executed pairwise comparisons. Since the MovieLens dataset consists of only four dimensions it is not used in this experiment.

It can be observed that *SCSA* shows that the number of pairwise comparisons in correlated, NBA and CoI is fewer than in independent datasets since most of the dominated data items are eliminated after filtration or during the identification of the group skylines, which helps reduce most of the pairwise comparisons across groups and clusters.

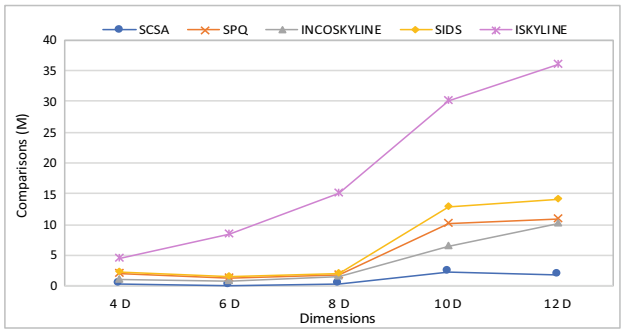
As for the experiment results, we observed that the *SCSA* is best compared to the other approaches (*SPQ*, *Incoskyline*, *SIDS*, and *Iskyline*) in terms of raising the number of dimensions and its minor influence on the number of pairwise comparisons. Our proposed method prunes the initial dataset by applying the sorting and filtration processes before applying the skyline process, which leads to eliminating many unwanted data items. The idea of filtration relies on exploiting the concept of generating *domination power (dp)* meaning the number of dimensions in which the data items is counted as a skyline. Furthermore, selecting superior local skylines by exploiting the highest value found in each dimension to help eliminating the dominated data items is employed in our approach. These two techniques have significantly contributed towards reducing the number of pairwise comparisons while identifying the final skylines. We also observed that *Iskyline* constitutes the least efficient technique due to its complicated process that results in deriving many local skylines in each distinct cluster. Furthermore, *Iskyline* also derives a large number of virtual skylines from different local skylines, which are compared to the local cluster skylines to identify the candidate skylines. These processes culminate in exhaustive pairwise comparisons among the entire dataset in order to retrieve final skylines. Nevertheless, the other three approaches, *SPQ*, *Incoskyline* and *SIDS* perform better than *Iskyline* for all datasets. However, these two approaches are worse than our proposed approach, *SCSA*.

Figure 21a, b, c, and d depict the processing time of all the different approaches to generate the skylines on the independent, correlated, NBA, and CoIL 2000 insurance company datasets. The parameter settings of this set of experiments are the same as the previous experiments for the synthetic dataset (independent and correlated) and the real dataset (NBA and CoIL 2000 insurance company). Figure 21a and b demonstrate that *SCSA* requires less processing time to identify the final skylines for both synthetic datasets. We have eliminated the dominated data items before applying the skyline technique in order to lessen the number of pairwise comparisons, which in turn decreases the processing time of computing the skylines. Most importantly, our approach incorporates the simultaneous run when evaluating the skylines of groups and clusters in order to accelerate the skyline identification process. From the figures we also learn that *Iskyline* underperforms in all cases, and its performance deteriorates when the number of dimension increases to six for the independent dataset (Fig. 21a) and increases to eight for the correlated dataset (Fig. 21b). However, *Incoskyline* performs only slightly worse than our approach on an independent dataset, and its performance declines when the number of dimensions is larger than eight on the correlated dataset. For NBA and CoIL 2000 insurance company real datasets, the parameter settings (dataset size and number of dimensions) are same as the previous experiment (Fig. 21c and d). Both figures show that our approach is superior to the other approaches in all the given cases.

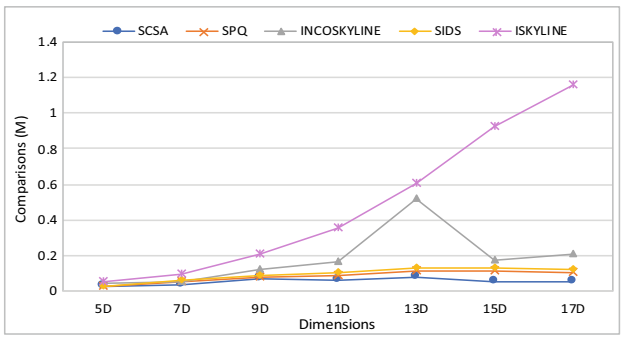
It is obvious that the increasing number of dimensions has an insignificant impact on the performance of our approach. This is due to the fact that applying the sorting and data



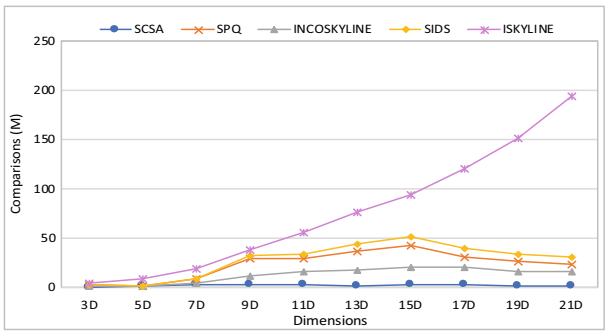

a) Independent




b) Correlated

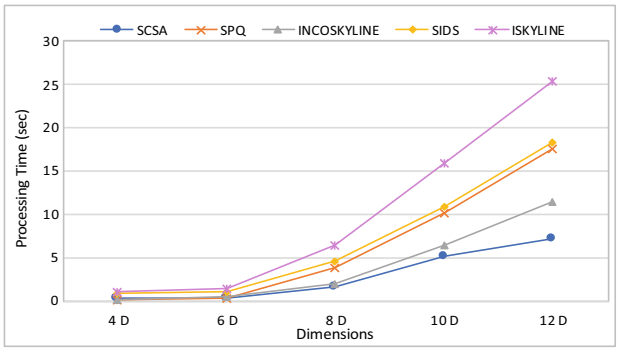



c) NBA

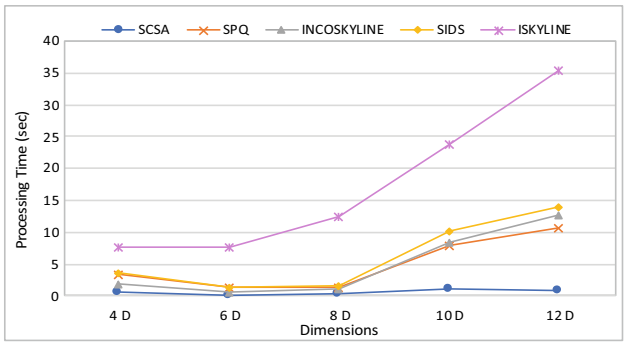



d) CoIL Insurance Company

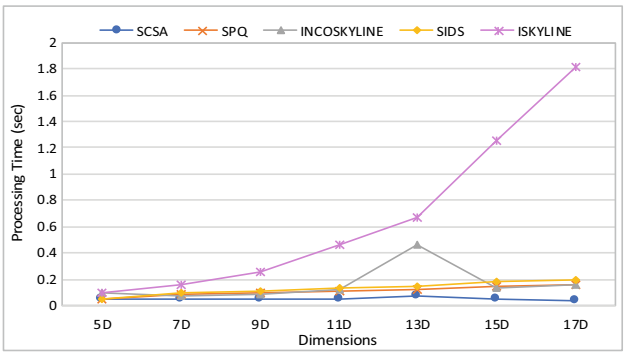
**Fig. 20** The effect of number of dimensions on the number of pairwise comparisons



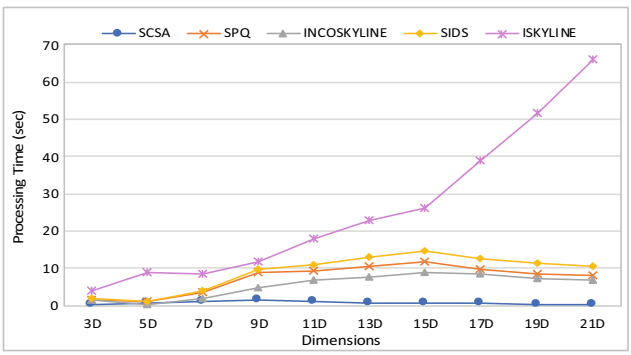

a) Independent




b) Correlated




c) NBA




d) CoIL Insurance Company

**Fig. 21** The effect of number of dimensions on the processing time

filtration techniques helps prune the initial dataset and eliminates many dominated data items from further processing. Furthermore, the optimization process embedded in *SCSA* helps prune many dominated local skylines before applying the skyline process. Therefore, a significant reduction in number of pairwise comparisons to identify the final skylines can

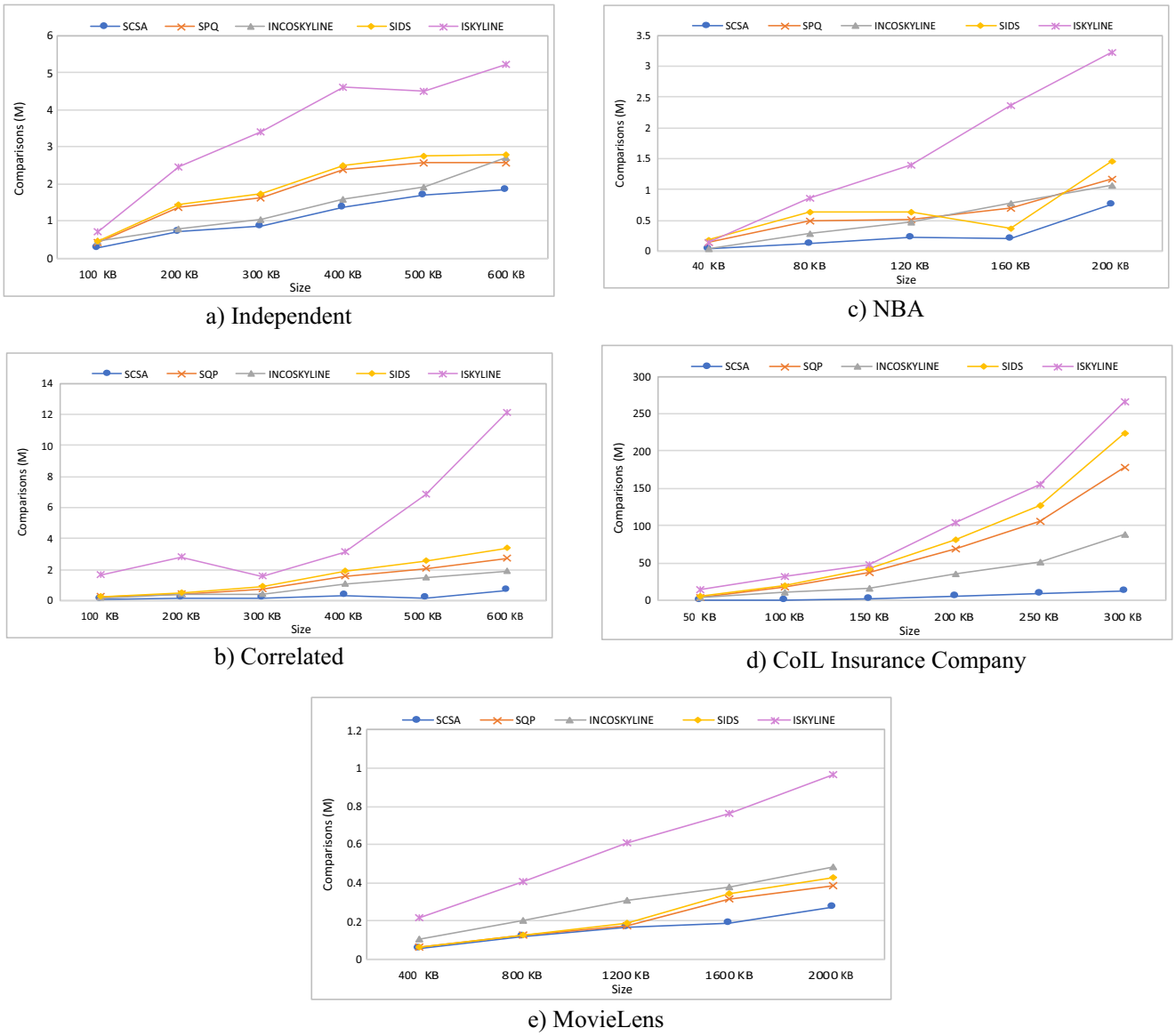


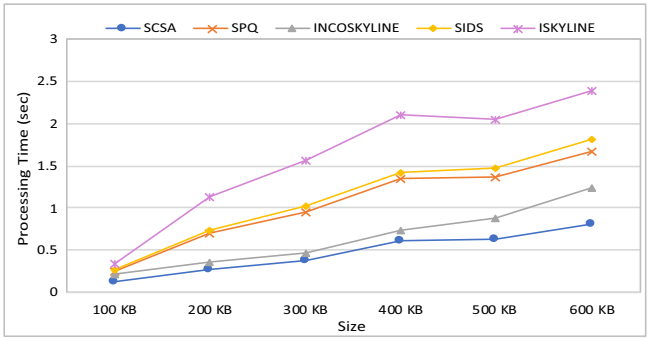

**Fig. 22** The effect of dataset size on the number of pairwise comparisons

be obtained. From Fig. 21 we conclude that our proposed approach (*SCSA*) outperforms all the other approaches designed to process skyline queries in incomplete datasets (*SPQ*, *Incoskyline*, *SIDS*, and *Iskyline*). For instance, while *SCSA* takes only less than two seconds to produce the skylines of a 300 KB dataset with 12 dimensions, *SPQ*, *Incoskyline*, *SIDS* and *Iskyline* require more than 10 s as shown in Fig. 21b).

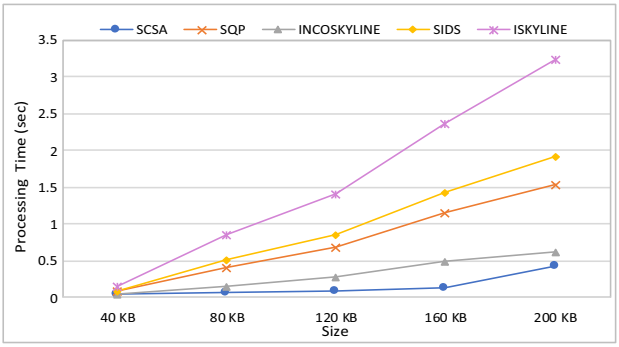
The idea of constructing clusters based on the data items' *domination power* and divining data items of clusters into smaller groups makes each cluster and each group within a cluster independent. Thus, this allows to process clusters and groups while simultaneously identifying the local skylines. This highly efficient technique allows *SCSA* to consume less processing time when identifying the final skylines, which makes *SCSA* efficient and highly effective compared to other approaches (*SPQ*, *Incoskyline*, *SIDS* and *Iskyline*).

### 5.1.2 Effect of dataset size

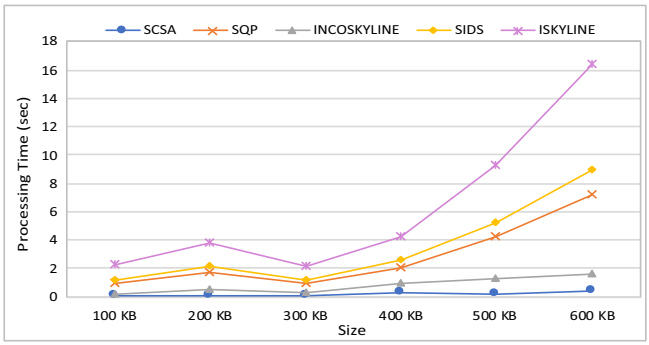
Figure 22a, b, c, d and e explain the results of the number of pairwise comparisons that have been executed on the data items during the skyline process for the independent, correlated, NBA, CoIL 2000 insurance company, and MovieLens datasets. This set of experiments examines the impact of the dataset size on the skyline computation process. For the synthetic dataset (independent and correlated), the number of dimensions is fixed to six while the dataset size varies from 100 KB to 600 KB (Fig. 22a and b). From the result presented in the figure, we conclude that the dataset size has a significant



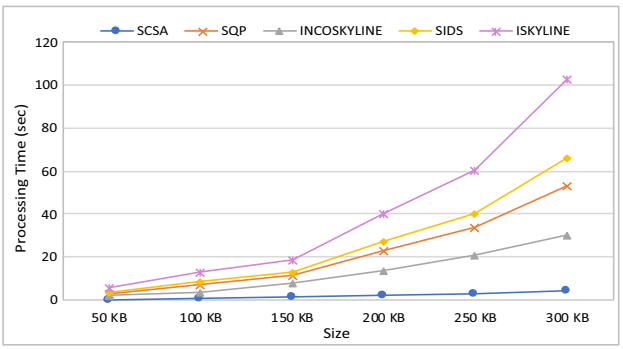

a) Independent



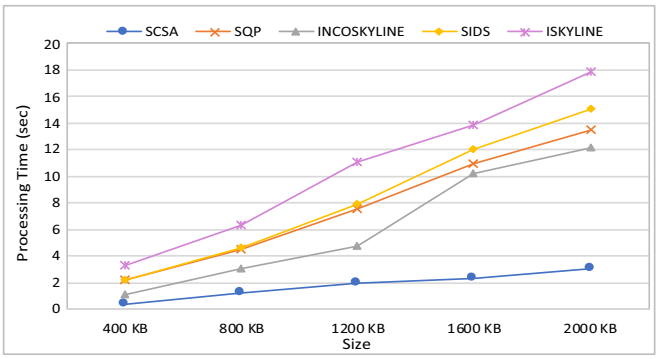

c) NBA




b) Correlated




d) CoIL Insurance Company




e) MovieLens

**Fig. 23** The effect of dataset size on the processing time

impact on the skyline process, as the number of pairwise comparisons is gradually raised when the dataset size is increased. We also notice that *SCSA* outperforms *SPQ*, *Incoskyline*, *SIDS*, and *Iskyline* in all cases since it makes use of the *domination power* (*dp*) for each data item. The *dp* denotes that a data item with low *dp* value is eliminated before applying the skyline technique. Figure 22c demonstrates the number of pairwise comparisons performed in order to identify the skylines for the NBA dataset. In this experiment, the number of dimensions is fixed to 17, while the dataset size varies from 40 to 200 KB. From the figure, we can conclude that our approach consistently outperforms *SPQ*, *Incoskyline*, *SIDS*, and *Iskyline* in all cases and that the performance of *Iskyline* dramatically deteriorates when the dataset size gradually increases. We also notice that *SPQ*, *Incoskyline* and *SIDS* approaches perform slightly worse than our approach in all cases. The main reason behind the slight improvement by SCSA is that the size of the dataset for synthetic datasets, CoIL 2000 insurance company and MovieLens, is far larger than the size of NBA, which significantly influences the performance of *Incoskyline*, *SIDS*, and *Iskyline*. Hence, these approaches generate more local skylines and execute more pairwise comparisons between data items. In contrast, SCSA generates a lower number of local skylines as compared to *Iskyline*, *SIDS*, and *Incoskyline*.

Figure 22d elaborates the results of the experiment on the real dataset, CoIL 2000 insurance company. In this experiment, 13 dimensions have been considered, and the dataset

size varies from 50 to 300 KB. The experiment result shows that the size of the dataset has a marginal impact on the performance of our approach. This is because many unwanted data items are removed during the filtration process performed before applying skyline technique, which helps avoid many unnecessary pairwise comparisons. Figure 22e depicts the experimental result obtained for the MovieLens real dataset. In this experiment, the number of dimensions is four, and the dataset size varies from 400 to 2000 KB. From this figure, we notice that the performance of our approach is marginally better than that of *SPQ* and *SIDS* if the dataset size is less than 1200 KB. However, there is a gap between the performance of our approach, *SQP* and *SIDS,* which gradually increases when the dataset size is larger than 1200 KB.

Figure 23a, b, c, d and e describe the processing time of identifying the skylines on an incomplete database for the independent, correlated, NBA, CoIL 2000 insurance company, and MovieLens datasets. The parameter settings of this set of experiments are the same as that of the previous experiments described in Fig. 22. Figure 23a and b describe the processing time of the skyline query operation in an incomplete database on the synthetic dataset (independent, correlated). From the figure, it is clear that our approach is better than *SPQ*, *Incokyline*, *SIDS*, and *Iskyline* in all cases as it requires less processing time as the *sorting and filtration* and *selecting superior local skylines* phase significantly contributes to removing many dominated data items while identifying the skylines in a database with incomplete data.

Figure 23c demonstrates the experiment result that was conducted on the NBA dataset. We notice that our approach outperforms the previous approaches in all cases and that *SPQ*, *Iskyline* and *SIDS* perform worse by requiring more processing time if the dataset size increases. Our approach performs slightly better than *Incoskyline* if the dataset size is less than 120 KB as it is not easy to find many dominated data items in a small dataset, as compared to a dataset with a large number of data items. This causes unnecessary pairwise comparisons between the data items, which further increases the processing time. This connection can also be observed in the synthetic dataset (independent and correlated).

The processing time result for the skyline operation on the CoIL 2000 insurance company dataset has been demonstrated in Fig. 23d and indicates that our approach steadily outperforms *SPQ*, *Iskyline*, *SIDS*, and *Incoskyline* in all cases. This can be explained by the fact that our approach avoids many unwanted pairwise comparisons among the data items, which in turn reduces the processing time. Lastly, Fig. 23e presents the experimental results obtained from the MovieLens dataset.

We can conclude that *SCSA* requires less processing time as compared to *Iskyline*, *SIDS*, *Incoskyline*, and *SPQ* during the skyline operation for the different dataset sizes. This is due to the fact that our approach successfully excludes many dominated data items from the process of pairwise comparison, which in consequence shortens the processing time considerably. The figure also demonstrates that *Iskyline*, *SIDS*, *Insoskyline* and *SPQ* perform worst if the dataset size keeps increasing, which forces them to scan the entire dataset more than once. Hence, multiple data scans generates a high number of pairwise comparisons to identify the skylines.

The experiment results presented throughout the paper show that our proposed technique outperforms the most recent techniques proposed for processing skyline queries in incomplete databases (*Iskyline*, *Incoskyline*, SIDS and *SPQ*). The experimental results prove the effectiveness and the efficiency of our proposed solution in managing the skyline query process in incomplete databases. Our proposed approach utilized the idea of sorting the data items based on the non-missing values before the skyline process unlike the *Iskyline* and *Incoskyline* techniques that apply the skyline process without first filtering the data items. Prior filtering of the initial data helps avoid unnecessary exhaustive pairwise comparison [37, 39, 43]. Most importantly, the new idea of generating the *domination power* for each data item also eliminates many unnecessary data items from further processing before applying the skyline operation. Moreover, the concept of creating clusters based on the *domination power* values of the data items also significantly contributes to simplifying the skyline process and avoiding unnecessary and unwanted pairwise comparisons. This stands in contrast to the examined *SIDS* and *SPQ* techniques that perform a sequential scan to all sorted data items without considering the value of the *domination power* in order to exclude unnecessary data items. Lastly, the concept of creating smaller groups from clusters based on the *bitmap representation* has greatly improved the efficiency of our proposed approach and minimizes the number of pairwise comparisons and the processing time of the skyline process.

## 6 Conclusion

Skyline queries are generally considered as an expensive process given their extensive domination tests when determining the skylines. The dataset size and the number of dimensions have a critical impact on the searching space and affect the computation process. This paper proposes a novel hybrid algorithm (SCSA) that derives skylines from incomplete data by minimizing the searching space and reducing the domination tests. SCSA processes the skylines by removing the dominated data items before applying skyline technique. The clustering of data items is achieved by implementing the highly efficient technique of generating the *domination power* of each data item. Furthermore, the idea of dividing clusters into smaller groups based on bitmap representation allows to process all clusters and groups within clusters simultaneously. These two preprocessing steps simplify the skyline process

involving incomplete databases. In order to prove the efficiency and the effectiveness of our proposed approach, several experiments have been run on real and synthetic datasets. The results have shown that our algorithm is superior and outperforms the most recent skyline algorithms proposed to process skyline queries in incomplete datasets.

**Acknowledgements** This research is supported by the project FRGS15-205-0491, Ministry of Education, Malaysia.

## References


1. Borzsony S, Kossmann D, Stocker K (2001) The Skyline operator. In: Proceedings 17th International Conference on Data Engineering, Cancun, Mexico, 2001. pp 421–430. doi:https://doi.org/10.1109/ICDE.2001.914855
2. Khalefa ME, Mokbel MF, Levandoski JJ (2008) Skyline Query Processing for Incomplete Data. In: IEEE 24th International Conference on Data Engineering, Cancun, (Mexico). PP. 556–565, 7–12 April 2008 2008. pp 556–565. doi:https://doi.org/10.1109/ICDE.2008.4497464
3. Alwan AA, Ibrahim H, Udzir NI (2014) A Framework for Identifying Skylines over Incomplete Data. In: 3rd International Conference on Advanced Computer Science Applications and Technologies (ACSAT), 2014 2014. IEEE, pp 79–84
4. Gulzar Y, Alwan AA, Salleh N, Shaikhli IFA, Alvi SIM (2016) A Framework for Evaluating Skyline Queries over Incomplete Data. Procedia Computer Science 94:191–198. https://doi.org/10.1016/j.procs.2016.08.030
5. Abidi A, Elmi S, Bach Tobji MA, HadjAli A, Ben Yaghlane B (2018) Skyline queries over possibilistic RDF data. Int J Approx Reason 93:277–289. https://doi.org/10.1016/j.ijar.2017.11.005
6. Elmi S, Benouaret K, Hadjali A, Bach Tobji MA, Ben Yaghlane B (2014) Computing Skyline from Evidential Data. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8720:148–161
7. Elmi S, Hadjali A, Tobji MAB, Yaghlane BB (2016) Imperfect top-k skyline query with confidence level. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), Nov. 29 2016-Dec. 2 2016 2016. pp 1–8. doi:https://doi.org/10.1109/AICCSA.2016.7945620
8. Elmi S, Tobji MAB, Hadjali A, Yaghlane BB (2016) Efficient Skyline Maintenance over Frequently Updated Evidential Databases. Communications in Computer and Information Science 611:199–210. https://doi.org/10.1007/978-3-319-40581-0_17
9. Gulzar Y, Alwan AA, Salleh N, Shaikhli IFA (2017) Processing skyline queries in incomplete database: Issues, challenges and future trends. J Comput Sci 13(11):647–658. https://doi.org/10.3844/jcssp.2017.647.658
10. Gulzar Y, Alwan AA, Salleh N, Al Shaikhli IF (2018) A Model for Skyline Query Processing in a Partially Complete Database. Adv Sci Lett 24(2):1339–1343. https://doi.org/10.1166/asl.2018.10745
11. Tan K-L, Eng P-K, Ooi BC (2001) Efficient progressive skyline computation. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB27), Roma, 2001. pp 301–310
12. Chan C-Y, Jagadish HV, Tan K-L, Tung AKH, Zhang Z (2006) On High Dimensional Skylines. In: Advances in Database Technology - EDBT 2006: 10th International Conference on Extending Database Technology, Munich, Germany, March 26–31, 2006. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 478–495. doi:10.1007/11687238_30
13. Chan C-Y, Jagadish HV, Tan K-L, Tung AKH, Zhang Z (2006) Finding k-dominant skylines in high dimensional space. Paper presented at the Proceedings of the 2006 ACM SIGMOD international conference on Management of data, Chicago
14. Mouratidis K, Bakiras S, Papadias D (2006) Continuous monitoring of top-k queries over sliding windows. Paper presented at the Proceedings of the 2006 ACM SIGMOD international conference on Management of data, Chicago
15. Yiu ML, Mamoulis N (2007) Efficient processing of top-*k* dominating queries on multi-dimensional data. Paper presented at the Proceedings of the 33rd international conference on Very large data bases, Vienna
16. Morse M, Patel JM, Grosky WI (2007) Efficient continuous skyline computation. Inf Sci 177(17):3411–3437. https://doi.org/10.1016/j.ins.2007.02.033
17. Kalyvas C, Tzouramanis T, Manolopoulos Y (2017) Processing skyline queries in temporal databases. Paper presented at the Proceedings of the Symposium on Applied Computing, Marrakech
18. Lofi C, El Maarry K, Balke W-T (2013) Skyline Queries over Incomplete Data - Error Models for Focused Crowd-Sourcing. In: Ng W, Storey VC, Trujillo JC (eds) Conceptual Modeling: 32th International Conference, ER 2013, Hong-Kong, China, November 11–13, 2013. Proceedings. Springer Berlin Heidelberg, Berlin, pp 298–312. doi:https://doi.org/10.1007/978-3-642-41924-9_25
19. Lee J, Lee D, Kim S-W (2016) CrowdSky: Skyline Computation with Crowdsourcing. In: EDBT, 2016. pp 125–136
20. Swidan MB, Alwan AA, Turaev S, Gulzar Y (2018) A Model for Processing Skyline Queries in Crowd-sourced Databases. Indonesian Journal of Electrical Engineering and Computer Science 10(2):798–806. https://doi.org/10.11591/ijeecs.v10.i2.pp798-806
21. Gulzar Y, Alwan AA, Salleh N, Shaikhli IFA (2017) Skyline Query Processing for Incomplete Data in Cloud Environment. In: Proceedings of the 6th International Conference on Computing & Informatics, Kuala Lumpur, Malaysia, 2017. J. & N. H. Zakaria (Eds.), pp 567–576. http://icoci.cms.net.my/PROCEEDINGS/2017/Pdf_Version_Chap12e/PID91-567-576e.pdf. 10 August 2017
22. Kossmann D, Ramsak F, Rost S (2002) Shooting stars in the sky: an online algorithm for skyline queries. Paper presented at the Proceedings of the 28th international conference on Very Large Data Bases, Hong Kong
23. Chomicki J, Godfrey P, Gryz J, Liang D (2003) Skyline with presorting. In: Proceedings 19th International Conference on Data Engineering (ICDE03), Bangalore (India), 5–8 March 2003 2003. pp 717–719. doi:https://doi.org/10.1109/ICDE.2003.1260846
24. Papadias D, Tao Y, Fu G, Seeger B (2003) An optimal and progressive algorithm for skyline queries. Paper presented at the Proceedings of the 2003 ACM SIGMOD international conference on Management of data, San Diego
25. Kung HT, Luccio F, Preparata FP (1975) On Finding the Maxima of a Set of Vectors. J ACM 22(4):469–476. https://doi.org/10.1145/321906.321910
26. Bentley JL, Kung HT, Schkolnick M, Thompson CD (1978) On the Average Number of Maxima in a Set of Vectors and Applications. J ACM 25(4):536–543. https://doi.org/10.1145/322092.322095
27. Bentley JL, Clarkson KL, Levine DB (1993) Fast linear expected-time algorithms for computing maxima and convex hulls. Algorithmica 9(2):168–183. https://doi.org/10.1007/BF01188711
28. Tomoiagă B, Chindriş M, Sumper A, Sudria-Andreu A, Villafafila-Robles R (2013) Pareto Optimal Reconfiguration of Power Distribution Systems Using a Genetic Algorithm Based on NSGA-II. Energies 6(3):1439
29. Rodger JA, Pankaj P, Nahouraii A (2014) A Petri Net Pareto ISO 31000 Workflow Process Decision Making Approach for Supply

Chain Risk Trigger Inventory Decisions in Government Organizations. Intell Inf Manag 6(03):157
30. Godfrey P, Shipley R, Gryz J (2005) Maximal vector computation in large data sets. Paper presented at the Proceedings of the 31st international conference on Very large data bases, Trondheim
31. Bartolini I, Ciaccia P, Patella M (2006) SaLSa: computing the skyline without scanning the whole sky. Paper presented at the Proceedings of the 15th ACM international conference on Information and knowledge management, Arlington
32. Zhang S, Mamoulis N, Cheung DW (2009) Scalable skyline computation using object-based space partitioning. Paper presented at the Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, Providence
33. Lee KC, Lee W-C, Zheng B, Li H, Tian Y (2010) Z-SKY: an efficient skyline query processing framework based on Z-order. VLDB J 19(3):333–362. https://doi.org/10.1007/s00778-009-0166-x
34. Lee J, S-w H (2014) Scalable skyline computation using a balanced pivot selection technique. Inf Syst 39(Supplement C):1–21. https://doi.org/10.1016/j.is.2013.05.005
35. Arefin MS, Morimoto Y (2012) Skyline sets queries for incomplete data. International Journal of Computer Science & Information Technology 4(5):67–80
36. Miao X, Gao Y, Chen L, Chen G, Li Q, Jiang T (2013) On Efficient k-Skyband Query Processing over Incomplete Data. In: Meng W, Feng L, Bressan S, Winiwarter W, Song W (eds) 18th International Conference on Database Systems for Advanced Applications, Wuhan, Chian, 2013. pp 424–439. doi:10.1007/978-3-642-37487-6_32
37. Bharuka R, Kumar PS (2013) Finding skylines for incomplete data. Paper presented at the Proceedings of the 24th Australasian Database Conference - Volume 137, Adelaide
38. Balke W-T, Güntzer U, Zheng JX (2004) Efficient Distributed Skylining for Web Information Systems. In, Berlin, Heidelberg, 2004. Advances in Database Technology - EDBT 2004. Springer Berlin Heidelberg, pp 256–273
39. Bharuka R, Kumar PS (2013) Finding superior skyline points from incomplete data. Paper presented at the Proceedings of the 19th International Conference on Management of Data, Ahmedabad
40. Zhang K, Gao H, Wang H, Li J (2016) ISSA: Efficient Skyline Computation for Incomplete Data. In: Gao H, Kim J, Sakurai Y (eds) Database Systems for Advanced Applications: DASFAA 2016 International Workshops: BDMS, BDQM, MoI, and SeCoP, Dallas, TX, USA, April 16–19, 2016, Proceedings. Springer International Publishing, Cham, pp 321–328. doi:10.1007/978-3-319-32055-7_26
41. Lee J, Im H, G-w Y (2016) Optimizing Skyline Queries over Incomplete Data. Inf Sci 361:14–28. https://doi.org/10.1016/j.ins.2016.04.048
42. Alwan AA, Ibrahim H, Udzir NI, Sidi F (2016) An Efficient Approach for Processing Skyline Queries in Incomplete Multidimensional Database. Arab J Sci Eng 41(8):2927–2943. https://doi.org/10.1007/s13369-016-2048-z
43. Wang Y, Shi Z, Wang J, Sun L, Song B (2017) Skyline Preference Query Based on Massive and Incomplete Dataset. IEEE Access 5:3183–3192. https://doi.org/10.1109/ACCESS.2016.2639558
44. Fotiadou K, Pitoura E (2008) BITPEER: continuous subspace skyline computation with distributed bitmap indexes. Paper presented at the Proceedings of the 2008 international workshop on Data management in peer-to-peer systems, Nantes
45. Wong RC-W, Fu AW-C, Pei J, Ho YS, Wong T, Liu Y (2008) Efficient skyline querying with variable user preferences on nominal attributes. Proc VLDB Endow 1(1):1032–1043. https://doi.org/10.14778/1453856.1453967
46. Soliman MA, Ilyas IF, Ben-David S (2010) Supporting ranking queries on uncertain and incomplete data. VLDB J 19(4):477–501. https://doi.org/10.1007/s00778-009-0176-8