IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 13, 2021, accepted March 30, 2021, date of publication April 12, 2021, date of current version April 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3072775

IDSA: An Efficient Algorithm for Skyline Queries Computation on Dynamic and Incomplete Data With Changing States

YONIS GULZAR[1], ALI A. ALWAN[2], HAMIDAH IBRAHIM[3], (Member, IEEE), SHERZOD TURAEV[4], SHARYAR WANI[2], ARJUMAND BANO SOOMRO[1,6], AND YASIR HAMID[5]

[1]Department of Management Information Systems, College of Business Administration, King Faisal University, Al-Ahsa 31982, Saudi Arabia
[2]Department of Computer Science, Faculty of Information and Communication Technology, International Islamic University Malaysia, Kuala Lumpur 53100, Malaysia
[3]Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang 43400, Malaysia
[4]Department of Computer Science and Software Engineering, United Arab Emirates University, Al Ain, United Arab Emirates
[5]Department of Information Security and Engineering Technology, Abu Dhabi Polytechnic College, Abu Dhabi 111499, United Arab Emirates
[6]Department of Software Engineering, Faculty of Engineering and Technology, University of Sindh, Sindh 76060 Pakistan

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

This work was supported by the Deanship of Scientific Research, King Faisal University, Saudi Arabia, through the Research Grant Program Nasher, under Grant 186331.

ABSTRACT Skyline queries have been widely used as an effective query tool in many contemporary database applications. The main concept of skyline queries relies on retrieving the non-dominated tuples in the database which are known skylines. In most database applications, the contents of the databases are dynamic due to the continuous changes made towards the database. Typically, the changes in the contents of the database occur through data manipulation operations (INSERT and/or UPDATE). Performing these operations on the database results in invalidating the most recent skylines before changes are made on the database. Furthermore, the presence of incomplete data in databases becomes frequent phenomena in recent database applications. Data incompleteness causes several challenges on skyline queries such as losing the *transitivity property* of the skyline technique and the test dominance process between tuples being *cyclic*. Reapplying skyline technique on the entire updated incomplete database to determine the new skylines is unwise due to the exhaustive pairwise comparisons. Thus, this paper proposes an approach, named Incomplete Dynamic Skyline Algorithm (IDSA) which attempts to determine the skylines on dynamic and incomplete databases. Two optimization techniques have been incorporated in IDSA, namely: pruning and selecting superior local skylines. The pruning process attempts to exploit the derived skylines before the INSERT/UPDATE operation made on the database to identify the new skylines. Moreover, selecting superior local skylines process assists in further eliminating the remaining non-skylines from further processing. These two optimization techniques lead to a large reduction in the number of domination tests due to avoiding re-computing of skylines over the entire updated database to derive the new skylines. Extensive experiments have been accomplished on both real and synthetic datasets, and the results demonstrate that IDSA outperforms the existing solutions in terms of the number of domination tests and the processing time of the skyline operation.

INDEX TERMS Dynamic database, incomplete database, pairwise comparison, skyline queries.

I. INTRODUCTION

Traditional queries operate in a very non-flexible manner as they either return data from a database that strictly satisfies the conditions given in the submitted query or return no result if otherwise. Thus, this rigid mechanism in processing traditional queries sometimes is impractical and might be undesirable for many modern database systems. A huge effort has been made aiming at developing more flexible query operators that attempt to relax these stringent requirements to retrieve the most preferred tuples from a database based on the preferences given in the submitted query, also known

The associate editor coordinating the review of this manuscript and approving it for publication was Arif Ur Rahman.

as user-defined preferences. These preference queries employ preference evaluation techniques which have achieved significant success, as they are widely used in a non-trivial number of applications related to multi-criteria decision support systems. For almost two decades, the area of skyline queries, which is one of the most prominent types of preference queries in database systems has received formidable attention and become a subject of intensive research by a large number of researchers in the database community. This intensive focus on skyline queries reflects the fact that skyline queries have tremendous benefits and have been applied on various domains including but not limited to web-based business [1], multi-criteria decision-making system [2]–[4], crowd-sourcing databases [5]–[9], temporal databases [10], cloud databases [11], [12] and decision support system [2], [3], [13].

Since the first introduction of skyline queries operator to database systems in 2001 by Borzsony *et al.* [14] most of the previous works are either focusing on optimizing the process of skyline queries computation assuming that the database is complete and all values of tuples are present [1]–[3], [14]–[18]. Among the most remarkable variation of skyline technique designed for a database with complete data are Divide-and-Conquer (D&C), Block Nested- Loop (BNL) [14], Bitmap and Index [15], Sort Filter Skyline (SFS) [16], Branch and Bound Skyline (BBS) [19], Linear Elimination Sort Skyline (LESS) [17], Sort and Limit Skyline algorithm (SaLSa) [18], Nearest Neighbor (NN) [20], ZSearch [21], and OSPS [22].Thus, the assumption of data completeness assures that all tuples are comparable against each other, and performing the pairwise comparisons is straightforward and results in identifying the skyline results.

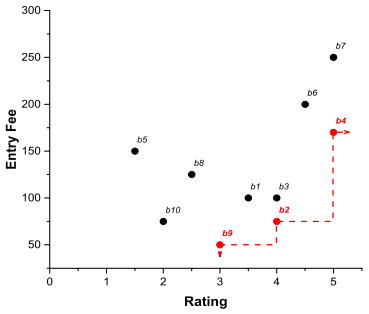
Another group of researchers made the effort to further investigate the challenges when processing skyline queries on a database with incomplete data [23]–[31] in which some values are not present during the skyline process. The focus was given to resolve the issue of *losing the transitivity* property of the skyline technique and avoid the problem of *cyclic dominance* because many tuples are incomparable against each other. The following running database example explains the problem of losing the *transitivity property* of the skyline technique and the *cyclic dominance* problem associated with determining the skylines on a database with incomplete data. Assume a bar database that consists of three dimensions (*rating*, *entry free, distance*) and contains three tuples. We also assumed that the values of certain dimensions of the tuples are missing (not present). The details of the three tuples are as follows. b_1(*, 150, 5), b_2(5, *, 7), and b_3(4, 140, *). The symbol (*) indicates the missing value in a particular dimension. Assume a person is looking for a bar with the following preferences, the nearest in *distance* from his current location, the highest in *rating,* and the cheapest in the *entry fee*. Applying the skyline technique on the bar database should return those tuples (skylines) which are not dominated by any other tuples in the database. A detailed analysis of the skyline process by performing the pairwise comparison process between these three tuples with incomplete data made the following conclusion. We noticed that bar b_1 dominates b_2 in the 3rd non-missing dimension only where the distance of b_1 is *less* than bar b_2. The remaining two dimensions (1st and 2nd) are incomparable since the value of d_1 for bar b_1 and the value of d_2 for bar b_2 are both not present (missing). While comparing bar b_2 with b_3 produces that b_2 dominates b_3 in the common non-missing dimension d_1 where b_2 is having a better rating than b_3. Nonetheless, the remaining two dimensions d_2 and d_3 are incomparable due to the missing values in one of these dimensions. Based on the concept of *transitivity property* of skyline, if b_1 dominates b_2 and b_2 dominates b_3 then as a consequence b_1 should also dominate b_3. However, the results of comparisons demonstrate that b_1 is not better than b_3 in any comparable dimensions. Hence, it can be concluded that the *transitivity property* of the skyline technique is no longer held when the database has missing values in one or more dimensions. Most importantly, the impact of data incompleteness is not only violating the *transitivity property* of the skyline, but it also leads to making the *dominance test* process among the tuples to be *cyclic*. From the above example, it is clear that the value of the *entry fee* dimension for b_3 is 140 while the value of the *entry fee* dimension for b_1 is 150. The comparison process indicates that b_3 dominates b_1 based on the common non-missing dimensions (*entry fee*), while *distance* and *rating* dimensions have been discarded as both dimensions are incomparable. Eventually, the *dominance test* process ends up with a result that depicts b_1 dominates b_2, b_2 dominates b_3, and b_3 dominates b_1 and retrieves an empty set of skylines.

Most importantly, we also observed that most of the previous works assumed that the contents of the database are *static* and update operations such as insert are very infrequent [24], [26], [27], [29], [30], [32]–[35]. However, the update operation is among the indispensable operations in database systems. Updating the database includes inserting new tuples into the database and removing some existing tuples from the database. Updates also encompass changing the values of some existing dimensions that belong to the tuples in the database. Update operations introduce a significant challenge in processing skylines queries as skyline results can be highly influenced by these updates. This effect can be seen in the skyline result as well as the skyline computation process. When an update occurs in a database by inserting new tuples a re-evaluation for skyline result must be achieved. This is due to the fact that some of the inserted tuples might dominate the existing skylines and should be included in the new skylines result. Re-evaluation of skylines on the entire updated database is unwise due to the prohibitive cost and the unnecessary exhaustive pairwise comparisons.

Referring to the bar database example with a different scenario to demonstrate the issue of dynamic contents of the database. Consider a case where a person is looking for a bar(s) preferring bars that have the minimum *entry fee* while the *ratings* are the highest. For simplicity and without losing the generality, the bar database example has

Bar ID	Rating	Entry Fee
b_1	3.5	100
b_2	4	75
b_3	4	100
b_4	5	170
b_5	1.5	150
b_6	4.5	200
b_7	5	250
b_8	2.5	125
b_9	3	50
b_{10}	2	75

a)

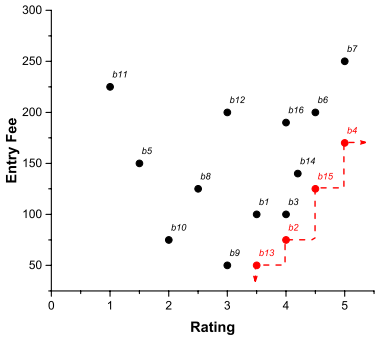


b)

FIGURE 1. Example of bar dataset before insert operation.

Bar ID	Rating	Entry Fee
b_1	3.5	100
b_2	4	75
b_3	4	100
b_4	5	170
b_5	1.5	150
b_6	4.5	200
b_7	5	250
b_8	2.5	125
b_9	3	50
b_{10}	2	75
b_{11}	1	225
b_{12}	3	200
b_{13}	3.5	50
b_{14}	4.2	140
b_{15}	4.5	125
b_{16}	4	190

a)



b)

FIGURE 2. Example of bar dataset after insert operation.

been modified by considering the *entry fee* and the *rating* dimensions only. However, the *distance* dimension has been discarded. According to the syntax rules of skyline operators, the SQL like syntax for a skyline query should be written as follows.

SELECT * FROM Bar
Where Beach= 'Hyams' AND City= 'Shoalhaven'
SKYLINE OF Rating MAX, Entry_Fee MIN;

The query engine involves a skyline operator and based on the person's given preferences should retrieve only those tuples which have the highest rating and the minimum entry fee from the bar database. **Figure 1a** represents the data of the bar database in the relational (tabular) form. The relation contains the details of 10 bars. while **Figure 1b** shows the graphical representation of the bar database in 2*D* space. According to the definition of the skylines technique, it can be learned that b_5, b_8, and b_{10} are being fully dominated by b_9, as b_9 is better than b_5, b_8, and b_{10} in both ratings (max) as well as in entry fee (min) dimensions. It can be also noticed that b_1 is fully dominated by b_2 whereas b_3 partially dominates b_1 based on entry fee dimension only. Moreover, it is also obvious that b_2 also dominates b_3 and b_6 and b_7 are dominated by b_4. From **Figure 1**, we can also notice that the entry fee of b_4 is cheaper than bar b_6 and b_7 and the rating of b_4 is higher than b_6 whereas the rating of b_4 is not worse than b_7. Finally, the skyline set encompasses bars b_2, b_4, and b_9 which are retrieved to the end-user.

Assume a new set of tuples have been inserted into the bar database through insert operation. The inserted tuples are b_{11}, b_{12}, b_{13}, b_{14}, b_{15}, b_{16} as depicted in **Figure 2a**. Inserting new tuples into the database through insert operation would make the skyline result before the insert to be no longer valid and a re-evaluation for skyline queries should be conducted. Based on the most recent information in the bar database, it can be noticed that the bar b_9 which has been reported as skyline before the insert operation has been dominated by the newly inserted bar b_{13} based on the rating dimension. This indicates that b_9 is no longer a valid skyline and should be removed from the skyline result. Similarly, the newly added bar b_{15} dominates other bars and it is not worse than any other bars in the entire database and should be reported in the new result of the skylines. We can observe that the other newly inserted tuples b_{11}, b_{12}, b_{14}, and b_{16} are dominated and should be discarded. Finally, the new skyline result after the changes made towards the initial database comprises b_{13}, b_2, b_{15}, and b_4.

The above example demonstrates that it is necessary to re-evaluate the skyline result every time a new tuple is inserted into the database to make sure that the skyline result reflects the most recent information in the database.

In the era of Internet-of-Things (IoT) data is enormous and collected from different sensors and monitors from remote locations. While transmission of data from these devices, there are high chances that certain tuples have missing values in one or more dimensions making the database to be incomplete. The frequent insert of new tuples from these devices makes the contents of the database dynamic. These insert operations performed towards the initial incomplete database made the skylines derived before the insert operation to be no longer valid in the new state of the database. It is unwise and impractical to directly apply the skyline technique on the entire database after changes are made to compute the new skylines. This is due to the fact that not all tuples are affected by the performed insert operation. Hence, the data incompleteness and dynamism nature of data make the process of identifying skylines a non-trivial task. Thus, an efficient skyline method aimed at avoiding re-computing skylines on the entire updated database is needed. The approach should take into consideration reducing the overhead of the skyline operation while maintaining skyline results based on the latest information in the database.

In this paper, we take the challenge to solve the problems involved to compute the skylines over an incomplete and dynamic database. This includes the issue of recomputing skylines in an incomplete and dynamic database by avoiding scanning the entire database after the update is made towards the initial incomplete database. Furthermore, this paper also emphasizes resolving the issue of sustaining the *transitivity property* of the skyline technique and preventing the cyclic dominance problem associated with the domination test process in an incomplete database. The idea of the proposed strategy concentrates on minimizing the number of pairwise

comparisons during the domination tests process which in turn reduces the processing time.

The main contributions of this paper are briefly explained as follows:

- The problem of processing skyline queries in an incomplete and dynamic database where values of certain dimensions of tuples are missing and the contents of the database are frequently updated through data manipulation operations (insert and update) has been highlighted.
- A detailed and comprehensive review of the most remarkable related works in the area of skyline queries has been carried out. The review encompasses examining the proposed solutions designed to work on both complete and incomplete databases. The review outlines the details of the strengths and weaknesses of each approach.
- An efficient algorithm named IDSA that is capable of answering skyline queries over an incomplete and dynamic database is proposed. IDSA comprises seven phases working towards determining the skylines of an incomplete and dynamic database.
- An efficient data filtration technique called *pruning* has been employed to eliminate the dominated newly inserted tuples before applying the skyline process. The idea of data *pruning* is very beneficial and results in a significant reduction in the number of pairwise comparisons performed to identify the skylines after the changes are made towards the initial incomplete database.
- An innovative data partitioning method has been proposed which divides the inserted tuples of the database into distinct clusters based on their domination power. The idea of data partitioning is crucial as it assists in minimizing the number of pairwise comparisons performed to determine the skylines after the changes are made towards the initial incomplete database.
- The efficiency of the proposed solution has been evaluated with numerous experiments conducted on both real and synthetic datasets. The experiment results prove the capability of IDSA in generating the skylines after changes are made towards a database.

The rest of the paper is organized as follows. In Section II, the previous works related to this research are presented and discussed deliberately. The basic definitions and the necessary notations related to skyline queries over dynamic and incomplete databases, which are used in the rest of the paper, are set out in Section III. The proposed algorithm for processing skyline queries on dynamic and incomplete data with changing states is illustrated in Section IV. The experimental results are presented and explained in Section V. The conclusion is described in the final section, Section VI.

II. RELATED WORK

Many variations of skyline techniques have been introduced in the past two decades. The first two skyline techniques introduced in the database systems are Block-Nested-Loop (BNL) and Divide and Conquer (DC) which are proposed by the work in [14]. The aim of these two approaches (BNL and DC) was to determine the non-dominated tuples which are also known as skylines of the database. The BNL approach operates by employing a sorting technique to assist in avoiding scanning the entire database when deriving the skylines. While D&C operates by exploiting the concept of dividing the initial large database into manageable small sub-databases. Then, it computes the skylines of these sub-databases separately and eventually identifies the final skylines by further comparing the skylines of these sub-databases against each other. Generally speaking, almost all the variations of skyline techniques proposed in the literature can be classified into two groups, namely: BNL-based group and DC-based group. Among the remarkable BNL-based approaches are SFS [16], LESS [17], and SaLSa [18], while index [15], NN [20], BBS [19], OSPS [22], ZSearch [21], and BSkyTree [36] are DC-based approaches. These techniques concentrate on improving the efficiency and the performance of the skyline process by reducing the exhaustive pairwise comparison between tuples and minimizing the searching space. Considering the literature, it can be concluded that the searching space constitutes the main concern of the skyline process and is considered as the most important factor that affects its performance. However, these techniques are proposed with the main aim to handle issues related to optimizing skyline computations over a database with complete data. This includes the issue of reducing the searching space, minimizing the pairwise comparisons between tuples to return the skylines, and reducing the total execution time of the skyline process.

In the recent past, the focus has been oriented towards resolving issues related to skyline queries in the presence of incomplete data. A variety of skyline approaches designed for an incomplete database have been established. In the following, we present and discuss the previous approaches proposed to process skyline queries in the incomplete database.

The early attempt on the discussion of the issue of processing skyline queries over incomplete data has been introduced in [24]. Two algorithms have been proposed tackling the issue of processing skyline queries in incomplete data, namely *Bucke*t and *Iskyline*. The idea of the *Bucket* algorithm relies on exploiting the bitmap representation to divide the tuples of the database into distinct buckets. Each bucket contains tuples with exact similar bitmap representations which denote that these tuples have missing values in the similar attributes. Then, the conventional skyline algorithm is utilized on each bucket separately to identify the skylines of each bucket which are also known as local skylines. Finally, the local skylines of each bucket are further compared to each other to identify the skylines of the entire database. The idea of creating distinct buckets has been further improved in the *Iskyline* algorithm with two optimization techniques, *virtual points,* and *shadow skylines*, to reduce the number

of local skylines in every bucket. Reducing the number of local skylines is very beneficial in decreasing the number of pairwise comparisons to be performed in identifying the skylines. However, the idea of the *Iskyline* technique is rather time-consuming as in each bucket many pairwise comparisons need to be performed to find the local skylines. This is because *Iskyline* involves a large 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. In 2012 Arefin and Morimoto [37] proposed an approach named *Replacement Based Sets Skyline Queries* (RBSSQ) for processing skyline queries on a database with missing values. The idea of RBSSQ relies on the concept of the *Bucket* algorithm proposed in [24] to replace the missing values with certain numbers that are larger than the domain value to prevent losing the *transitivity property* of the skyline and to avoid the issue of *cyclic dominance*. The RBSSQ consists of two phases, (i) data pre-processing and (ii) skyline sets computation. In the data pre-processing phase the missing values of the tuples are replaced with values beyond the domain of each attribute. While in the skyline sets computation phase, skylines are computed from the processed data.

In 2013 Bharuka and Kumar [26], introduced a *Sort-based Incomplete Data Skyline* algorithm (SIDS) to compute the skylines on an incomplete database by exploiting the concept of sorting proposed in [38]. The process of SIDS starts by selecting attributes of the tuples in a round-robin fashion and the tuple with the next best value in that attribute being chosen for processing. This step is important to eliminate the dominated tuples in the early stage to avoid the exhaustive pairwise comparisons in determining the skylines. SIDS assumes that all tuples in the database are candidate skylines before removing dominated tuples from the candidate set. Then, if a tuple has not been pruned yet and has been processed k times, where k is the count of complete attributes for the tuple, it is determined to be a skyline and can be returned immediately. The rationale here is that any tuple with k complete attributes can be dominated in at most k attributes. Thus, SIDS can progressively return skylines whenever the above condition is satisfied. However, SIDS assumes that the lists have to be accessed in sequential order, and the system has to receive the results of all lists before moving to the next phase. Thus, increasing the number of lists may degrade the performance of the skyline process and delay generating the skylines for the end-user. Moreover, SIDS lacks any optimization that could simplify the process of identifying the skylines, particularly for a database with dynamic contents. This sequential access renders the process of pairwise comparisons to be exhaustive as many unnecessary pairwise comparisons are needed to eliminate the dominated tuples.

Bharuka and Kumar [27] have further investigated the issue of handling skyline query computation in an incomplete database. They have proposed a hybrid solution named *Incomplete Data Frequent Skyline* (IDFS) based on the concept of the *top-k* frequent skyline technique suggested in [3] to identify the skylines of the incomplete database. The IDFS approach resolves the issue of controlling the size of the skylines utilizing the concept of top-k frequent skyline technique in [3] returning the superior skylines from the database with missing values ordered by their fractional skyline frequency. However, IDFS may not be suitable to be applied on a database with incomplete data and dynamic contents as it has to be reapplied whenever the contents of the database are changed due to the changes made through the data manipulation operations such as insert and update.

Miao *et al.* [39] introduced two efficient algorithms for skyline query processing on incomplete data, in which tuples might have missing values in one or more attributes. These algorithms are *k-SkyBand* algorithms (*kISB*) and *Virtual Point-based* algorithm (VP), which include novel concepts such as the expired skyline, shadow skyline, and thickness warehouse that boost the search performance. The *kISB* algorithm is designed for *kSB* query (*k-Skyband query*) on incomplete data, which employs the concepts of thickness warehouse and expired skyline to improve the search performance. On the other hand, the VP algorithm utilizes the concept of virtual point, expired skyline, and shadow skylines to reduce the size of the candidate set and boost the query performance. The concept of both algorithms inspired by the work in [24].

Alwan *et al.* [40] introduced an algorithm named *Incoskyline* to compute the skylines of an incomplete database taking into consideration the issue of losing the *transitivity property* of the skyline technique and the problem of *cyclic dominance* in deriving skylines. *Incoskyline* comprises of four phases, clustering data, grouping and identifying local skylines, generating k-dom skylines, and identifying incomplete skylines. The *Incoskyline* processes the skyline queries with missing values in an intuitive way by exploiting the idea of generating virtual tuples out of the local skylines of the clusters which are called k-dom skylines. The derived k-dom skylines are combined to produce one global k-dom skyline to be inserted on top of each cluster to prevent the dominated tuples from further processing. A list of candidate skylines is established containing the local skylines of clusters that are not worse than the global k-dom. Eventually, these candidate skylines are further processed by comparing them against each other to return the final skylines of the entire database. However, *Incoskyline* is inadequate to be applied directly on a dynamic database with incomplete data. This is because *Incoskyline* is designed to work on a database with static content and update operations such as insert, and update are very infrequent to be performed on the initial incomplete database. Nevertheless, if any update has been made towards the initial incomplete database, *Incoskyline* has to be reapplied on the entire updated database. This means re-scanning the entire database and unnecessary exhaustive pairwise comparisons have to be performed to specify the new skylines reflecting the most recent information of the database.

Besides, the work in [32] has also highlighted the issue of processing skyline queries in a database with missing data. They have designed a framework called COBO that employs a skyline approach named ISSA to compute the skylines in two stages, namely: *pruning compared lists* and *reducing expected comparison times*. The *pruning compared list* has exploited the idea of the *Bucket* algorithm proposed in [24] to prune the unwanted tuples from each bucket. While *reducing expected comparison times* stage is responsible to find the total sum of complete attributes of all non-dominated tuples from each bucket and sorts those tuples in an ascending order (smaller value considered as best). This technique minimizes the number of pairwise comparisons and the total processing time of the skyline queries operation.

Wang *et al.* [29] have also discussed the issue of processing skyline queries over massive incomplete databases. They have proposed an approach named, *Skyline Preference Query* SPQ, which attempts to derive the skylines of the database through three main steps. The first step attempts to divide the initial massive incomplete database into two separated subsets based on the priority level of the attribute. The skylines of the first subset, which is also known as local skylines, are retrieved using the idea of the SIDS proposed in [26]. In the second step, the concept of bitmap representation has been adopted along with the DC strategy proposed in [14] to return the skylines of the other subsets. Lastly, the final list of skylines of the entire massive incomplete database is formed by comparing the local skylines of both subsets against each other. SQP needs to construct a large number of distinct arrays in which each array is being processed sequentially to specify the skylines. This large number of arrays and the exhaustive pairwise comparisons that need to be performed to determine the skylines incur a longer processing time. This is because many unwanted pairwise comparisons are performed during the process of identifying the local skylines of each subset.

The work presented in [41] has tackled the issue of skyline queries in probabilistic incomplete databases. In probabilistic databases, the values of an attribute are not completely missing, but they are present in a pre-defined range. An algorithm named EP has been designed based on the idea of multi-level grouping. The proposed strategy adopts the concepts of bitmap representation to divide the initial database into different buckets, then the tuples in each bucket are sorted in descending order. Moreover, the EP algorithm applied several pruning strategies aiming at reducing the computational costs of the skyline process. Lastly, the final skylines of the entire database are retrieved by comparing the local skylines of the buckets against each other.

Furthermore, Gulzar *et al.* [42] proposed an efficient algorithm called the *Sorting-based Cluster Skyline Algorithm* (SCSA) processing skyline queries in an incomplete database. SCSA employs the idea of bitmap representation introduced in [40], [24], to distribute the tuples with missing values into various distinct clusters. The clusters are further divided into smaller manageable groups to avoid many unwanted pairwise comparisons between the tuples to generate the skylines. Several optimization techniques have been incorporated in the SCSA algorithm to assist in eliminating many dominated tuples before retrieving the final skylines. Removing those dominated tuples before applying the skyline technique is very beneficial as it leads to reducing the cost of the skyline operation.

To the best of our knowledge, the most recent work that has tackled the issue of processing skyline queries on incomplete databases has been contributed by [43], which proposed three methods for probabilistic skyline computation on incomplete data. The first method called *Sorting-based Probabilistic Skyline* on incomplete data complying with Independent distribution (SPISkyline). The second method is called *Sorting-based Probabilistic Skyline* on incomplete data complying with Correlated distribution (SPCSkyline) and the third method is named *Sorting-based Probabilistic Skyline* on incomplete data complying with Anti-correlated distribution (SPASkyline). Both SPISkyline and SPCSkyline methods employed pruning strategy, optimization of the process of probability computation, and sorting technique to improve the efficiency of probabilistic skyline computation. While the SPASkyline applying optimization of the process of probability computation, and sorting technique to improve the efficiency of probabilistic skyline computation. While the SPASkyline applying optimization of the process of probability computation and sorting technique to improve the efficiency of probabilistic skyline computation. The main reason for not applying the pruning technique in the SPASkyline method is that it is ineffective for incomplete data over anti-correlated databases.

From the work reviewed in this section, we can conclude that most of the previous works have assumed that the skyline queries process is performed on an incomplete database with static contents. However, data manipulation operations such as insert, and update operations are indispensable in most modern database applications. Thus, these update operations most likely result in invalidating the skylines produced before the update is made towards the initial incomplete database. It is unwise to blindly re-apply a cost-prohibitive skyline technique on the entire updated database aiming at generating the new skylines. This is due to the fact that many of the newly inserted tuples might be dominated and exhaustive pairwise comparisons between the existing and the recently added tuples could be avoided. Nevertheless, some algorithms are proposed specifically for a dynamic database; most of them are based on top-k. For instance, the work presented in [44] focuses on top-k and top-k dominating and reviews algorithms for evaluating continuous preference queries under the sliding window streaming model. Hence, an efficient skyline approach that avoids re-scanning the initial incomplete database after performing the data manipulation operations (insert and/or update) operations is extremely needed. The approach should update the result of the skylines to reflect the most recent information in the database while maintaining a reasonable cost and processing time.

TABLE 1. Symbols and description.

Symbol	Description
D	Dataset
dt, a	Tuple
n	Number of tuples
d	Dimension
new-D	Newly inserted data
dom-p	Dominated power
dom-p-list	Dominated power list
M	Total number 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
mid-max-dom-p	The threshold to eliminate some tuples based on *dom-p*

III. PRELIMINARIES

This section introduces the definitions and the necessary notations related to skyline queries in dynamic and incomplete databases that are used throughout this paper. These definitions and notations assist in clarifying the proposed approach. Table 1 summarizes the symbols used throughout the paper.

Definition 1 (Incomplete Database): A database, D, with m dimensions, $d = \{d_1, d_2, \ldots, d_m\}$ and n tuples $D = \{p_1, p_2, \ldots, p_n\}$ is incomplete denoted as D_I if and only if it contains at least a tuple p_i with missing value in one or more of its dimensions, d_j, where $d_j \in d$; otherwise, it is complete. We use the symbol '–' to denote a missing value. For example, the tuple p_i(–, 6, 2, –) demonstrates that the first and fourth dimensions have missing values.

Definition 2 (Comparable): Given a database D with n tuples $D = \{p_1, p_2, \ldots, p_n\}$, p_i and p_j are said to be comparable if and only if they have the same bitmap representation. Each tuple is represented as a bitmap representation where bit 1 is used to represent the dimensions with no missing values while bit 0 is used to represent the dimensions with missing values. Tuples that are comparable imply either they are complete, or they have missing values in the same dimension(s); otherwise, they are said to be incomparable. For instance, the bitmap representations of the tuples p_i(–, 6, 2, –) and p_j(7, –, –, 9) are 0110 and 1001, respectively. Thus, these two tuples are incomparable.

Definition 3 (Dominance Relationship): Given a database, D, with m dimensions, $d = \{d_1, d_2, \ldots, d_m\}$ and n tuples $D = \{p_1, p_2, \ldots, p_n\}$, p_i is said to dominate p_j denoted by $p_i \succ\succ p_j$ if and only if the following condition holds: $\forall d_k \in d, p_i.d_k \geq p_j.d_k \wedge \exists d_l \in d, p_i.d_l > p_j.d_l$. Throughout this paper, we assume that bigger values are preferred over smaller ones. For instance, consider the tuples p_i(7, 6, 2, 8) and p_j(7, 5, 1, 8), $p_i \succ\succ p_j$ as p_i is better than p_j in the second and third dimensions (true on the second part of the condition), while p_i is equal to p_j in the first and fourth dimensions (true on the first part of the condition). This definition applies to those tuples that are comparable (refer to *Definition* 2).

Definition 4 (Skylines): Given a database D with n tuples $D = \{p_1, p_2, \ldots, p_n\}$, p_i is a skyline of D if there are no other tuples $p_j \in D$ that dominates p_i. Skylines hold the *transitivity property* that means if p_i dominates p_j and p_j dominates p_k, this implies that p_i dominates p_k [27]. We use the symbol S to denote the set of skylines of a database D. However, in an incomplete database the *transitivity property* of skylines no longer holds due to *cyclic dominance* where none of the tuples is considered as skyline as every tuple is dominated by at least one other tuple.

Definition 5 (Revised Bitwise): Given a database D with n tuples $D = \{p_1, p_2, \ldots, p_n\}$, p_i and p_j with different bitmap representations, i.e., p_i and p_j are incomparable as defined in *Definition* 2. Nevertheless, p_i and p_j are comparable on their *revised bitwise* if it is not equal to 0, which is obtained by performing the AND operation on the bitmap representations of p_i and p_j. For instance, the bitmap representations of the tuples p_i(–, 6, 2, –) and p_j(7, –, –, 9) are 0110 and 1001, respectively. Based on *Definition* 2, these two tuples are incomparable. They are also incomparable based on their *revised bitwise* which is obtained by performing the AND operation on the 0110 and 1001; which results in 0000.

Definition 6 (Dominance Relationship on the Revised Bitwise): Given a database, D, with m dimensions, $d = \{d_1, d_2, \ldots, d_m\}$ and n tuples $D = \{p_1, p_2, \ldots, p_n\}$; and two tuples p_i and $p_j \in D$ with different bitmap representations. If their *revised bitwise* is not equal to 0, then p_i is said to dominate p_j on the *revised bitwise* denoted by $p_i \succ\succ p_j$ if and only if the following condition holds: $\forall d_k \in d', p_i.d_k \geq p_j.d_k \wedge \exists d_l \in d', p_i.d_l > p_j.d_l$ where d' is a set of dimensions whose revised bitwise representation is 1 and $d' \subset d$. For instance, consider the tuples p_i(–, 6, 2, –) and p_j(–, 5, –, 5), $p_i \succ\succ p_j$ as p_i is better than p_j in the second dimension.

Definition 7 (Dynamic Database): A database, D, is said to be dynamic denoted as D_D if the tuples in the database keep on changing in which a new tuple(s) is inserted into the database.

Definition 8 (Database State): Given an incomplete database, D_I (refer to *Definition* 1), its state is changed to a new state, D_{new}, due to the following operations:

- Insert Operation: $D_{new} = D_I \bigcup D_{<insert>}$ where $D_{<insert>}$ is a set of tuples to be inserted into the initial database, D_I.
- Delete operation: $D_{new} = D_I - D_{<delete>}$ where $D_{<delete>}$ is a set of tuples to be deleted from the initial database, D_I.
- Update operation: $D_{new} = (D_I - D_{<delete>}) \bigcup \cup D_{<insert>}$ where an update operation is considered as a delete operation followed by an insert operation.

Definition 9 (Skylines of D_{new} due to $D_{<insert>}$): Given a database D with n tuples $D = \{p_1, p_2, \ldots, p_n\}$, p_i is a skyline of D if there are no other tuples $p_j \in D$ that dominates p_i. Assume that the set of skylines derived based on D is S. Given the $D_{<insert>} = \{q_1, q_2, \ldots, q_k\}$, the $D_{new} = \{p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_k\}$, the set S is still valid iff for

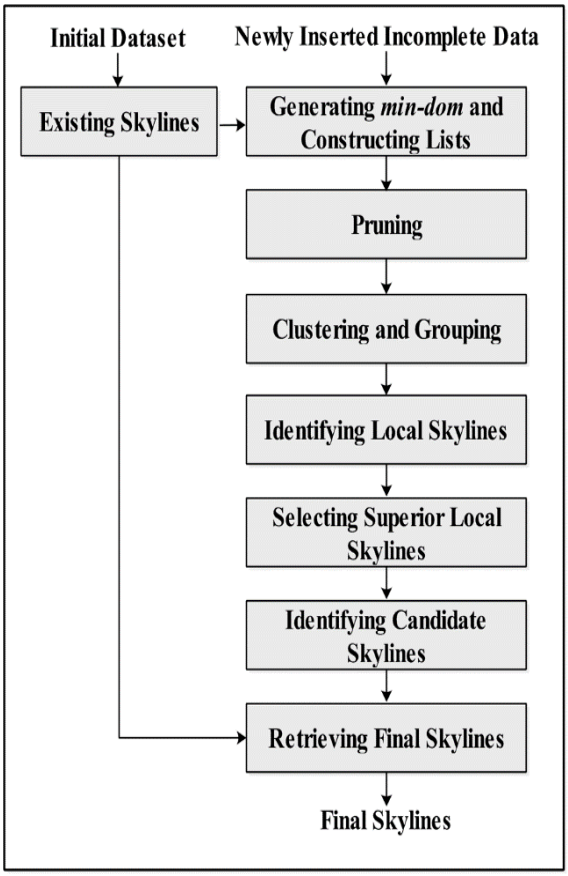


FIGURE 3. Phases of IDSA algorithm.

every $p_l \in S$ there are no tuples $q_k \in D_{<insert>}$ that dominates p_l, otherwise a new set of skylines S' needs to be computed.

IV. IDSA ALGORITHM

This section presents the proposed algorithm named *Incomplete Dynamic Skyline Algorithm* (IDSA), which aims at identifying skyline results over a database with incomplete data when changes are made towards the database due to the insert operations. The proposed solutions emphasize on avoiding the unnecessary skyline computations to introduce the new skyline result when new tuples are added into the database. The process of re-computing skylines over the update database is inevitable as the skyline results produced earlier (before insertion) is no longer valid. However, it is unwise to blindly reapply the skyline technique over the entire updated database. This is due to the fact that many tuples have been analyzed previously and might not be affected by the changes made by the insert operations and re-examining them results in many unwanted expensive pairwise comparisons. Thus, to avoid the unnecessary pairwise comparisons while attempting to identify the skyline result after the insert operation on the initial database, it is important to perform a progressive scanning process considering the most candidate skylines in the initial incomplete database and the newly added data items to identify the skyline results based on updated content. Intuitively, the idea of a progressive

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	*

FIGURE 4. The initial incomplete database.

scan can lead to a significant reduction in the number of pairwise comparisons and processing time in the subsequent computation of skylines. The IDSA algorithm comprises of seven phases, namely: Generating *min-dom* and Constructing Lists, Pruning, Clustering, and Grouping, Identifying Local Skylines, Selecting Superior Local Skylines, Identifying Candidate Skylines, and Retrieving Final Skylines. Each phase plays a crucial role in identifying the skylines of the incomplete database. The main purpose of the IDSA is to derive the new skylines after a change has been made towards the initial incomplete database due to the insert operation. Taking into consideration the issue of reducing the search space by avoiding the re-computation process of the skyline on the entire database. **Figure 3** illustrates the phases of the IDSA algorithm presented in this paper. These phases are further explained in detail in the following subsections.

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	4	8
m_{21}	7	1	7	9	*	7
m_6	*	4	*	8	7	9

FIGURE 5. Skylines of the initial incomplete database (before insert operation).

Since the IDSA algorithm is designed to work on a dynamic incomplete database, thus, we assume that the initial database has tuples with some missing values in one or more dimensions as shown in **Figure 4**. For simplicity and to focus on the main issue concern in this work, we assume that the skyline result of the existing tuples of the database before the insert operation has been identified using the skyline technique proposed in [31]. The skylines set *E-skylines* of the initial incomplete database (before insert operation) are elucidated in **Figure 5**. It should be noted that the work introduced in [31] is limited to resolve the issue of processing skyline queries over incomplete data assuming no changes towards the database and the skyline computation performed only once. However, this work emphasizes dealing with the problem of skyline queries when a database state is changed due to the data manipulation operation(s); in which a new tuple(s) is inserted into the database or updated from the database. Thus, a new set of skylines should be derived to reflects the latest changes in the database contents. To achieve our main aim, i.e. avoiding unnecessary computations of skylines and utilizes the skylines of the previous state of the database (before insert and/or update operation) during the process of identifying a new set of skylines. However, re-computing the skylines over the entire updated database is unwise as not only it will incur undesirable storage costs, but also results in many exhaustive unnecessary pairwise comparisons in the subsequent processes of skyline computations. Hence, optimizing the skyline process over a dynamic and incomplete database is an issue to be dealt with in this work.

To explain the functionality of the IDSA technique new tuples have been inserted into the initial incomplete database as elaborated in Figure 6. The figure indicates that 40 new tuples with ID (n_0 – n_{39}) have been added into the initial incomplete database.

A. GENERATING MIN-DOM AND CONSTRUCTING LISTS

In this section, we explain the process of how lists for each dimension of newly inserted tuples are constructed. The contents of lists are determined by comparing the newly inserted tuples with the virtual tuple called min-dom. The entire process of constructing the lists starts by deriving a virtual tuple from the existing skyline set, E-skylines. Virtual tuple, min-dom must contain the same number of dimensions as the

ID	d_1	d_2	d_3	d_4	d_5	d_6
n_0	*	3	6	7	5	1
n_1	4	*	1	4	8	1
n_2	3	9	2	*	2	6
n_3	6	8	1	9	2	*
n_4	*	1	1	1	9	4
n_5	9	4	*	7	9	6
n_6	8	*	9	1	1	1
n_7	7	2	3	2	*	4
n_8	6	6	2	4	7	*
n_9	*	8	2	5	*	7
n_{10}	2	8	9	*	7	2
n_{11}	6	8	5	2	*	9
n_{12}	5	7	6	6	8	*
n_{13}	*	8	6	*	1	5
n_{14}	4	*	3	2	5	2
n_{15}	6	1	*	3	7	2
n_{16}	6	3	3	5	*	5
n_{17}	7	2	4	1	*	6
n_{18}	3	6	7	1	4	*
n_{19}	*	3	5	6	8	8
n_{20}	6	8	5	*	3	7
n_{21}	5	*	2	3	6	5
n_{22}	*	7	*	6	*	4
n_{23}	2	2	*	6	5	6
n_{24}	1	*	1	8	2	*
n_{25}	*	4	8	3	6	8
n_{26}	6	1	8	6	*	5
n_{27}	5	7	4	*	6	6
n_{28}	4	6	5	5	*	5
n_{29}	*	2	5	3	2	*
n_{30}	6	*	3	3	*	9
n_{31}	7	7	*	1	8	2
n_{32}	8	5	8	*	6	9
n_{33}	3	*	6	3	8	*
n_{34}	4	2	*	9	7	7
n_{35}	4	*	5	5	7	6
n_{36}	1	2	9	*	7	9
n_{37}	7	1	*	6	*	2
n_{38}	1	7	1	*	7	*
n_{39}	1	8	*	6	7	2

FIGURE 6. Newly inserted tuples to the initial incomplete database.

entire database contains. Each dimension value of min-dom is determined by considering the lowest value in each dimension of the E-skylines set. The min-dom is represented using the following formula.

$$\boldsymbol{min - dom} = \{\boldsymbol{V.d_i} : \boldsymbol{V.d_i} = \min\{\boldsymbol{p.d_i} : \boldsymbol{p} \in \boldsymbol{E - skylines}\}, \quad \boldsymbol{1 \leq i \leq n}\}$$

where n is the dimensionality of *E-skylines*.

After the virtual tuple (*min-dom*) is derived, the process of constructing the lists starts by comparing the *min-dom* dimension values with the values of the corresponding dimension of the newly inserted tuple(s). Note that the number of lists should be equal to the dimensionality (including dimensions with missing values) of the database. If the dimension value of a newly inserted tuple is either missing or less than the value of the corresponding dimension of *min-dom* then the *ID* of the newly inserted tuple is added to the corresponding list. This process continues by reading the entire newly inserted

ID	d_1	d_2	d_3	d_4	d_5	d_6
Min-dom	4	1	4	8	4	6

FIGURE 7. The min-dom virtual tuple derived from the skylines of the initial incomplete database.

u_1	u_2	u_3	u_4	u_5	u_6
n_0	n_1	n_1	n_0	n_2	n_0
n_2	n_6	n_2	n_1	n_3	n_1
n_4	n_{14}	n_3	n_2	n_6	n_3
n_9	n_{21}	n_4	n_3	n_7	n_4
n_{10}	n_{24}	n_5	n_4	n_9	n_6
n_{13}	n_{30}	n_7	n_5	n_{11}	n_7
n_{18}	n_{33}	n_8	n_6	n_{13}	n_8
n_{19}	n_{35}	n_9	n_7	n_{16}	n_{10}
n_{22}		n_{14}	n_8	n_{17}	n_{12}
n_{23}		n_{15}	n_9	n_{20}	n_{13}
n_{24}		n_{16}	n_{10}	n_{22}	n_{14}
n_{25}		n_{21}	n_{11}	n_{24}	n_{15}
n_{29}		n_{22}	n_{12}	n_{26}	n_{16}
n_{33}		n_{23}	n_{13}	n_{28}	n_{18}
n_{36}		n_{24}	n_{14}	n_{29}	n_{21}
n_{38}		n_{30}	n_{15}	n_{30}	n_{22}
n_{39}		n_{31}	n_{16}	n_{37}	n_{24}
		n_{34}	n_{17}		n_{26}
		n_{37}	n_{18}		n_{28}
		n_{38}	n_{19}		n_{29}
		n_{39}	n_{20}		n_{31}
			n_{21}		n_{33}
			n_{22}		n_{37}
			n_{23}		n_{38}
			n_{25}		n_{39}
			n_{26}		
			n_{27}		
			n_{28}		
			n_{29}		
			n_{30}		
			n_{31}		
			n_{32}		
			n_{33}		
			n_{35}		
			n_{36}		
			n_{37}		
			n_{38}		
			n_{39}		

FIGURE 8. Lists constructed for the newly inserted tuples with their ID's.

tuples considering all dimensions. The following equation demonstrates the formula of generating lists based on *min-dom*.

$$u_i = \{n_k \in new - D : n_k.d_i < V.d_i, V.d_i \in min - domorn_k.d_i \text{ is missing}\}, \quad 1 \le i \le n$$

where n is the dimensionality of *E-skylines*, and *new-D* is a set of newly inserted data

We believe that this step is very beneficial and helps in simplifying the process of identifying the skylines on a database with incomplete data when changes are made towards the database with insert operation. We conclude that the construction of lists with the help of *min-dom* leads to prevent a non-trivial number of the dominated newly inserted tuples from further being processed. Most importantly, this step contributes significantly to reducing the number of pairwise comparisons that need to be performed to derive the new skylines results after the insert operation.

Based on skyline results reported in **Figure 5**, we can notice that the minimum value in dimension d_1 is 4, the minimum value in dimension d_2 is 1, the minimum value in

***Algorithm* 1**

Input: A set of final skylines, *E_ skylines*, and newly inserted tuples, *new_D*
Output: Lists of tuples with their ID's, *Lists* = $\{u_1, u_2, \ldots, u_n\}$

1. **foreach** *dimension* d_i of d in *E-skylines* **do**
2. **if** the value of d_i of all tuples in *E-skylines* is missing **then**
3. set d_i value of *min-dom* = '*' // * denotes *missing*
4. **else**
5. set d_i value of *min-dom* = minimum value found in dimension d_i
6. **end**
7. **end**
8. **foreach** *dimension* d_i in d **do**
9. **foreach** tuple t_j in *new-D* **do**
10. **if** the value of $t_j.d_i$ is either missing or $t_j.d_i < min\text{-}dom.d_i$ **then**
11. add the *ID* of the t_j to $Lists.u_i$
12. **end**
13. **end**
14. **end**
15. **return** *Lists*

FIGURE 9. Algorithm for generating min-dom and constructing lists.

dimension d_3 is 4, the minimum value in dimension in d_4 is 8, the minimum value in dimension in d_5 s 4 and d_6 is 6. Therefore, the derived *min-dom* virtual tuple produced based is the skylines presented in **Figure 5** is (4, 1, 4, 8, 4, 6) as shown in **Figure 7**.

With the help of *min-dom,* the lists are generated containing the tuple *IDs* of corresponding dimensions whose values are either missing or less than the value of the corresponding dimension of *min-dom*. **Figure 8** depicts the lists generated from the inserted tuples. From the figure, we can notice that list u_1 has been constructed based on dimension d_1 which contains the *ID* of 18 tuples. Similarly, list u_2 that has been created based on dimension d_2 includes the *ID* of 7 tuples. The process ends by creating a list u_3, u_4, u_5, and u_6 that contain the *ID* of tuples 21, 37, 17, and 22, respectively.

Figure 9 presents the detailed steps of generating *min-dom* and constructing lists algorithm. The input of the algorithm encompasses the skylines of the initial incomplete database (before insert), *E-skylines,* and the newly inserted tuples, *new-D*, while the output of the algorithm is a set of constructed lists containing the *IDs* of the newly added tuples, *new-D*. The algorithm starts by analyzing each dimension of the skyline tuples, *E-skylines* (step 1). If no value is found in dimension d_i for all tuples in the skylines, *E-skylines* (step 2), then it sets the value of the dimension d_i of *min-dom* to '*', which denotes that the value of the d_i of *min-dom* is not present (missing) (step 3). Otherwise, the value of dimension d_i of *min-dom* is set to be the smallest value found in dimension d_i concerning all tuples in the skyline result, *E-skylines* (step 5). Steps 1 to 5 are repeated until the *min-dom* virtual tuple has been formed. Next, the lists are constructed based on the values of the dimensions of the newly inserted tuples as follows. First, each dimension, d_i, of the newly inserted tuples, *new-D*, is analyzed (step 8-9).

If the value of d_i of the tuple t_j is either missing or less than the value of the corresponding dimension d_i of *min-dom*, then, the *ID* of the tuple t_j is inserted into the list, $Lists.u_i$ (steps 10 – 11). This process continues until all the tuples in each dimension, *new-D*, is read (step 8-14). Lastly, the algorithm returns the created lists containing the *ID* of the newly inserted tuples, *new-D* (step 15).

Time complexity analysis of Algorithm 1.

Let $|E\text{-}skylines| = m$, $|new\text{-}D| = n$ and $|d| = k$, where $|X|$ denotes the number of items in X. Determining if $min\text{-}dom.d_i =$ '*' or finding the minimum value in each dimension d_i takes $O(m)$ time. Thus, the first part of Algorithm 1 (i.e., lines 1-7) takes $O(k \times m)$ time. Since for each tuple t_j in *new-D*, it's every dimension value $t_j.d_i$must be checked if it is missing or less than $min\text{-}dom.d_i$, the second part of Algorithm 1 (i.e., lines 8-15) takes $O(k \times n)$ time. Hence, the running time of Algorithm 1, $T_1(m, n, k)$, is

$$T_1(m, n, k) = O(k \times (m + n)).$$

If we assume that $k = O(1)$, i.e., the dimensionality of databases is fixed, then $T_1(m, n) = O(m + n)$.

B. PRUNING

The main purpose of this phase is to eliminate the dominated tuples of the newly inserted data to the initial incomplete database before applying the skyline technique. To do so, the tuples in each constructed list are scanned in sequence to count their dominated power (*dom-p*) value followed by removing those tuples with the high *dom-p* value from further processing. The idea of removing those tuples with a high *dom-p* value is very useful as it assists in a safe early termination for the scanning process. This is because these tuples will be dominated by the candidate skylines or by the tuples present in *new-D*. Therefore, these tuples have no potential to form the final skylines of the updated database. Hence, these tuples can be safely removed to avoid many unwanted pairwise comparisons. We argue that the idea of pruning is very beneficial and contributes to optimizing the skyline process over an incomplete database with insert operation.

The process starts by scanning each constructed list u_i and calculating the *dom-p* value of each tuple based on their appearance in the constructed lists. The *dom-p* value indicates the domination value in which the tuple is being dominated by *min-dom*. If the tuple has a large *dom-p* value, this means the tuple has been dominated by *min-dom* in a large number of non-missing dimensions and vice versa. This process ends by generating a 2D array named *dom-p-list* that consists of the *ID* of the tuples and their *dom-p* value. The following equation demonstrates the formula for computing the *dom-p* value of a newly inserted tuple t_i:

$$dom - p\,(t_i) = \sum_{u \in Lists} \chi_u\,(t_i)$$

where u is the generated list based on the newly inserted tuples, *new-D*.

The idea of the pruning technique works as follows. If the *dom-p* value of the tuple t_i is equal to $(d - 1)$ where d is the number of dimensions, then this indicates that the tuple t_i is being dominated by *min-dom* at least in $(d - 1)$ dimensions. Thus, it shows that the dominated tuple t_i will not be part of the final new skylines and can be safely removed before applying the skyline technique. The elimination of the dominated tuples will help in avoiding many unnecessary pairwise comparisons during deriving the new skylines based on the recent update status of the incomplete database. The idea of pruning relies on generating a threshold value named *mid-max-dom-p* based on the highest *dom-p* value produced in the corresponding *dom-p-list* divided by 2. Then, a comparison is performed between the *mid-max-dom-p* and *dom-p* value of each tuple t_i in *dom-p-list*. Notice that the *mid-max-dom-p* value is rounded up to the next higher integer number if the value is produced of *mid-max-dom-p*. The formal definition of pruning is given below.

Definition 10 (Pruning): Given a tuple $t_i \in dom - p - list$ of newly inserted tuples and let *dom-p* equal to the domination power of t_i. Then tuple t_i is removed from the *dom-p-list* list if and only if the following condition hold ($t_i.dom\text{-}p > mid\text{-}max\text{-}dom\text{-}p$). Otherwise, the tuple t_i remains for further processing.

$$dom\_p\_list = \{\forall t_i : iff dom - p \text{ of } t_i < mid - max - dom - p\}$$

With the help of the proposed pruning technique, there will be around 40% to 50% of the newly inserted tuples successfully removed from further processing. This means a 50% reduction of the pairwise comparisons which further leads to a significant reduction in the total processing time of the skyline process.

From the running database example, the list of tuples with their *ID*'s derived from the previous phase is scanned based on their dimensions. The process begins by scanning list u_1 to count the *dom-p* value of each tuple in list u_1. The process continues to read all tuples in lists u_2, u_3, u_4, u_5, and u_6, respectively. Eventually, a new 2D array named *dom-p-list* is created which comprises of the *ID* of the tuple and its corresponding *dom-p* value. It should be noted that every time the tuple is read, the value of *dom-p* of the corresponding tuple increases by 1. **Figure 10** demonstrates the result of the *dom-p-list* produced based on the running database example. From the figure, it could be observed that the *dom-p* of the newly inserted tuple n_0 is equal to 3. This is because tuple n_0 has appeared 3 times in many lists including u_1, u_4, and u_6. Similarly, the *dom-p* of n_2 is equal to 4, indicates that n_4 has been found in 4 different lists u_1, u_2, u_4, and u_5.

One can notice that, in Figure 10, the highest *dom-p* value found in *dom-p-list* is 5, which denotes that one or more tuples in *dom-p-list* have been examined 5 times. Hence, the *mid-max-dom-p* value is equal to 3 since $\lceil 5/2 \rceil = 3$. Thus, all tuples with a *dom-p* value greater than *mid-max-dom-p* can be safely removed from further processing. This is because these tuples are dominated by other tuples in many non-missing dimensions and therefore will not be listed in the skyline

Tuple ID	dom-p
n_0	3
n_2	4
n_4	4
n_9	4
n_{10}	3
n_{13}	4
n_{18}	3
n_{19}	2
n_{22}	5
n_{23}	3
n_{24}	5
n_{25}	2
n_{29}	4
n_{33}	4
n_{36}	2
n_{38}	4
n_{39}	4
n_1	4
n_6	4
n_{14}	4

Tuple ID	dom-p
n_{21}	4
n_{30}	4
n_{35}	2
n_3	3
n_5	2
n_7	4
n_8	3
n_{15}	3
n_{16}	4
n_{31}	3
n_{34}	1
n_{37}	4
n_{11}	2
n_{12}	2
n_{17}	2
n_{20}	2
n_{26}	3
n_{27}	1
n_{28}	3
n_{32}	1

FIGURE 10. The domination power of the newly inserted tuples.

Tuple ID	dom-p
n_0	3
n_{10}	3
n_{18}	3
n_{19}	2
n_{23}	3
n_{25}	2
n_{36}	2
n_{35}	2
n_3	3
n_5	2
n_8	3

Tuple ID	dom-p
n_{15}	3
n_{31}	3
n_{34}	1
n_{11}	2
n_{12}	2
n_{17}	2
n_{20}	2
n_{26}	3
n_{27}	1
n_{28}	3
n_{32}	1

FIGURE 11. Result of *Candidate Tuples* After Pruning.

result. This process leads to a significant reduction in the number of pairwise comparisons in identifying the skylines based on the recently updated contents of the database due to the elimination of many dominated tuples from further processing. **Figure 11** depicts the result of the pruning process elucidating that many tuples t_i with $dom\text{-}p.t_i > 3$ have been removed from the *dom-p-list*. This includes removing tuples n_2, n_4, n_9, n_{13}, n_{22}, n_{24}, n_{29}, n_{33}, n_{39}, n_1, n_6, n_{14}, n_{21}, n_{30}, n_7, n_{16}, and n_{37} from the *dom-p-list*. A total of 17 tuples out of 40 have been removed successfully before applying the skyline technique, which means a 43% reduction from the total amount of newly inserted tuples. It is obvious that if we compare these removed tuples with the skylines of the initial incomplete database (before insertion), none of these removed tuples can be in the skyline result of the recently updated database as these tuples will be dominated by some other skylines in *E-skylines*. Hence, to avoid this expensive process these tuples are deleted in the early stage to prevent the prohibitive expensive pairwise comparisons during skyline operation.

Figure 12 presents the detailed steps of the algorithm that generates the *dom-p-list*. Given a set of lists, *Lists*, containing the *ID*'s of tuples as an input to the algorithm, while the output

Algorithm 2

Input: Lists of tuples with their ID's, $Lists = \{u_1, u_2, \ldots, u_k\}$
Output: A list of tuples with their *dom-p*, *dom-p-list*

```
1.  foreach list u_i of Lists do
2.      foreach tuple t_j in u_i do
3.                  If t_j is read before then
4.                    dom-p.t_j ++
5.                  else
6.                    add the corresponding ID of t_j to dom-p-list
7.                    set dom-p.t_j =1
8.                  end
9.      end
10.  end
11.  return dom-p-list
```

FIGURE 12. *dom-p-list* Algorithm.

is a list of tuples with their corresponding domination powers, *dom-p-list*. The algorithm starts by reading each tuple in all lists (steps 1-2), if the tuple has been read before (step 3), then the domination power, *dom-p* of the tuple is incremented by 1. Else, if the tuple has not been read before, then add the tuple to the *dom-p-list* and set the domination power of the tuple to be 1 (steps 5-7). This process continues until all tuples in all lists are read (steps 1-10). The algorithm ends by returning the *dom-p-list*.

Time complexity analysis of Algorithm 2

Let $T_2(k, n)$ denote the running time of Algorithm 2. From the analysis of Algorithm 1 above, we can assume that for each u_i in *Lists*, $|u_i| = O(n)$ where $n = |new\text{-}D|$. Since $|d| = k$, we have $|Lists| = k$. To compute the *dom-p* value of each tuple t_j belonging to some u_i, the algorithm scans every list in *Lists*. Thus, $T_2(k, n) = k \times O(n) = O(k \times n)$. Since *dom-p-list* may contain almost all tuples of *new-D*, $|dom\text{-}p\text{-}list| = O(n)$.

Figure 13 elucidates the algorithm steps for deriving new candidate tuples of the newly inserted tuples. The algorithm input consists of the *dom-p-list* and the output of the algorithm is a list of new candidate tuples. The algorithm begins by analyzing the values of the dominated power, *dom-p* of the tuples in *dom-p-list* to determine the maximum *dom-p* value, *max-dom-p,* and generates the *mid-max-dom-p* (steps 1-2). The value of *mid-max-dom-p* is generated by dividing the *max-dom-p* by 2 and round-up the result to the next higher integer number. Then the algorithm scans each tuple in *dom-p-list* to eliminate those tuples with a *dom-p* value greater than the derived *mid-max-dom-p* (step 3-7). Lastly, the algorithm returns the remaining tuples as a new set of new candidate tuples (step 8).

C. CLUSTERING AND GROUPING

The idea of clustering and grouping implemented in this phase has been inspired by our previous approach in [31]. The main aim of the clustering and grouping phase is further to simplify the process of scanning the tuples present in the candidate skyline list, *CD,* by dividing the tuples into different clusters based on their domination power (*dom-p*) values.

Algorithm 3

Input: *dom-p-list*, …

Output: A candidate tuples set, *CD*={…}

1. Let *max-dom-p* = highest value in *dom-p-list*
2. Let *mid-max-dom-p*= $\lceil max - dom - p/2 \rceil$
3. **foreach** tuple t_i in *dom-p-list* **do**
4. **if** $t_{i\cdot dom\text{-}p} <$ *mid-max-dom-p* **then**
5. add dt_i to *CD*
6. **end**
7. **end**
8. **return** *CD*

FIGURE 13. Algorithm for identifying the new candidate tuples.

Cluster C_1 *dom-p*= 1		*Cluster* C_2 *dom-p*= 2		*Cluster* C_3 *dom-p*= 3	
ID	*dom-p*	*ID*	*dom-p*	*ID*	*dom-p*
n_{34}	1	n_{19}	2	n_0	3
n_{27}	1	n_{25}	2	n_{10}	3
n_{32}	1	n_{36}	2	n_{18}	3
		n_{35}	2	n_{23}	3
		n_5	2	n_3	3
		n_{11}	2	n_8	3
		n_{12}	2	n_{15}	3
		n_{17}	2	n_{31}	3
		n_{20}	2	n_{26}	3
				n_{28}	3

FIGURE 14. Clustering of tuples based on domination power.

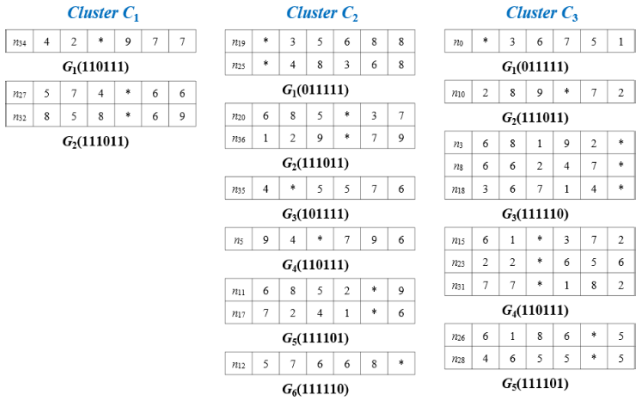


FIGURE 15. Groups of distinct clusters.

With n distinct *dom-p* values, the tuples are divided into n distinct clusters $C_1, C_2, \ldots, C_n$, where all tuples with the same *dom-p* value belong to the same cluster. It is obvious that applying the concept of the domination power on a cluster is very beneficial and assists to avoid the unwanted comparison among tuples with the entire candidate skyline set if those tuples are being dominated by other tuples within the cluster. The idea of clustering and grouping has been proven to be an effective and practical solution for processing skyline queries over both complete and incomplete data.

Many variations of skyline techniques have exploited the idea of divide-and-conquer in deriving skylines [31]. Due to the presence of data incompleteness and the frequent insert operation towards the database, the *transitivity property* of the skyline no longer holds, and the dominance relationship will be *cyclic*. To prevent these challenges and ensure the correctness and completeness of the skyline result, an efficient grouping technique proposed in [31] has been employed. The grouping process attempts to divide the tuples in one cluster C_i into many manageable subclusters called groups $C_i.G_1, C_i.G_2, \ldots, C_i.G_n$, based on the *bitmap representation* of the tuples. For simplicity and without loss of generality, the results of the pruning process demonstrated in **Figure 11** indicate that there are three types of tuples based on their *dom-p* value. Therefore, according to the idea of clustering, candidate tuples are divided into 3 different clusters. Cluster C_1 contains those tuples that have *dom-p* equal to 1. Similarly, cluster C_2 and C_3 encompass tuples whose *dom-p* values are 2 and 3, respectively. **Figure 14** depicts the result of the clustering tuples process based on domination power. It should be noted that these clusters are further partitioned into groups. Hence, tuples with similar *bitmap representation* are placed into one group to ensure that the *transitivity property* of the skyline technique always holds, and the issue of the *cyclic dominance* is eliminated. **Figure 15** elucidates that cluster C_1 is divided into two groups, namely: $C_1.G_1$ and $C_1.G_2$ with bitmap representation 110111 and 111011, respectively. Similarly, clusters C_2 and C_3 are also partitioned into six distinct groups ($C_2.G_1$–$C_2.G_6$) and 5 groups ($C_3.G_1$–$C_3.G_5$), respectively.

Here, we provide a brief explanation of the remaining phases of the proposed approach for retrieving skylines on a database with dynamic and incomplete data, while the details can be found in [31]. The input of *Algorithm* 2 presented in [31] will be the candidate tuples set *CD* which contains the *ID*s of the newly inserted tuples and their *dom-p* values. While the output is a set of clusters containing these tuples based on their *dom-p* values. This process assists in simplifying the process of identifying the new set of skylines based on the most recent state of the database. The input of *Algorithm* 3 comprises of a set of clusters for the newly inserted tuples while the output is a new set of groups for the newly inserted tuples.

D. IDENTIFYING LOCAL SKYLINES

This phase intends to determine the local skylines of each cluster for the newly inserted tuples. This is achieved by, first, deriving the skylines of each group that belongs to the constructed clusters, then followed by identifying the skylines of the entire cluster. This is performed by comparing the group skylines against each other to remove the dominated tuples and exclude them from further processing. This phase contributes significantly to reducing the number of tuples to be processed in the next phases which, in its turn, decreases the number of pairwise comparisons that need to be performed to identify the new skylines after the insert operation. The process of identifying local skylines of the newly inserted tuples presented in this section is carried out by utilizing

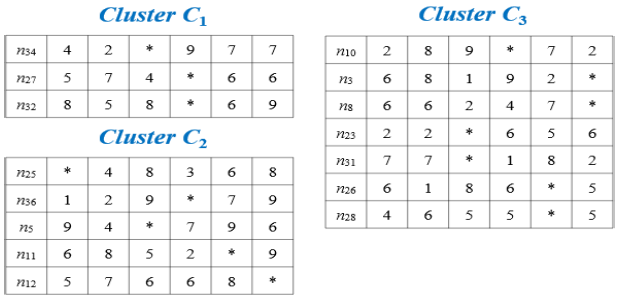


FIGURE 16. Local skylines of clusters.

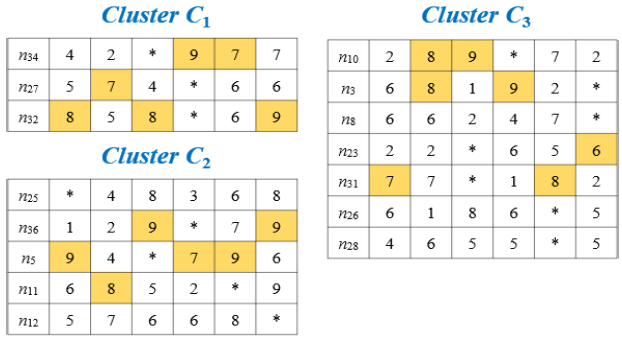


FIGURE 17. Selecting superior local skylines of clusters.

Algorithms 4 and 5 proposed in [31]. The process starts by running *Algorithm* 4 on the created groups of each cluster from the previous phase to identify the group skylines. Thus, the algorithm's input is the set of groups of each cluster, while the output is a set of group skylines. This is followed by running *Algorithm* 5 to retrieve the skylines of the clusters. The algorithm input includes the skylines of all groups in each cluster, while the output is the local skylines of the cluster.

Figure 16 demonstrates the results of identifying local skylines of each cluster for the newly inserted tuples. Notice that none of the tuples are dominating one another in cluster C_1. Thus, tuples n_{34}, n_{27}, and n_{32} are identified as local skylines of C_1. In cluster C_2, tuples n_{25}, n_{36}, n_5, n_{11}, and n_{12} have been identified as the local skylines in which four tuples have been eliminated from further processing as they are being dominated by these local skylines of Cluster C_2. In cluster C_3, three tuples have been removed, namely: n_0, n_{18}, and n_{15} as these tuples have been dominated by other tuples in cluster C_3. Thus, seven tuples remain in C_3 and are reported as the local skylines of cluster C_3. It should be noticed that there were 22 tuples before the skyline processing technique was implemented and seven dominated tuples have been removed successfully. It comprises of 32% of reduction of tuples before identifying the new set of the final skylines. Hence, eliminating these dominated tuples is crucial to avoid many unnecessary pairwise comparisons which in turn results in a significant reduction in the processing time of the skyline process to produce the new set of the skylines.

E. SELECTING SUPERIOR LOCAL SKYLINES

This phase aims to further refine the local skylines of each cluster of the newly inserted tuples produced in the previous phase to identify the superior local skylines of each cluster for those inserted tuples. The idea of identifying the superior local skylines relies on scanning the local skylines of each cluster aiming at identifying the highest values in any of the non-missing dimensions. Then all local skylines with the highest values are combined and reported as superior local skylines of the clusters. In contrast, those local skylines with values less than the highest values found in each dimension can be safely removed. This is because the local skylines of each cluster will be compared with the local skylines of other clusters. Therefore, if these removed tuples with values less than the highest are compared against the tuples of the other clusters they will be dominated. With careful analysis of those removed tuples, we can conclude that these removed tuples are not better than the superior local skylines and have no contribution in forming the final skylines. Hence, the idea of prior deletion of these dominated tuples helps in a significant reduction in the number of pairwise comparisons that need to be performed among the local skylines of the clusters. We believe this refinement step facilitates in simplifying the skyline process when the changes towards the database are made through insert operation. The detailed steps of selecting superior local skylines have been outlined in *Algorithm* 7 in [31]. The algorithm input encompasses the local skylines of the cluster C_{i-n}, while the output of the algorithm is a set of superior local skylines of the cluster for the newly inserted tuples. **Figure 17** illustrates the detailed process of identifying the superior local skylines of clusters for the running database example that has been used throughout this paper. Notice that the shaded cells denote the superior local skylines of each cluster. For instance, n_{34}, n_{27}, and n_{32} are the superior skylines of cluster C_1 as these tuples have the highest values in the non-missing dimensions 4, 2, and 1 respectively. Similarly, tuples n_{36} n_5, and n_{11} are reported as the superior local skylines of cluster C_2. However, the tuples n_{25} and n_{12} have been removed from further processing due to the fact that tuples n_{25} and n_{12} are dominated by other tuples of the cluster. Likewise, n_{10}, n_3, n_{23}, and n_{31} are reported as the superior local skylines of the cluster, C_3. While the tuples n_8, n_{26}, and n_{28} are eliminated from cluster C_3.

Figure 18 elucidates the final set of the superior local skylines of the clusters for our running database example. We argue that identifying superior local skylines has a crucial role in eliminating a significant portion of the tuples from further processing. It has been concluded that the total number of tuples that could be safely removed from these three clusters, C_1, C_2, and C_3 is 5, which represents up to a 33% reduction in the amount of data to be considered in the next phases. This in turn leads to a significant reduction in the number of pairwise comparisons performs to retrieve the final skyline set. It is important to notice that both processes, identifying

Cluster C_1

n_{34}	4	2	*	9	7	7
n_{27}	5	7	4	*	6	6
n_{32}	8	5	8	*	6	9

Cluster C_2

n_{36}	1	2	9	*	7	9
n_5	9	4	*	7	9	6
n_{11}	6	8	5	2	*	9

Cluster C_3

n_{10}	2	8	9	*	7	2
n_3	6	8	1	9	2	*
n_{23}	2	2	*	6	5	6
n_{31}	7	7	*	1	8	2

FIGURE 18. Superior local skylines of clusters.

ID	d_1	d_2	d_3	d_4	d_5	d_6
n_{34}	4	2	*	9	7	7
n_{32}	8	5	8	*	6	9
n_{36}	1	2	9	*	7	9
n_5	9	4	*	7	9	6
n_{11}	6	8	5	2	*	9
n_{10}	2	8	9	*	7	2
n_3	6	8	1	9	2	*
n_{31}	7	7	*	1	8	2

FIGURE 19. Candidate skylines.

local skylines and selecting superior local skylines, are carried out concurrently as clusters and groups are independent of each other. The concurrent running of these two processes is very crucial and leads to a great reduction in the processing time.

F. IDENTIFYING CANDIDATE SKYLINES

This phase is responsible to remove the superior local skylines of clusters that are dominated by the superior local skylines of other clusters from further processing. This is accomplished by comparing the superior skylines of clusters against each other aiming at reducing the number of candidate skylines. The detailed steps of the algorithm for identifying candidate skylines, Algorithm 5 can be found in [31]. The input of the algorithm includes a set of superior local skylines of clusters for the newly inserted tuples, while the output is a set of candidate skylines for the newly inserted tuples. For instance, the superior local skylines, n_{34}, n_{27}, and n_{32}, of cluster C_1 will be compared against the superior local skylines of clusters C_2 and C_3, and the superior local skylines of C_2 will be compared with the superior skylines of cluster C_3. If the superior local skylines of C_1 dominate the superior local skylines of C_2 and C_3 based on common non-missing dimensions, then these superior local skylines of C_2 and C_3 are removed immediately. However, if the superior local skylines of C_1 are dominated by the superior local skylines of C_2 or C_3 then that tuples of C_1 will be removed at the end of the iteration. This process continues until the candidate skylines of the entire newly inserted tuples are returned. Figure 19 depicts the final list of the candidate skylines of the clusters C_1, C_2, and C_3 of the database running example.

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	4	8
m_6	*	4	*	8	7	9
n_{32}	8	5	8	*	6	9
m_{11}	6	8	5	2	*	9

FIGURE 20. The final skylines.

G. RETRIEVING FINAL SKYLINES

This section presents the last phase of the IDSA algorithm aiming at identifying the skylines of an incomplete database when the state of a database changed due to the insert operation(s) made toward the initial incomplete database. This phase attempts to retrieve the new skyline result based on the most recent updated contents of the database. To this end, the candidate skylines of the newly inserted tuples are compared against the skylines of the initial incomplete database before the insert operation.

The process of retrieving the final skyline might lead to one of the following three cases.

1) ***Case* 1:** If the result of the pairwise comparison indicates that the candidate skylines of the newly added tuples are dominated by the skylines of the initial incomplete database, then these candidate skylines will not contribute to forming the new set of skylines and can be safely removed.
2) ***Case* 2:** If the result of the pairwise comparison introduces that the candidate skylines of the newly added tuples dominate the skylines of the initial incomplete database, then these skylines are no longer valid and should be removed from the new set of the final skylines.
3) ***Case*3:** if the result of the pairwise comparison denotes that the candidate skylines of the newly inserted tuples and the skyline of the initial incomplete data do not dominate each other. Then, these candidate skylines of the newly inserted tuples will be considered as part of the new set of the final skylines.

The retrieved tuples from this process represent the skylines of the latest state of the incomplete database after the insert operation. The detailed steps of the algorithm (Algorithm-5) can be found in the work in [31]. The input of the algorithm is a set of candidate skylines of the newly inserted tuples and the skylines of the initial incomplete database before the insert operation. While the output of the algorithm is the skylines set of the most recent state of the database. **Figure 20** depicts the new final skylines to be retrieved to the end-user. We can conclude that the tuples m_{30}, m_{31}, m_{29}, m_6, n_{32}, and n_{11} represent the new skylines of the entire database. We can also notice that these tuples are not dominated by any other tuples in the database including the newly inserted tuples.

V. RESULTS AND DISCUSSION

To measure the efficiency and the performance of the proposed solution, IDSA, in processing skyline queries over a dynamic and incomplete database, various experiments with different parameter settings have been designed and developed. To the best of our knowledge, this research work is the first attempt that addresses the issue of processing skyline queries over dynamic incomplete databases. Thus, we compare our proposed solution with the most recent well-known strategies that are the closest to this research, namely: SCSA [42], SPQ [29], *Incoskyline* [40], SIDS [26], and *Iskyline* [24]. These skyline approaches considered in this paper are mainly proposed to handle the data incompleteness problem when processing skyline queries. All experiments have been conducted on a computer machine with the following specifications. Intel Core *i*3 1.6GHz processor with 3GB memory and Windows 7 32-bit platform. All algorithms have been implemented using the C# programming language.

All experiments for the previous existing algorithms considered in this work have been accomplished in the following manner. We first run SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline* approaches over the initial incomplete database to retrieve the skyline result. Then, these approaches are run again after the database state is changed with the insert operation in which a new set of tuples with incomplete data are added to the initial incomplete database to produce the new set of skylines. In contrast, for experimental purposes, our proposed solution works as follows. We first run our previous approach SCSA proposed in [42] over the initial incomplete database to return the skylines of the database. Running the SCSA approach over the initial incomplete database helps in preparing the skyline set of the initial incomplete database to be used in the subsequent steps. Then, the proposed approach IDSA is run over the newly inserted tuples to derive the new set of skylines. These results are compared to the results of SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*, based on both datasets, namely: synthetic and real datasets. It has proven in the literature that the skyline query process is a CPU exhaustive process [2], [14], [24], [26], [29], [40], [45], [46] thus, the experiments of this research work have considered two performance metrics which are the number of pairwise comparisons and the processing time. In all experiments, these two metrics are measured by varying the total number of dimensions including dimensions with and without missing values, the number of dimensions with missing values, and the total size of the database for both types of data sets (synthetic and real). Two different data distributions for the synthetic dataset were chosen and generated in the experiments. First, an independent synthetic data set in which values of one dimension are independent of the values of other dimensions. While the second type of synthetic data set is correlated in which the values of one dimension are correlated with the values of other dimensions. Besides, three different real-world data sets have been involved, namely: CoIL 2000 insurance company, NBA, and MovieLens datasets. These datasets are mostly used by a large number of previous works in the area of processing skyline queries in incomplete databases [2], [14], [24], [26], [29], [40], [45], [46]. In all experiments, we assume that greater values are preferable compared to smaller ones when retrieving the skylines. Table 2 describes the parameter settings for the synthetic and real data sets used to evaluate the proposed approach for handling skylines queries in incomplete databases.

TABLE 2. The Parameter Settings of the Synthetic and Real datasets.

Parameter Settings	Dataset Name				
	Synthetic		Real		
	Independent	Correlated	CoIL 2000 Insurance Company	NBA	MovieLens
No. of Dimensions	7	7	14	18	4
No. of Dimensions with Missing Values (d')	6	6	13	17	3
Incompleteness	20-35%				
Initial Dataset Size (KB)	300	300	200	120	1200
Newly Inserted Dataset Size (KB)	100, 200, 300, 400, 500, 600	100, 200, 300, 400, 500, 600	50, 100, 150, 200, 250, 300	40, 80, 120, 160, 200	400, 800, 1200, 1600, 2000
Changing Rate (%)	30-200	30-200	25-150	30-160	30-160

A. EXPERIMENT RESULTS

This section reports and discusses the result of the experiments for the IDSA solution in deriving skyline queries over a dynamic and incomplete database, in which the state of the database is changed due to the insert operations made towards the initial incomplete database. In this set of experiments, the focus is given on investigating the impact of insert new tuples with missing values into the database on the performance of the skyline algorithms when computing the skylines based on the most recent state of the database. Thus, the size of the dataset will be varying in a predefined range while the number of dimensions remains unchanged. The experiment concentrates on studying the impact of adding new tuples to the initial incomplete database on the number of pairwise comparisons that need to be performed and the processing time of the skyline query process. It should be noted that in this set of experiments we assume that the previous skyline approaches (*Iskyline*, SIDS, *Incoskyline,* SPQ, and SCSA) compare the existing skylines of the dataset with the newly inserted tuples to identify the new set of skylines. Nevertheless, our approach is developed to avoid the entire scanning of the updated dataset when recomputing the new set of skylines after the insert operation.

1) EFFECT OF DATASET SIZE

The skyline literature evidenced that one of the crucial factors that have a significant impact on the performance of the skyline techniques when identifying skylines is the data set size [2], [14], [24], [26], [29], [40], [45], [46]. Hence, this section illustrates the experimental results of our proposed approach, IDSA, and the previous approaches for both the

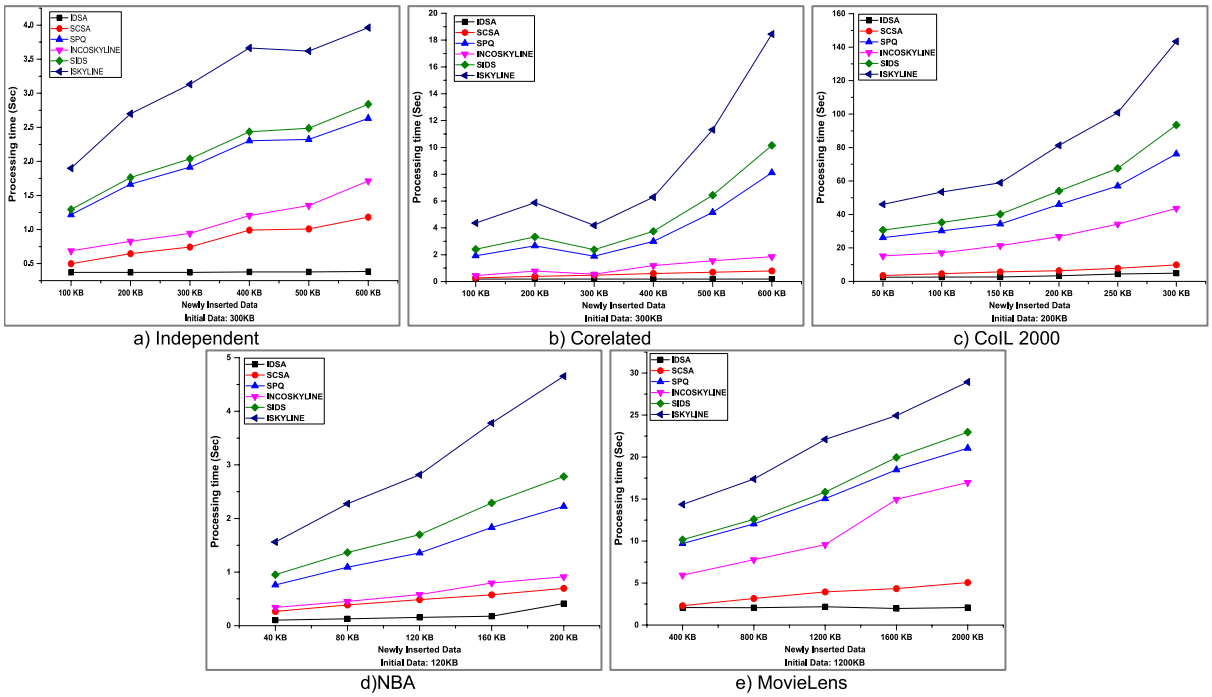


FIGURE 21. The effect of dataset size on the processing time.

synthetic and real data sets with respect to the processing time and the number of pairwise comparisons, by varying the size of the data set size through inserting new tuples with incomplete data to the initial incomplete database. **Figure 21(a)**, **21(b)**, **21(c)**, **21(d)**, and **21(e)** present the experiment results of the processing time consumed during the operation of identifying the new set of skylines on a dynamic incomplete database on independent, correlated, CoIL 2000 insurance company, NBA and MoviesLens data sets, respectively. For independent and correlated data sets, the initial size of the data set is 300KB and every tuple comprises seven dimensions including dimensions with missing values. The size of the data set increases gradually in the range of 100-600KB. **Figures 21(a)**, and **21(b)** depict the processing time of each approach considered in this work for both types of the synthetic data sets, namely: independent and correlated when deriving skyline queries over the dynamic incomplete database. From the figures, it can be noticed that IDSA outperforms the previous existing approaches (SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*). From the figures also it can be concluded that *Iskyline* is the worst compared to the other previous approaches (*Incoskyline*, SIDS, and SPQ) by consuming the longest processing time during the skyline query process. **Figure 21(c)** elucidates the results of the experiments for the CoIL 2000 insurance company dataset, in which the initial data present in the database is 200KB and the dataset size gradually increases in the range of 50-300KB. Besides, the number of dimensions has been fixed to 14 including the dimensions with missing data. Furthermore, **Figures 21(d)**, and **21(e)** elaborate the experiment results of both real-world datasets, namely: NBA and MovieLens. The initial size of the NBA and MovieLens datasets is set to be 120KB and 1200KB, respectively. Then, the sizes of these two datasets are varying in the range of 40-200KB for NBA and 400-2000KB for MovieLens. We set the number of dimensions for NBA to be 18, while the number of dimensions considered for the MovieLens dataset is set to 4 including dimensions with missing values. The experimental results of all datasets considered in this work proved the superiority of our proposed solution in all cases in comparison with the most recent approaches designed for processing skyline queries over incomplete data. From the results, we can also notice that the processing time consumed by our proposed strategy, IDSA is almost half of the processing time consumed by the second-best strategy, SCSA to identify the skylines in the dynamic incomplete database. Moreover, the results of the experiment also prove that our proposed approach is also superior and outperforms the other approaches (SQP, *Incoskyline*, SIDS, and *Iskyline*) in all cases. The main reason for this significant improvement in processing skyline queries on dynamic incomplete databases by IDSA is due to applying the data pruning technique on the newly inserted data to eliminate all dominated tuples before applying the skyline process. The idea of data pruning is very beneficial and helps in reducing the processing time of the skyline process on dynamic and incomplete databases significantly. In contrast, all other approaches assumed that the skyline process should be applied again over the entire database after the changes made towards the database through the insert and/or update operations. This is clearly shown in

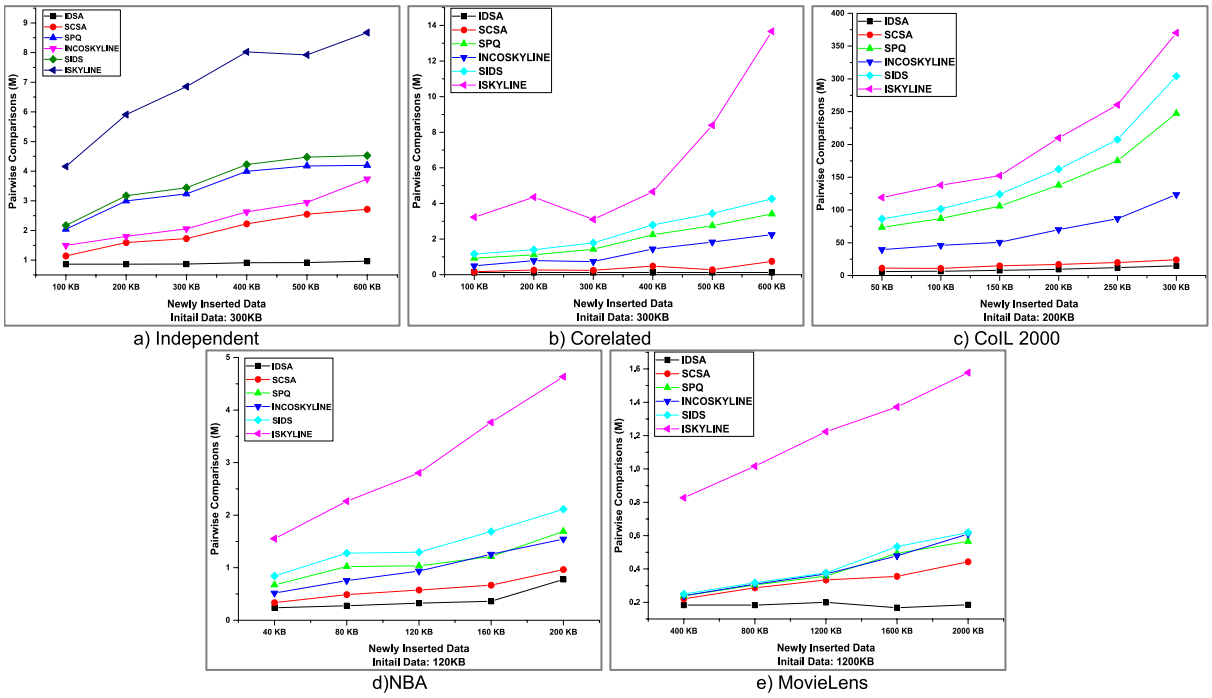


FIGURE 22. The effect of dataset size on the number of Pairwise comparisons.

all experiments conducted throughout this work. For instance, in the synthetic independent dataset, the skyline process is first applied on the initial incomplete database in which the size is set to be 300KB. Then, when a new set of tuples is added to the initial database which represents 100KB in this set of experiment, the IDSA attempts to apply the skyline process by only consider the newly added tuples. Doing so led to avoid many unnecessary exhaustive pairwise comparisons and reduces the number of tuples to be considered when identifying the new set of the skylines of the recently updated states of the database. The second reason that further improve the performance of IDSA when processing skyline queries on the dynamic and incomplete database is the idea of the concurrent execution of the clustering, grouping, identifying local skylines, and selecting superior local skylines phases in our proposed approach. We can conclude that the impact of inserting new tuples into the initial incomplete database and recomputing the skylines of the recently updated database using IDSA is insignificant.

Figure 22(a), **22(b)**, **22(c)**, **22(d)**, **22(e)**, and **22(f)** present the results of pairwise comparisons achieved by IDSA, SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*, based on independent and correlated, CoIL 2000 Insurance company, NBA and MovieLens datasets, respectively. The parameter setting of this set of experiment is similar to the previous set of experiments in terms of number of dimensions and dataset size as described in Table 2. **Figure 22(a)** and **22(b)** shows the total number of pairwise comparisons performed by each algorithm (IDSA, SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*) based on both synthetic dataset independent and correlated, respectively. For the synthetic dataset, the initial size of the data is set to 300KB and the size of the dataset increases continuously after adding a new set of tuples using the insert statement. The range of the increment in the size is varying between 100 KB to 600KB, while the number of dimensions remains unchanged with 6 dimensions including dimensions with missing values. Moreover, **Figure 22(c)** depicts the numbers of pairwise comparisons carried out by each algorithm (IDSA, SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*) for Coil 2000 Insurance company real-world dataset. In this set of experiment, we assume that the initial size of the dataset is 200KB and the size of the dataset gradually increases by adding a new set of tuples in the range of 50KB to 300 KB. Likewise, the number of dimensions considered is fixed to 14. Also, **Figure 22(d)** illustrate the result of the pairwise comparison performed by each skyline method considered in this work (IDSA, SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*) on the NBA real-world dataset. In this real-world dataset, the number of dimensions is fixed to 18 including dimensions with missing values, and the initial size of the dataset is 120KB. Then new tuples are added through the insert operation which results in a gradual increment in the size of the dataset between 40 KB to 200KB.

Figure 22(e) illustrates the number of pairwise comparisons performed by each algorithm (IDSA, SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*) during the skyline query process on the MovieLens real-world dataset. We assumed that the initial size of the dataset is 1200KB and a new set of tuples is added to the dataset which varies from 400 KB

to 2000KB, while the number of dimensions is fixed to 4 including dimensions with missing values.

From **Figure 22**, IDSA shows a steady performance which reflects that the number of new tuples added to the initial incomplete database has no significant impact on the performance of *IDSA*. When the number of added tuples increases, the number of pairwise comparisons performed increases, as reflected in the results of all the previous algorithms. The increment shown by IDSA is trivial and *IDSA* achieved better performance as compared to SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*, since it avoids unnecessary skyline computations by limiting the focus to only consider the skylines of the initial database and the newly added tuples while utilising the idea of generating min-dom and constructing lists and data pruning processes that leads to a significant reduction in the number of tuples to be considered during the skyline process. The figures also indicate that the other methods, SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline* perform pairwise comparisons over the entire database including the newly added tuples after the insert operation made towards the initial incomplete database. Doing so leads to unnecessary exhaustive pairwise comparisons due to including those dominated tuples that are not contributing towards the new set of skylines after the changes made towards the database. Last but not least, from the results of the experiments shown in **Figure 22** it is obvious that our proposed strategy is the best in comparison with other strategies designed for processing skyline queries over an incomplete database (SCSA, SPQ, *Incoskyline*, SIDS, and *Iskyline*) in terms of number of pairwise comparisons performed to identify the new skylines after the insert operation made towards the database. The main reasons behind this improvement accomplished by IDSA are due to applying the idea of *pruning* which produce a virtual tuple called *min-dom* that helps in eliminating many dominated tuples from the newly inserted tuples before executing the skyline process. Besides, the idea of selecting superior local skylines incorporated in IDSA is also very beneficial as it assists in discarding a large number of dominated tuples from further processing which in turn leads to a significant reduction in the number of pairwise comparisons.

VI. CONCLUSION

In this paper, a new skyline solution called IDSA is proposed which is capable of retrieving the skylines over a dynamic and incomplete database in which the database state changed due to the insert operation performed towards the initial incomplete database. The idea of the IDSA aims at avoiding examine the entire database contents when new tuples are added into the initial incomplete database. The IDSA relies on limiting the re-computation of the new set of skylines to only considering the skylines of the previous state of the initial incomplete database and the newly inserted tuples. The proposed approach adopts the idea of the *virtual tuple,* called *min-dom* that is very beneficial in simplifying the skyline process over the dynamic and incomplete database. The idea of *min-dom* assists in removing a significant portion of the dominated tuples from the newly inserted tuples before applying the skyline technique. This optimization leads to a significant reduction in the searching space when attempting to derive the new set of skylines. Furthermore, two optimization techniques (*pruning* and *selecting superior local skylines*) have been incorporated to expedite the skyline process by eliminating the dominated tuples in the early stages to avoid the unnecessary pairwise comparison. The results of the experiment demonstrate that the proposed solution is capable of preventing many unnecessary pairwise comparisons during skyline computations by concentrating only on the parts of the databases that are affected by the changes.

REFERENCES

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

[2] C. Y. Chan, H. V. Jagadish, K. L. Tan, A. K. H. Tung, and Z. Zhang, *On High Dimensional Skylines* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3896. Berlin, Germany: Springer, 2006, pp. 478–495.

[3] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2006, pp. 503–514.

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

[5] M. B. Swidan, A. A. Alwan, S. Turaev, and Y. Gulzar, "A model for processing skyline queries in crowd-sourced databases," *Artic. Indones. J. Electr. Eng. Comput. Sci.*, vol. 10, no. 2, pp. 798–806, 2018.

[6] M. B. Swidan, A. A. Alwan, S. Turaev, H. Ibrahim, A. Z. Abualkishik, and Y. Gulzar, "Skyline queries computation on crowdsourced-enabled incomplete database," *IEEE Access*, vol. 8, pp. 106660–106689, 2020.

[7] M. B. Swidan, A. A. Alwan, Y. Gulzar, and A. Z. Abualkishik, "An overview of query processing on crowdsourced databases," *Sci. J. King Faisal Univ.*, vol. 22, no. 1, pp. 5–12, 2021.

[8] X. Miao, Y. Gao, S. Guo, L. Chen, J. Yin, and Q. Li, "Answering skyline queries over incomplete data with crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1360–1374, Apr. 2021.

[9] X. Miao, Y. Gao, G. Chen, and T. Zhang, "K-dominant skyline queries on incomplete data," *Inf. Sci.*, vols. 367–368, pp. 990–1011, Nov. 2016.

[10] C. Kalyvas, T. Tzouramanis, and Y. Manolopoulos, "Processing skyline queries in temporal databases," in *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 893–899.

[11] Y. Gulzar, A. A. Alwan, N. Salleh, and I. F. Al-Shaikhli, "Skyline query processing for incomplete data in cloud environment," in *Proc. 6th Int. Conf. Comput. Informat. (ICOCI)*, 2017, pp. 567–576.

[12] Y. Gulzar, A. A. A. Aljuboori, N. Salleh, and I. F. Al Shaikhli, "Identifying skylines in cloud databases with incomplete data," *J. Inf. Commun. Technol.*, vol. 18, no. 1, pp. 19–34, Jan. 2019.

[13] M. L. Yiu and N. Mamoulis, "Efficient processing of top-k dominating queries on multi-dimensional data," in *Proc. 33rd Int. Conf. Very Large Data Bases. VLDB Endowment*, Vienna, Austria, 2007, pp. 483–494.

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

[15] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proc. 27th Int. Conf. Very Large Data Bases (VLDB)*, vol. 1, 2001, pp. 301–310.

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

[17] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in *Proc. 31st Int. Conf. Very Large Data Bases. VLDB Endowment*, Trondheim, Norway, 2005, pp. 229–240.

[18] I. Bartolini, P. Ciaccia, and M. Patella, "SaLSa: Computing the skyline without scanning the whole sky," in *Proc. 15th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, Arlington, VI, USA, 2006, pp. 405–414.

[19] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, San Diego, CA, USA, 2003, pp. 467–478.

[20] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *Proc. 28th Int. Conf. Very Large Data Bases. VLDB Endowment*, Hong Kong, 2002, pp. 275–286.
[21] K. C. K. Lee, W.-C. Lee, B. Zheng, H. Li, and Y. Tian, "Z-SKY: An efficient skyline query processing framework based on Z-order," *VLDB J.*, vol. 19, no. 3, pp. 333–362, Jun. 2010.
[22] S. Zhang, N. Mamoulis, and D. W. Cheung, "Scalable skyline computation using object-based space partitioning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 483–494.
[23] G. B. Dehaki, H. Ibrahim, F. Sidi, N. I. Udzir, A. A. Alwan, and Y. Gulzar, "Efficient computation of skyline queries over a dynamic and incomplete database," *IEEE Access*, vol. 8, pp. 141523–141546, 2020.
[24] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline query processing for incomplete data," in *Proc. IEEE 24th Int. Conf. Data Eng.*, Apr. 2008, pp. 556–565.
[25] Y. Gulzar, A. A. Alwan, N. Salleh, I. F. A. Shaikhli, and S. I. M. Alvi, "A framework for evaluating skyline queries over incomplete data," *Procedia Comput. Sci.*, vol. 94, pp. 191–198, Jan. 2016.
[26] R. Bharuka and P. S. Kumar, "Finding skylines for incomplete data," in *Proc. 24th Australas. Database Conf.*, vol. 137. Adelaide, SA, Australia: Australian Computer Society, 2013, pp. 109–117.
[27] R. Bharuka and P. S. Kumar, "Finding superior skyline points from incomplete data," in *Proc. 19th Int. Conf. Manage. Data*, 2013, pp. 35–44.
[28] Y. Gulzar, A. A. Alwan, N. Salleh, and I. F. Al Shaikhli, "Processing skyline queries in incomplete database: Issues, challenges and future trends," *J. Comput. Sci.*, vol. 13, no. 11, pp. 647–658, Nov. 2017.
[29] Y. Wang, Z. Shi, J. Wang, L. Sun, and B. Song, "Skyline preference query based on massive and incomplete dataset," *IEEE Access*, vol. 5, pp. 3183–3192, 2017.
[30] Y. Gulzar, A. A. Alwan, N. Salleh, and I. F. A. Shaikhli, "A model for skyline query processing in a partially complete database," *Adv. Sci. Lett.*, vol. 24, no. 2, pp. 1339–1343, Feb. 2018.
[31] Y. Gulzar, "Skyline query approaches in static and dynamic incomplete databases," Int. Islamic Univ. Malaysia, Selangor, Malaysia, Tech. Rep. tQA76.9D32G971S2018, 2018.
[32] K. Zhang, H. Gao, H. Wang, and J. Li, "ISSA: Efficient skyline computation for incomplete data," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2016, pp. 321–328.
[33] Y. Gulzar, A. A. Alwan, and S. Turaev, "Optimizing skyline query processing in incomplete data," *IEEE Access*, vol. 7, pp. 178121–178138, 2019.
[34] K. Zhang, H. Gao, X. Han, Z. Cai, and J. Li, "Probabilistic skyline on incomplete data," in *Proc. ACM Conf. Inf. Knowl. Manage.*, Nov. 2017, pp. 427–436.
[35] Y. Gulzar, A. A. Alwan, A. Z. Abualkishik, and A. Mehmood, "A model for computing skyline data items in cloud incomplete databases," *Procedia Comput. Sci.*, vol. 170, pp. 249–256, Jan. 2020.
[36] J. Lee and S.-W. Hwang, "Scalable skyline computation using a balanced pivot selection technique," *Inf. Syst.*, vol. 39, pp. 1–21, Jan. 2014.
[37] M. Shamsul Arefin, "Skyline sets queries for incomplete data," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 5, pp. 67–80, Oct. 2012.
[38] W.-T. Balke, U. Güntzer, and J. X. Zheng, "Efficient distributed skylining for Web information systems," in *Proc. Int. Conf. Extending Database Technol.*, 2004, pp. 256–273.
[39] X. Miao, Y. Gao, L. Chen, G. Chen, Q. Li, and T. Jiang, "On efficient k-skyband query processing over incomplete data," in *Proc. 18th Int. Conf. (DASFAA)*, Wuhan, China, Apr. 2013, pp. 424–439.
[40] A. A. Alwan, H. Ibrahim, N. I. Udzir, and F. Sidi, "An efficient approach for processing skyline queries in incomplete multidimensional database," *Arabian J. Sci. Eng.*, vol. 41, no. 8, pp. 2927–2943, Aug. 2016.
[41] Y. Zeng, K. Li, S. Yu, Y. Zhou, and K. Li, "Parallel and progressive approaches for skyline query over probabilistic incomplete database," *IEEE Access*, vol. 6, pp. 13289–13301, 2018.
[42] Y. Gulzar, A. A. Alwan, R. M. Abdullah, Q. Xin, and M. B. Swidan, "SCSA: Evaluating skyline queries in incomplete data," *Appl. Intell.*, vol. 49, no. 5, pp. 1636–1657, May 2019.
[43] K. Zhang, H. Gao, X. Han, Z. Cai, and J. Li, "Modeling and computing probabilistic skyline on incomplete data," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 7, pp. 1405–1418, Jul. 2020.
[44] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous processing of preference queries in data streams," in *Proc. 36th Int. Conf. Current Trends Theory Pract. Comput. Sci.*, Špindlerův Mlýn, Czech Republic, Jan. 2010, pp. 47–60.
[45] Y. Gulzar, A. A. Alwan, H. Ibrahim, and Q. Xin, "D-SKY: A framework for processing skyline queries in a dynamic and incomplete database," in *Proc. 20th Int. Conf. Inf. Integr. Web-Based Appl. Services (iiWAS)*, 2018, pp. 164–172.
[46] J. Lee, H. Im, and G.-W. You, "Optimizing skyline queries over incomplete data," *Inf. Sci.*, vols. 361–362, pp. 14–28, Sep. 2016.