Received 16 May 2022, accepted 14 June 2022, date of publication 22 June 2022, date of current version 27 June 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3185087

CIDS: An Efficient Algorithm for Processing Skyline Queries for Partially Complete Data in Cloud Environment

YONIS GULZAR[1] **AND ALI A. ALWAN**[2]
[1]Department of Management Information Systems, College of Business Administration, King Faisal University, Al-Ahsa 31982, Saudi Arabia
[2]School of Theoretical and Applied Science, Ramapo College of New Jersey, Mahwah, NJ 07430, USA

Corresponding authors: Yonis Gulzar (ygulzar@kfu.edu.sa) and Ali A. Alwan (aaljuboo@ramapo.edu)

This work was supported by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia, under Project GRANT64.

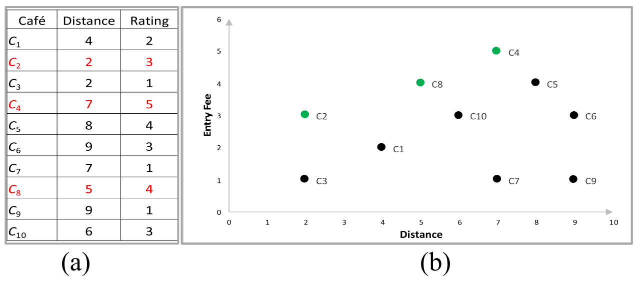
ABSTRACT From a set of existing tuples, a skyline operator retrieves only a subset, *superior tuples* that are of a person's interest and are non-dominant. Processing of queries using the skyline operator is an expensive and exhaustive task. It gets more complicated when skyline queries are applied on partially complete data and databases are distributed over different data centers. Incompleteness in data raises many issues such as compromise on *transitivity property* and the threat of *cyclic dominance* to occur within database. To overcome such issues this paper proposes an efficient algorithm called Cloud-based Incomplete Data Skyline algorithm (CIDS) for processing skyline queries over partially complete databases in cloud environment. The algorithm retrieves superior tuples with the aim of reducing domination tests between the tuples, decreasing processing time and reducing the huge amount of data flow from one data center to another. Several experiments have been conducted over different types of datasets, and results have proven that the proposed algorithm outplays the existing algorithms in terms of processing time, domination tests as well as the amount of data flow.

INDEX TERMS Cloud databases, distributed databases, incomplete databases, query processing, skyline queries.

I. INTRODUCTION

From a *multi*-dimensional dataset D, a skyline query returns a set of tuples S from D that is not being dominated by any other tuple(s) in D. A tuple t from D dominates another tuple u iff t is not worse than u in all dimensions and t is better than u in at least one dimension. Let us take an example of a café recommendation system. In which it is assumed that a person is looking for a café to have a coffee, priority is given to minimum *distance* from his residence and highest *rating* of a café. Figure 1a shows 10 cafés with the corresponding information i.e., distance (in kilometers) and rating. Figure 1b portrays the representation of café dataset in 2D space with the x-axis denoting distance and the y-axis denoting the rating. From figure 1b it can be noticed that café C_8 dominates C_{10}, C_7, and C_9, since C_8 has a better rating and shorter distance than C_{10}, C_7, and C_9. It can be also noticed that C_2 and C_3 are at the same distance however, the rating of C_2 is better than C_3.

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



Café	Distance	Rating
C_1	4	2
C_2	2	3
C_3	2	1
C_4	7	5
C_5	8	4
C_6	9	3
C_7	7	1
C_8	5	4
C_9	9	1
C_{10}	6	3

FIGURE 1. Example conventional of skyline query.

In that case, C_2 dominates C_3. Also, C_4 dominates C_5 and C_6 as the C_4 has a better rating and shorter distance than C_5 and C_6. After comparing C_2, C_8, and C_4 with each other it can be observed that none of these cafés are fully dominating each other. In that case, C_2, C_8, and C_4 are considered skyline tuples.

Skyline operator has been very popular among researchers in database systems for the past 2 decades. The '*skyline* *operator*' was introduced by Borzsony *et al.* [1]. Since then, it has been extensively explored by the database community [2]–[6]. Among preferences queries, the skyline query is considered one of the best queries when it comes to user preference. These queries have been intensively used in many databases such as multi-criteria decision making, decision support [7], [8], crowdsourcing databases [9]–[13], hotel recommender [14], restaurant finder [2], [14], temporal databases [15], and cloud databases [16]. Since the skyline operator was introduced, the researchers have been working on improving it, in terms of efficiency and accuracy. It has been implemented on both complete and incomplete databases. A lot of research has been done in improving the efficiency of skyline operators in complete databases. Many approaches have been proposed in the literature [4]–[8], [17], [18] focusing on minimize the searching space, reduce the domination tests and increase efficiency. The task is easier when it comes to identifying the skyline tuples (points) in complete databases. As all the dimension values are present in complete databases. Nevertheless, it becomes complicated when some dimension values are missing in the database especially in large multi-dimensional databases or cloud/distributed databases. The incompleteness of data does not only make the skyline query process complicated but has a negative impact on the output as well.

Due to the incompleteness of the data, numerous challenges have risen while processing skyline queries. Not only that but also made the process of identifying skyline points complex. The approaches proposed for identifying skyline points in complete databases have become irrelevant when it comes to partially complete/incomplete databases. As these approaches have been designed for a database(s) where all the dimensions of the tuples are comparable. When it comes to partially complete databases, there may be chances where some of the dimensions of two or more tuples are incomparable. That in turn leads to exhaustive unnecessary pairwise comparisons between the tuples. It is important to note that the incompleteness of data may lead to lose *transitivity property* of the skyline. If that happens then there are high chances of occurring *cyclic dominance* Problem [2], [19]–[21].

Losing *transitivity property* and occurrence of *cyclic dominance* is considered a huge challenge while computing skyline queries in partially complete databases. To have a better understanding of these two challenges let us reconsider the above-given example with some modifications. Let us assume that there are three dimensions now in the café database i.e., distance, rating, and price. In the third dimension, the price of a particular item to be sold in all cafés is considered. Suppose there are three cafés (C_1, C_2, and C_3) and the parameters of these tuples are C_1 (5, 4, #), C_2 (6, #, 17), and C_3 (#, 5, 20). The '#' represents the missing values of the dimensions. The preference to choose the café for a person is to have minimum distance (first dimension) from his residence, better rating (second dimension) of café and minimum price (third dimension) of a particular item. Considering this scenario and while comparing these tuples with each other. It is noticed that the café, C_1 is dominating café C_2 in the first dimension as C_1 is having a minimum distance than C_2 and the other two dimensions are incomparable due to missing values in the third dimension of C_1 and second dimension of C_2. It can be also noticed that C_2 dominates C_3 as the price of C_2 is minimum than C_3. According to the transitivity process if C_1 dominates C_2 and C_2 dominates C_3 then C_1 should dominate C_3. However, this is not the case here as it can be seen clearly that C_1 is not superior to C_3 in any of the dimensions. which states that the *transitivity property* does not hold in the current scenario. Additionally, it can be further noticed that it is C_3 which has superiority over C_1 in the second dimension (better rating). Which leads to the case of *cyclic dominance*. At the end of the pairwise comparison process, the results are obtained where no cafés have been identified as skyline(s). It is because all the cafés are dominated by one another. The formal definitions of transitivity and cyclic dominance are as follows:

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

Many approaches have been proposed [2], [22]–[25] for computing skyline queries in partially complete databases. These approaches aim to improve the efficiency, reduce the search space and the domination tests between the tuples while identifying the skyline queries.

The computation of skyline queries gets more complicated when it comes to partially complete databases where data is stored in different data centers and distributed remotely over a cloud environment. To Identify the skylines from cloud incomplete databases it is important to note that data should be transferred from all the involved data centers to the host data center where a query is submitted and will be processed, and global skylines will be generated. However, this process is considered impractical and expensive [26], [27]. It is because a lot of data is needed to be transferred between the data centers. It also leads to unwanted pairwise comparisons which in turn negatively impacts on the process of generating skyline queries. The approaches proposed for processing skyline queries for incomplete databases cannot be directly implemented on cloud incomplete databases as those approaches are designed for centralized databases. So, there is a need for an efficient algorithm for processing skyline queries in partially complete databases in cloud environment.

The following points summarize the contributions of this paper:

- The problem of retrieving the global skylines in a partially complete database distributed over the cloud is discussed and an explanation is provided why there is a need for an efficient approach.
- A comprehensive review of the leading studies has been conducted on skyline queries in database systems. Which covers the previous approaches designed for

complete, incomplete, and distributed databases. The review highlights the strengths and the weaknesses of each approach.

- An efficient algorithm called *Cloud-based Incomplete Data Skyline* (CIDS) is proposed for processing skyline queries in partially complete data over cloud environment which efficiently identifies global skylines from all data centers involved. CIDS comprises two phases towards determining the skylines of a partially complete cloud database.
- An innovative technique is designed and developed which classifies the tuples into different clusters based on the domination power of tuples.
- Two optimization techniques, filtration and optimization have been incorporated that help in reducing the domination tests between tuples. Filtration prunes the non-competent tuples before the skyline process begins. Whereas optimization pickup only the superior tuples from the local skylines identified from each cluster, which have a good chance to be part of skylines and eliminate the rest of the tuples.
- The effectiveness and the efficiency of the proposed algorithm are evaluated through experiments using both synthetic and real datasets. Experiments show the effectiveness of the proposed algorithm.

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

II. RELATED WORK

Incorporating skyline operator by Borzsonay *et al.* [1] in 2001 was a huge development in database systems. Skyline operator has evolved query processing to the next level. Since 2001 the skyline operator has been used in many databases such as complete [1], incomplete [2], [9], [28], distributed [10], [13], [14] as well as uncertain databases [30]. Borzsonay *et al.* [1] proposed two algorithms BNL and D&C. BNL uses the *sorting* approach whereas D&C uses an approach of *partitioning* the dataset. After that, numerous algorithms have been proposed in the literature for different databases using above mentioned approaches. The aim of all those algorithms is to minimize the searching space, reduce the domination tests, reduce the execution time of processing skyline queries. In this section, the existing algorithms designed for incomplete databases and distributed incomplete databases are reviewed and investigated.

After Borzsonay *et al.* [1], it was Khalefa *et al.* [2] who incorporated skyline operator in incomplete databases by proposing *ISkyline* algorithm. The *ISkyline* Algorithm uses a partitioning approach by dividing the dataset into different sections called *Buckets* based on the bitmap representation of the tuples. They used two optimization techniques (*virtual points* and *shadow skylines*). To overcome the issue identified in the Bucket [2] algorithm. They formulated a *virtual* point (tuple) within each bucket ($B_i - B_n$) and used this virtual point to compare with the other buckets (B_i with $B_{i+1} - B_n$). This is done to avoid comparing all the tuples of B_i with the rest of the tuples present in $B_{i+1} - B_n$. The aim behind this is to avoid all unwanted pairwise comparisons between non-dominating tuples and eliminate such tuples. A set of tuples called *shadow Skylines* is created to dominate some tuples across the buckets. There is a drawback of the *ISkyline* algorithm though. As it can be seen the order of the tuples within the bucket matters while getting compared with virtual point (v). It is because there might be a case where a tuple p, when compared with v, gets dominated. However, there is a tuple q down the line which will be dominated by p but not v. But p has been eliminated already so there is no chance that q will be removed and will be represented as candidate skyline whereas it is clear that q should not be in the list.

Arefin *et al.* [31] proposed RSSSQ algorithm for processing skyline queries in incomplete data. In their approach, they implemented a new technique by replacing missing values with a certain value that is larger than the domain value to avoid the threat of losing transitivity property. Miao *et al.* [32] proposed three different approaches to process skylines in incomplete data (baseline, virtual point (VP) and k-iSkyband (kISB)). The baseline algorithm has avoided creating different buckets so has to go through a lot of domination tests to produce results. Whereas the VP algorithm overcomes such issues. Finally, kISB improves the VP algorithm by reducing a large number of domination tests and eliminating redundant data storage. In 2013 Bharuka and Kumar proposed a new algorithm (SIDS) [33] to improve the *ISkyline* algorithm. SIDS used a sorting approach, unlike *ISkyline* and RSSSQ. In SIDS, the dataset is being sorted column-wise in descending order. The sorted values are replaced with the tuple ID within the sorted dataset. Then skyline process starts by comparing the first tuple present p at $x_i . y_i$ with other tuples present in the same column ($x_{i+1} . y_j - x_n . y_j$). While comparing, if p dominates q then q gets eliminated immediately and if otherwise then p gets eliminated at the end of the iteration. It is because p is believed to have more power to dominate other tuples in the list as it is on the top of the list. After the first iteration the next tuple present at $x_i . y_{j+1}$ is selected for processes and the same approach is being used. During this process, a counter is set to keep a count of how many times a tuple is being processed. If the processing count is equal to the number of non-missing dimensions of p. Then p is moved to ResultSet. Once all the items are being processed at least once then comparing process stops and the tuples present in ResultSet are considered as final skylines. SIDS somehow has shown significant improvement in skyline execution while comparing with *ISkyline.* However, there are some issues with SIDS such as there might be a case where a tuple t gets eliminated whereas tuple v is kept just because p and v are non-comparable. Another issue is that in SIDS there as chances of indulging to having cyclic dominance as there is not any technique to have domination tests between tuples with the same bitmap representation. Comparing all the tuples with each other without using any optimization technique ends up having a lot of unwanted domination tests between tuples. Which makes SIDS exhaustive and time-consuming.

There are other numerous approaches found in literature, proposed for skyline queries computation in incomplete databases SOBA [23], ISSA[34], IncoSkyline [35], SCSA [36], OIS [37]. All these approaches use the same technique of partitioning the dataset based on their bitmap representation. However, in ISSA the algorithm gets executed in two phases whereas in the first phase they use segmentation which helps to remove some dominated tuples at the early stage. In the second phase, they use aggregate function (SUM) to sum the non-missing values of the tuples and sort them in non-ascending order, to further eliminate some dominated tuples. In SOBA, after partitioning (based on the same namespace) they incorporated an optimization technique in which tuples present in each bucket are sorted in non-descending order to identify local skylines. however, to identify skylines locally, SOBA compares tuples across the buckets, unlike ISSA. There is the same issue(s) found in ISSA as of SIDS because ISSA uses the same approach (sorting) as SIDS. In SOBA there are chances of having false positive and false negative tuples in each bucket because these tuples have been compared across the buckets only.

In IncoSkyline [35], the authors divide the tuples into clusters based on bitmap representation and then further divide those clusters into groups based on the similar highest value(s). Two tuples are being created the *upper bound(ub)* and *lower bound(lb)* from each cluster, this *ub* and *lb* are used to stop the comparison process between each group when the *ub* is less than or equal to *lb*. This process helps to eliminate non-dominant tuples from each group. After identifying the local skyline from each group. A virtual tuple is created from each cluster and that tuple is used to compare with the local skylines of other clusters in order to identify the final skylines. Although, IncoSkyline has improved the Iskyline algorithm, but IncoSkyline and Iskyline both use the same technique of creating virtual tuples which have a negative impact on the accuracy of the results as discussed earlier in the section.

Wang *et al.* [25] has proposed Skyline Preference Query, the SPQ approach to processing skyline queries in incomplete data. SPQ and SIDS have a similar procedure of processing. In SPQ the process starts by dividing the dataset into two subsets based on the preference given by the user. The first subset is of high preference and another subset is of low preference. For the first subset, the SIDS approach is used to find local skylines whereas for the second subset divide and conquer approach is implemented to identify local skylines. To retrieve the final skylines the local skylines of each subset are compared. From the results, it can be seen that SPQ has outperformed SIDS, but SQP is still lacking in terms of holding transitivity property. Moreover, there is not any pruning done before identifying local skylines which do not help in avoiding unwanted pairwise comparisons between the tuples. A new technique using adaptive two-level grids called TLG has been proposed in [38]. They used a technique in which non-skyline points were pruned from each region as early as possible. The final skylines were then computed by considering all the regions. A MRBIG algorithm [39] has been proposed in another study to identify the top-k queries in incomplete big data. It identifies the top-k queries by executing parallel operations on multiple nodes (partitioned dataset).

An optimized approach was proposed (OIS) [37], where the dataset is first divided into different subsets based on the bitmap representation. Then each dimension present in each subset is sorted in non-ascending order. After that, the optimization technique is used to select only those tuples from each subset that has the highest values(s) (up to 2 levels) based on the users' preference. Then only these tuples are compared with each other to identify the local skylines of $G_1 - G_n$. To identify the final skylines, the local skylines of G_1 are compared with the local skylines of G_{1++} to G_n. This process continues until all the subsets are compared with one another. The remaining tuples in each subset are not dominated by any other tuple and are considered as final skylines. Whereas in SCSA [36] algorithm the dataset is first sorted in non-ascending order for each dimension. Then clusters are formed based on the domination power which was calculated by counting the presence of tuples in each dimension until all the tuples have been read at least once. After that clusters are further divided into subgroups based on the bitmap representation of tuples. Then skyline process starts by identifying local skylines from each group within the cluster and then candidate skylines are identified from each cluster by comparing local skylines of each group. Finally, candidate skylines are compared with each other to identify final skylines. SCSA has used two optimization techniques to eliminate non-dominating tuples at an early stage. Nevertheless, OIS and SCSA are optimized and are performing well to identify skylines in incomplete data, but they are not meant for distributed databases. It will take a lot of unwanted domination tests to identify skylines in cloud incomplete databases. It is because the data present in the different data centers needs to be transferred to the *query submitted* data center. It is impractical to directly use such approaches (SIDS, Incoskyline, ISSA, SPQ, OIS or SCSA) on cloud incomplete databases.

Alwan *et al.* [26] has proposed Jincoskyline for processing skyline queries in distributed databases. In this approach, the author has assumed that the data present in the databases is partitioned vertically and stored over the distributed databases. The Jincoskyline process starts by identifying skylines (local) in each distributed database and then only those skylines are transferred to the *query submitted* database where all those skylines are compared with each other to identify the final skylines. During the process of identifying local skylines, the database is divided into clusters based on bitmap representation, then further partitioned into groups based on similar values. Then local skylines are identified for each group. To identify the skylines of each cluster a virtual tuple, *k-dom* is created and compared with other tuples present in other clusters without comparing all the tuples of a cluster with the tuples present in another cluster (s). This approach has opted similar technique as of Incoskyline where both approaches are used *k-dom* to identify local skylines of each cluster. Both approaches lack accuracy because there are high chances of false negative and false positive cases due to *k-dom*.

To the best of our knowledge, the latest work on the issue of processing skyline queries in incomplete data over cloud environment is ICS (incomplete-data Cloud Skyline) [27], [29]. In ICS the author assumed that the dataset is divided horizontally and is stored in many data centers at remote places over the cloud. ICS and Jincoskyline use the same approach of identifying the skylines of each data center separately then merging the local skyline to find final skylines. However, ICS uses a sorting approach while identifying skylines. in ICS a pruning technique has been implemented to eliminate unwanted tuples that do not have the potential to be part of skylines. Even though ICS is performing well, however, there are high chances of occurring cyclic dominance as there is not any method used to control non-comparable tuples from comparing. Which in turn can produce skylines which contain false negative and false positive tuples.

From the work(s) reviewed in this section, it can be concluded that most of the previous works assume that the database is centralized, and data is stored in a single database. Implementing these approaches directly on cloud incomplete databases is impractical because of the nature of those approaches. If done so, these approaches will turn into expensive ones due to a high number of pairwise comparisons and a high volume of data transfer from one data center to another. And the approaches proposed for cloud incomplete databases are not efficient enough in terms of providing accurate results with minimum domination tests and efficient processing time. Hence, there is a need for an efficient algorithm to identify skylines in cloud incomplete databases with an aim of reducing the amount of data transfer from one data center to another, minimizing domination tests between tuples and reducing processing time.

III. DEFINITIONS

In this section, some important annotations, and definitions are presented which are related to the skyline queries in cloud database with partially complete data. it is must to explain these annotations and definitions for our proposed algorithm. Table 1 summarizes the symbols and notations used throughout the paper. These terms are further explained below.

The proposed algorithm is developed in the context of a relational database with missing data. Let us assume that there is a relation in a database D, which is denoted as $R(d_1, d_2, d_3, d_4, \ldots, d_n)$ where R is the name of the relation with multiple dimensions d_n. $d_1, d_2, \ldots, d_n$ is a set of dimensions in R.

Definition 1 (Incomplete Database): given a database D $(R_1, R_2, R_3, R_4, \ldots, R_n)$, where R_i is a relation denoted by $R_i(d_1, d_2, d_3, d_4, \ldots, d_n)$. D is said to be *incomplete* if and only if it contains at least a tuple t_i with missing values in one or more dimensions d_k; otherwise, it is *complete.*

Definition 2 (Dominance on Incomplete Database): Given two tuples t_i and $t_j \in D$ incomplete database with d dimensions, p_i dominates p_j (denoted by $t_i \succ t_j$) if (and only if) the following three conditions hold:

- The values of d_k and $d_l \in d$ for t_i and t_j must be non-missing and
- $\forall d_k \in d, t_i.\, d_k \geq t_j.\, d_k$ and
- $\exists\, d_l, d, t_i.\, d_l > t_j.\, d_l$

Definition 3 (Skylines on Partially Complete Database: Select a tuple t_i from the set of D incomplete database if (and only if) t_i is as good as t_j (where $i \neq j$) in all common non-missing dimensions and strictly better than t_j in at least one common non-missing dimension. S is used to denote the set of skyline tuples on an incomplete database, $S = (t_i \forall t_i, t_j \in D, t_i \succ t_j)$.

Definition 4 – Comparable: Let the tuple t_i and $t_j \in R$, t_i and t_j are comparable (denoted by $t_i \varepsilon\ t_j$) if and only if they have no missing values in at least one identical dimension; otherwise, t_i is incomparable to t_j (denoted by $t_i \varepsilon\!\!/\ t_j$).

Definition 5 – Cloud Database: given a set of databases D $(DB_1, DB_2, \ldots, DB_n)$, where DB_1 is a database denoted by DB $(R_1, R_2, \ldots, R_n)$, where R represent a database relation belong to DB_i, D is a cloud database if the databases are deployed over different data centers located on different sites.

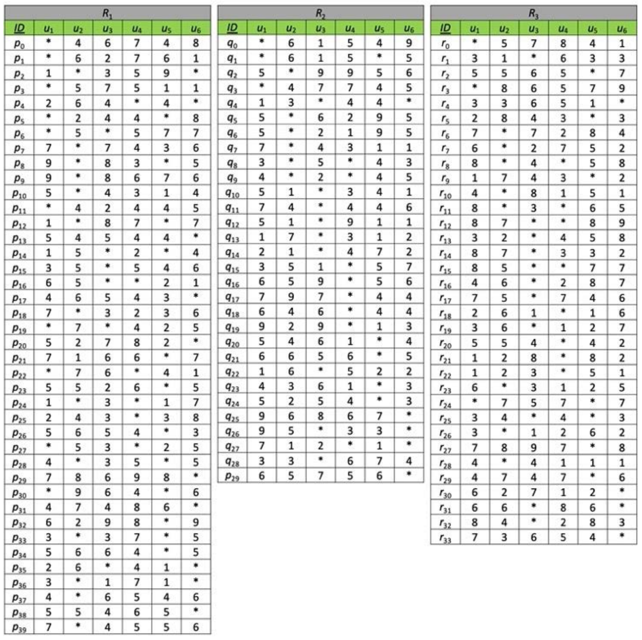
TABLE 1. Symbols and descriptions.

Notation	**Definition**
D	Dimension
R_i	Relation
DC	Data Center
t	Tuple
v	View
TPR	Total number of tuples in Relation R_i
dp	Domination Power
th	Threshold set by User
ADPC	Array of domination power count
C	Cluster
G	Group
BP	Bitmap Representation
NTC	Number of Tuples in Cluster C_i
ND	Number of Dimensions

IV. CIDS ALGORITHM

This section presents the proposed algorithm called *Cloud-based Incomplete Data Skyline algorithm* (CIDS), which aims at processing skyline queries for partially complete data in cloud environment. Where data is stored in different data centers at different locations. The proposed algorithm emphasizes on reducing unwanted computations between the tuples to identify the skyline points, reducing the amount of data transfer from one data center to another, reducing the processing time to identify global skylines. The CIDS algorithm composes of two main phases namely: i) Identifying local skylines from individual data centers, and ii) Transferring of local skylines and retrieving global skylines.

To have a better understanding of how the CIDS algorithm operates, an example of an incomplete database containing the data of Movie ratings given by 6 different users, has been used as shown in Figure 2. Our proposed algorithm can generate skylines from numerous data centers however, for simplicity it is assumed that there are three data centers (DC_1, DC_2, & DC_3) involved which contain similar type data and each data center contains one relation (DC_1. R_1, DC_2. R_2, & DC_3. R_3). It is also assumed that the relation R_1 contains 40 tuples, R_2 has 30 tuples and R_3 has 34 tuples. For simplicity and without losing the generality, it is assumed that the given skyline query aims at identifying tuples with the highest values in each dimension of a relation(s). The $*$ symbol is used to denote the missing value(s) in the relation dimensions.



R_1						
ID	u_1	u_2	u_3	u_4	u_5	u_6
p_0	*	4	6	7	4	8
p_1	*	6	2	7	6	1
p_2	1	*	3	5	9	*
p_3	*	5	7	5	1	1
p_4	2	6	4	*	4	*
p_5	*	2	4	4	*	8
p_6	*	5	*	5	7	7
p_7	7	*	7	4	3	6
p_8	9	*	8	3	*	5
p_9	9	*	8	6	7	6
p_{10}	5	*	4	3	1	4
p_{11}	*	4	2	4	4	5
p_{12}	1	*	8	7	*	7
p_{13}	5	4	5	4	4	*
p_{14}	1	5	*	2	*	4
p_{15}	3	5	*	5	4	6
p_{16}	6	5	*	*	2	1
p_{17}	4	6	5	4	3	*
p_{18}	7	*	3	2	3	6
p_{19}	*	7	*	4	2	5
p_{20}	5	2	7	8	2	*
p_{21}	7	1	6	6	*	7
p_{22}	*	7	6	*	4	1
p_{23}	5	5	2	6	*	5
p_{24}	1	*	3	*	1	7
p_{25}	2	4	3	*	3	8
p_{26}	5	6	5	4	*	3
p_{27}	*	5	3	*	2	5
p_{28}	4	*	3	5	*	5
p_{29}	7	8	6	9	8	*
p_{30}	*	9	6	4	*	6
p_{31}	4	7	4	8	6	*
p_{32}	6	2	9	8	*	9
p_{33}	3	*	3	7	*	5
p_{34}	5	6	6	4	*	5
p_{35}	2	6	*	4	1	*
p_{36}	3	*	1	7	1	*
p_{37}	4	*	6	5	4	6
p_{38}	5	5	4	6	5	*
p_{39}	7	*	4	5	5	6

R_2						
ID	u_1	u_2	u_3	u_4	u_5	u_6
q_0	*	6	1	5	4	9
q_1	*	6	1	5	*	5
q_2	5	*	9	9	5	6
q_3	*	4	7	7	4	5
q_4	1	3	*	4	4	*
q_5	5	*	6	2	9	5
q_6	5	*	2	1	9	5
q_7	7	*	4	3	1	1
q_8	3	*	5	*	4	3
q_9	4	*	2	*	4	5
q_{10}	5	1	*	3	4	1
q_{11}	7	4	*	4	4	6
q_{12}	5	1	*	9	1	1
q_{13}	1	7	*	3	1	2
q_{14}	2	1	*	4	7	2
q_{15}	3	5	1	*	5	7
q_{16}	6	5	9	*	5	6
q_{17}	7	9	7	*	4	4
q_{18}	6	4	6	*	4	4
q_{19}	9	2	9	*	1	3
q_{20}	5	4	6	1	*	4
q_{21}	6	6	5	6	*	5
q_{22}	1	6	*	5	2	2
q_{23}	4	3	6	1	*	3
q_{24}	5	2	5	4	*	3
q_{25}	9	6	8	6	7	*
q_{26}	9	5	*	3	3	*
q_{27}	7	1	2	*	1	*
q_{28}	3	3	*	6	7	4
p_{29}	6	5	7	5	6	*

R_3						
ID	u_1	u_2	u_3	u_4	u_5	u_6
r_0	*	5	7	8	4	1
r_1	3	1	*	6	3	3
r_2	5	5	6	5	*	7
r_3	*	8	6	5	7	9
r_4	3	3	6	5	1	*
r_5	2	8	4	3	*	3
r_6	7	*	7	2	8	4
r_7	6	*	2	7	5	2
r_8	8	*	4	*	5	8
r_9	1	7	4	3	*	2
r_{10}	4	*	8	1	5	1
r_{11}	8	*	3	*	6	5
r_{12}	8	7	*	*	8	9
r_{13}	3	2	*	4	5	8
r_{14}	8	7	*	3	3	2
r_{15}	8	5	*	*	7	7
r_{16}	4	6	*	2	8	7
r_{17}	7	5	*	7	4	6
r_{18}	2	6	1	*	1	6
r_{19}	3	6	*	1	2	7
r_{20}	5	5	4	*	4	2
r_{21}	1	2	8	*	8	2
r_{22}	1	2	3	*	5	1
r_{23}	6	*	3	1	2	5
r_{24}	*	7	5	7	*	7
r_{25}	3	4	*	4	*	3
r_{26}	3	*	1	2	6	2
r_{27}	7	8	9	7	*	8
r_{28}	4	*	4	1	1	1
r_{29}	4	7	4	7	*	6
r_{30}	6	2	7	1	2	*
r_{31}	6	6	*	8	6	*
r_{32}	8	4	*	2	8	3
r_{33}	7	3	6	5	4	*

FIGURE 2. **Movie rating database example stored over cloud.**

A. IDENTIFY LOCAL SKYLINES FROM INDIVIDUAL DATA CENTERS

The first phase, identifying local skylines from the individual data center, attempts at identifying local skylines from each relation stored in different data centers over the cloud. The aim of this phase is to eliminate all those tuples which do not have the potential to be part of the final skyline set before transferring the data to the query submitted data center. This phase of the CIDS algorithm aims at reducing the amount of data transfer from one data center to another by propagating only superior candidate tuples in the next phase. To make sure that only candidate tuples are be transferred to the next phase which have high chances to be part of final skylines. The data within this phase goes through some optimization techniques and preprocessing. This phase completes in five stages which are elaborated in detail in the following subsections.

Stage 1: Sorting and Filtering: in this stage, the tuples are sorted in non-ascending order for each dimension based on the values present in each dimension of R and a *view*, v is created the same as R, which stores the sorted information. It is important to note that the dimensions of v store only *ids* of the tuples of R. After the sorting of all dimensions present in *view*, v are scanned in a *round-robin* fashion from dimension $v.d_i$ - $v.d_n$ to count (*domination power*) the presence of each tuple in v. The scanning process continues until all the tuples are scanned at least once. A user-defined threshold (*th*) is set to filter out such tuples whose domination power is lower than *th* and those tuples are eliminated from further processing. It is certain that such eliminated tuples do not have any impact on the skyline and will be dominated by other tuples if compared. Thus, eliminating such tuples from further processing help in reducing many domination tests which in turn improves the execution time.

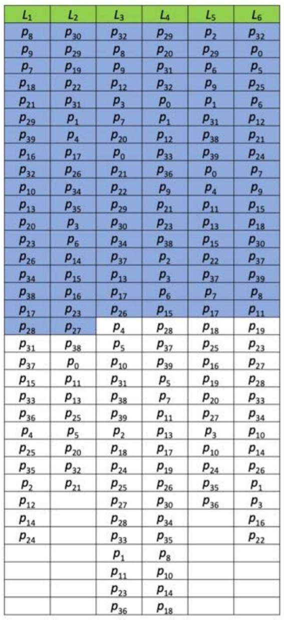
The process of this stage starts by sorting the relation R_i present in each data center DC_i simultaneously and parallelly. This process is the same for all data centers (DC_i - DC_n) so keeping that thing in mind and for simplicity, without losing the generality the execution process is explained with the help of R_1 of DC_1. All relations (in Figure 2) are sorted in non-ascending order for each dimension (d_1 to d_6) based on non-missing values. A view v_i is created for all relations present in each data center, having the same dimensions as R_i. Figure 3 shows the *view*, v_1 of sorted tuples according to R_1. The dimension l_1 of v_1 contains the *ids* of all the tuples which have values present in dimension d_1 of R_1 in non-ascending order. It is important to note that the tuples with missing data are ignored and their *ids'* are not mentioned in the sorted *view*.

Once the sorting process completes, another process called *scanning* begins by scanning the dimension of v (l_1 to l_6) in a round-robin fashion to calculate the *domination power* (*dp*) of tuples present in each relation. The formula of calculating the *dp* is as follows:

$$dp = \sum_{k=1}^{l} t_i, \text{ iff } t_i.k \succ t_j.k$$

The reason for calculating the domination power during the scanning process is to find out the domination power of each tuple, which shows the probability of dominating other tuples. The larger the domination power of a tuple, the more potential it has to dominate other tuples.

As per our running database exams, the scanning process works as follows: from the v_1 the scanning process begins with reading p_8 from l_1 and sets it *dp* to 1. Then read $p_{30}, p_{32}, p_{29}, p_2$ from l_2, l_3, l_4, and l_5 respectively, and sets their *dp* value equal to 1. In $l_6 p_{32}$ gets repeated so its *dp* values get incremented by 1 and is equal to 2 now. The process of scanning the tuples continues until a point where all the tuples are scanned (read) at least once. According to our example,

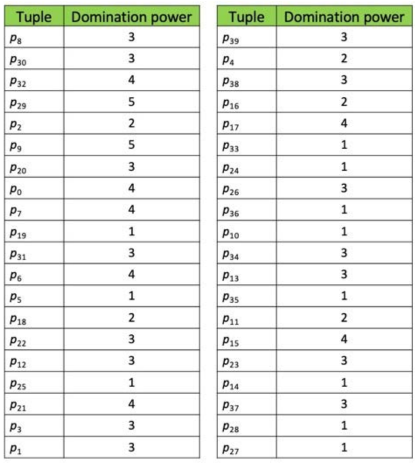


L_1	L_2	L_3	L_4	L_5	L_6
p_8	p_{30}	p_{32}	p_{29}	p_2	p_{32}
p_9	p_{29}	p_8	p_{20}	p_{29}	p_0
p_7	p_{19}	p_9	p_{31}	p_6	p_5
p_{18}	p_{22}	p_{12}	p_{32}	p_9	p_{25}
p_{21}	p_{31}	p_3	p_0	p_1	p_6
p_{29}	p_1	p_7	p_1	p_{31}	p_{12}
p_{39}	p_4	p_{20}	p_{12}	p_{38}	p_{21}
p_{16}	p_{17}	p_0	p_{33}	p_{39}	p_{24}
p_{32}	p_{26}	p_{21}	p_{36}	p_0	p_7
p_{10}	p_{34}	p_{22}	p_9	p_4	p_9
p_{13}	p_{35}	p_{29}	p_{21}	p_{11}	p_{15}
p_{20}	p_3	p_{30}	p_{23}	p_{13}	p_{18}
p_{23}	p_6	p_{34}	p_{38}	p_{15}	p_{30}
p_{26}	p_{14}	p_{37}	p_2	p_{22}	p_{37}
p_{34}	p_{15}	p_{13}	p_3	p_{37}	p_{39}
p_{38}	p_{16}	p_{17}	p_6	p_7	p_8
p_{17}	p_{23}	p_{26}	p_{15}	p_{17}	p_{11}
p_{28}	p_{27}	p_4	p_{28}	p_{18}	p_{19}
p_{31}	p_{38}	p_5	p_{37}	p_{25}	p_{23}
p_{37}	p_0	p_{10}	p_{39}	p_{16}	p_{27}
p_{15}	p_{11}	p_{31}	p_5	p_{19}	p_{28}
p_{33}	p_{13}	p_{38}	p_7	p_{20}	p_{33}
p_{36}	p_{25}	p_{39}	p_{11}	p_{27}	p_{34}
p_4	p_5	p_2	p_{13}	p_3	p_{10}
p_{25}	p_{20}	p_{18}	p_{17}	p_{10}	p_{14}
p_{35}	p_{32}	p_{24}	p_{19}	p_{24}	p_{26}
p_2	p_{21}	p_{25}	p_{26}	p_{35}	p_1
p_{12}		p_{27}	p_{30}	p_{36}	p_3
p_{14}		p_{28}	p_{34}		p_{16}
p_{24}		p_{33}	p_{35}		p_{22}
		p_1	p_8		
		p_{11}	p_{10}		
		p_{23}	p_{14}		
		p_{36}	p_{18}		

FIGURE 3. **Sorted view of R_1.**

p_{27} is the last tuple to be read in l_2 and the process of scanning terminates there.

The domination power (*dp*) of each tuple and their *ids* is stored in a 2D array as shown in Figure 4.



Tuple	Domination power	Tuple	Domination power
p_8	3	p_{39}	3
p_{30}	3	p_4	2
p_{32}	4	p_{38}	3
p_{29}	5	p_{16}	2
p_2	2	p_{17}	4
p_9	5	p_{33}	1
p_{20}	3	p_{24}	1
p_0	4	p_{26}	3
p_7	4	p_{36}	1
p_{19}	1	p_{10}	1
p_{31}	3	p_{34}	3
p_6	4	p_{13}	3
p_5	1	p_{35}	1
p_{18}	2	p_{11}	2
p_{22}	3	p_{15}	4
p_{12}	3	p_{23}	3
p_{25}	1	p_{14}	1
p_{21}	4	p_{37}	3
p_3	3	p_{28}	1
p_1	3	p_{27}	1

FIGURE 4. **Domination power of tuples.**

As we know skyline queries are a type of preference queries and it is up to the preference of the *user*, what kind of criteria s/he sets for his/her skyline query results. For that reason, a user-defined threshold (*th*) is set to filter out some tuples that do not have any potential to be part of the skyline even before the skyline process begins. Eliminating such tuples happens in the filtering process. So, all those tuples whose *dp* value is less than *th* will be eliminated from the further process. The main aim of implementing such an optimization technique is to reduce the number of domination tests which in turn reduces the execution time. The formula of filtering is as follows:

$$ft - list = \{\forall t_i, \text{ iff } dp \geq th\}$$

where ft-list is the remaining tuples after filtration

According to our running example if we set the threshold, *th* as 2, then all those tuples whose *dp* is less than 2 will be eliminated from further processing. Such as p_{19}, p_5, p_{25}, p_{33}, p_{24}, p_{36}, p_{10}, p_{35}, p_{14}, p_{28}, and p_{27} have been successfully eliminated from further processing. With the help of the sorting and filtering process, a sizable number of tuples have been eliminated from further processing. Out of 40, 11 tuples have been selected and removed, which is around 27% of tuples that are eliminated from further processes. Such technique plays a vital role in reducing domination tests between tuples. It is important to note that the eliminated tuples if compared with those tuples which have higher *dp*, will be dominated (Tuples having *dp* value 5 or 4).

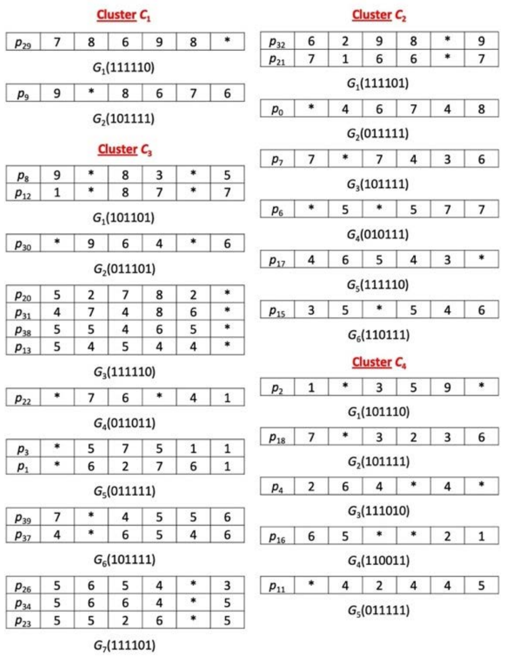
Calculating the domination power of each tuple has a huge impact on the overall skyline process. It does not only help to filter out dominated tuples but also helps in creating different clusters based on *dp*, which is explained in the next stage. It is also important to highlight that our proposed approach is capable of eliminating those tuples as well which have *dp* as 2 or 3 depending upon the number of the dimensions present in the dataset. After many successful experiments, it can be claimed that the proposed algorithm can eliminate all those tuples with *dp* value 2 without having any impact on the final skylines. So, the *th* can be changed according to users' preferences.

Stage 2: Clustering and Grouping: the aim of this stage is to further simplify the procedure of executing the skyline process by dividing the tuples based on domination power. So, all those tuples that have similar *dp* will be grouped in one cluster. This is attained by scanning the domination power list (except eliminated tuples). Therefore, numerous numbers of clusters ($C_\text{i} - C_n$) will be created. Even after applying the clustering method, there is still one problem remaining which is incompatibility. i.e., tuples may be incomparable due to missing values. That also raises the issue of occurring *cyclic dominance*. To avoid such a problem from occurring, another partitioning technique is being implemented to divide clusters into groups based on the *bitmap representation* of tuples. Applying of divide and conquer technique has been proven to be an effective way of processing skyline queries in database systems [1], [2], [23], [35]. In bitmap representation tuples are represented in 1's and 0's. If a value of a dimension is available, then it is represented as 1 otherwise 0. For instance, if a tuple t has 4 dimensions t (6, *, 8, *), * represents missing values, then the bitmap representation of t is (1010). Therefore, according to this all the tuples present in each cluster will be divided into different groups $G_i.C_n - C_i.G_m$). the formula of creating groups are as follows:

$$C_i.G_m = \{\forall t_i, \text{ iff } t_i.bitmap = t_j.bitmap\}$$

Grouping does not just divide the tuples based on bitmap representation but helps in reducing domination tests between tuples by eliminating non-dominant tuples within groups rather than comparing those with all the tuples present in the whole cluster.

According to our running example, the remaining tuples present in the domination power list are divided into clusters. Four clusters are created (C_1, C_2, C_3, and C_4) based on the domination power. It is important to note that clusters will be formed based on decreasing order of *dp*. So, C_1 contains all those tuples whose domination power is 5, C_2, C_3, and C_4 contain tuples whose domination power is 4, 3, and 2 respectively. Cluster C_1 contains only 2 tuples (p_{29}, p_9), whereas cluster C_2, C_3, and C_4 contain 7, 15, and 5 tuples respectively. These clusters are further divided into groups based on the bitmap representation of tuples. So according to the different bitmap representations, cluster C_1 has two groups containing one tuple in each so C_1. G_1 contains p_{29} whereas C_1. G_2 contains p_9. Likewise, clusters C_2, C_3 and C_4 contain 6, 7 and 5 groups respectively as shown in Figure 5.


Cluster C_1

p_{29}	7	8	6	9	8	*

G_1(111110)

p_9	9	*	8	6	7	6

G_2(101111)

Cluster C_3

p_8	9	*	8	3	*	5
p_{12}	1	*	8	7	*	7

G_1(101101)

p_{30}	*	9	6	4	*	6

G_2(011101)

p_{20}	5	2	7	8	2	*
p_{31}	4	7	4	8	6	*
p_{38}	5	5	4	6	5	*
p_{13}	5	4	5	4	4	*

G_3(111110)

p_{22}	*	7	6	*	4	1

G_4(011011)

p_3	*	5	7	5	1	1
p_1	*	6	2	7	6	1

G_5(011111)

p_{39}	7	*	4	5	5	6
p_{37}	4	*	6	5	4	6

G_6(101111)

p_{26}	5	6	5	4	*	3
p_{34}	5	6	6	4	*	5
p_{23}	5	5	2	6	*	5

G_7(111101)

Cluster C_2

p_{32}	6	2	9	8	*	9
p_{21}	7	1	6	6	*	7

G_1(111101)

p_0	*	4	6	7	4	8

G_2(011111)

p_7	7	*	7	4	3	6

G_3(101111)

p_6	*	5	*	5	7	7

G_4(010111)

p_{17}	4	6	5	4	3	*

G_5(111110)

p_{15}	3	5	*	5	4	6

G_6(110111)

Cluster C_4

p_2	1	*	3	5	9	*

G_1(101110)

p_{18}	7	*	3	2	3	6

G_2(101111)

p_4	2	6	4	*	4	*

G_3(111010)

p_{16}	6	5	*	*	2	1

G_4(110011)

p_{11}	*	4	2	4	4	5

G_5(011111)

FIGURE 5. List of clusters and their groups.

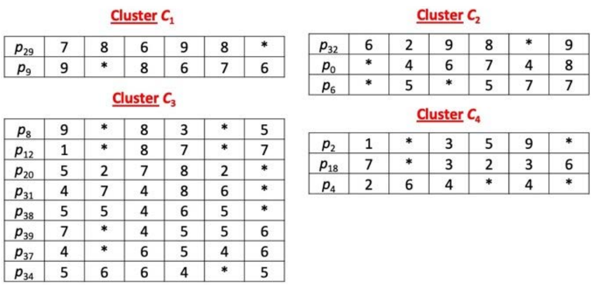
Stage 3: Finding local Skylines: in this stage, local skylines of each cluster are identified. The dominated tuples will be removed from further processing. The aim of finding local skylines is to reduce overall pairwise comparisons to identify final skylines. To identify local skylines of each cluster (C_i – C_n), CIDS finds local skylines of each group first. As tuples within the groups are having the same bitmap representation, so it is easier to compare them with one another. Comparing tuples having the same bitmap representation helps in holding transitivity property, which in turn stops cyclic dominance from occurring. It is important to note that all the local skylines of each cluster are identified parallelly and simultaneously. The idea behind this is to expedite the execution process and reduce the processing time.

The process begins by identifying local skylines of all the groups present in each cluster (C_i. G_n–C_i. G_m, C_{i++}. G_n–C_{i++}. G_m, . . . , C_n. G_n–C_n. G_m). the tuple t_i from group C_i. G_n is selected for processing and is compared with the next tuple t_j of C_i. G_n. if t_i dominates t_j then t_j gets eliminated immediately from further processing and if otherwise then t_i gets eliminated at the end of the iteration. It is due to the fact that t_i being at top of the list has more potential to dominate other tuples. So, this process of comparing continues until all the tuples (not dominated) within a group are compared with each other. the remaining tuples in group C_i. G_n are considered the local skyline of group C_i. G_n. The same procedure is applied to identify the local skylines of all the groups within each cluster. Furthermore, to identify local skylines of each cluster (C_i – C_n) the tuples of each group are compared with the tuples of another group within the cluster. So, t_i of C_i. G_n is compared with t_i of C_i. G_{n++}- C_i. G_m and the same technique is used to eliminate dominated tuples as mentioned above. In this way, all the dominated tuples will be removed from further processing and clusters contain only non-dominated tuples.

According to our running database example, the local skylines of each cluster are identified by comparing the tuples of each group with one another than compared with across the groups within the same cluster. From cluster C_1 there are two groups (C_1. G_1 and C_1. G_2) and each group contains only one tuple p_{29} and p_9. So, p_{29} and p_9 are considered as local skylines of groups C_1. G_1 and C_1. G_2. To identify local skylines of cluster C_1 tuple p_{29} is compared with p_9 and during comparison it can be noticed that p_{29} is getting dominated by p_9 at dimension d_1, at dimension d_2 they are incomparable, at dimension d_3 again p_{29} gets dominated by p_9. However, it is p_{29} which dominates p_9 at dimension d_4. Therefore, the process of comparison stops here as both tuples do not dominate each other completely so are considered as local skylines of cluster C_1 as shown in Figure 6.

There are 6 groups in cluster C_2. In group C_2. G_1 there are two tuples p_{32} and p_{21}, p_{32} is selected for processing and is compared with p_{21}. After comparison, it can be noticed that neither p_{29} dominates p_{21} nor p_{21} dominates p_{32} completely. In that case, both p_{32} and p_{21} are considered as local skylines of group C_2. G_1. rest of the groups have one tuple each, so they are considered as local skylines of those groups. After comparing tuples of each group with other groups such as tuples of C_2. G_1 are compared with C_2. G_2, C_2. G_3, C_2. G_4, C_2. G_5, and C_2. G_6. After the comparison process, it can be noticed that only p_{32}, p_0, and p_6 are the local skylines of cluster C_2, the rest of the tuples (p_{21}, p_7, p_{17}, and p_{15}) are dominated and eliminated from further processing. Likewise, local skylines of clusters C_3, C_4 are identified as shown in Figure 6.

With the help of this stage 13 out of 29 remaining tuples have been successfully eliminated as those were dominated by other tuples. That is around 45% of tuples eliminated from further processing. Incorporating this technique has a huge impact on reducing pairwise comparisons in the overall process of identification of final skylines.



Cluster C_1

p_{29}	7	8	6	9	8	*
p_9	9	*	8	6	7	6

Cluster C_2

p_{32}	6	2	9	8	*	9
p_0	*	4	6	7	4	8
p_6	*	5	*	5	7	7

Cluster C_3

p_8	9	*	8	3	*	5
p_{12}	1	*	8	7	*	7
p_{20}	5	2	7	8	2	*
p_{31}	4	7	4	8	6	*
p_{38}	5	5	4	6	5	*
p_{39}	7	*	4	5	5	6
p_{37}	4	*	6	5	4	6
p_{34}	5	6	6	4	*	5

Cluster C_4

p_2	1	*	3	5	9	*
p_{18}	7	*	3	2	3	6
p_4	2	6	4	*	4	*

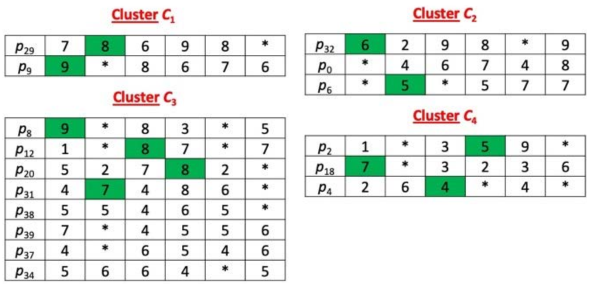
FIGURE 6. Local skylines of clusters.

Stage 4: Choosing Superior local Skylines: in this stage, another optimization technique is used to filter out such tuples that do not have the potential to be part of the final skyline of relation R_i. This is attained by eliminating some of the non-competent tuples from each cluster way before comparing them with other tuples across the clusters. Doing so helps in further decreasing the pairwise comparison between the tuples across the clusters, which in turn accelerates the skyline process. The process begins by identifying the highest value in each dimension then selecting only those tuples which have the highest value in any of the dimensions. The rest of the tuples that do not have the highest value in any of their dimensions are eliminated. To generalize it, the following formula shows the process of choosing the superior skyline of each cluster.

$$SLS = \{\forall t_i \; of \; C_i, \; t_i \max(t_i.d_j)\} \quad t_i \; where \; 0 \geq d_j \geq u$$

It is important to note it is claimed that after many experimentations the eliminated tuples from this stage if compared across the clusters will be dominated by other tuples present in the cluster with higher *dp* values. Therefore, these tuples can be successfully eliminated without having any impact on the final skylines. Also, it is important to highlight that this process (choosing superior local skylines) runs simultaneously and parallelly for each cluster.

According to our running example, from cluster C_1 the p_9 has the highest value in dimension d_1 so p_9 is selected. Likewise in dimension d_2, it is p_{29} which has the highest value so is selected. The process of choosing a superior skyline for cluster C_1 terminates here as this cluster contains only two tuples and both tuples are being selected. From Cluster C_2 it is p_{32} which has the highest value in dimensions d_1, d_3, d_4, and d_6 and tuple p_6 have the highest value in dimension d_2, and d_5. Therefore, only these two tuples are being selected from C_2 and p_0 is eliminated from further processing. Likewise, from cluster $C_3 p_{38}$, p_{39}, p_{37}, and p_{34} are eliminated from further processing as these tuples do not contain any highest value in any of their dimensions. From cluster C_4 all the tuples are selected for further processing as all the tuples (p_2, p_{18}, p_4) contain the highest value in at least one of the dimensions as shown in Figure 7.



Cluster C_1

p_{29}	7	**8**	6	9	8	*
p_9	**9**	*	8	6	7	6

Cluster C_2

p_{32}	**6**	2	9	8	*	9
p_0	*	4	6	7	4	8
p_6	*	**5**	*	5	7	7

Cluster C_3

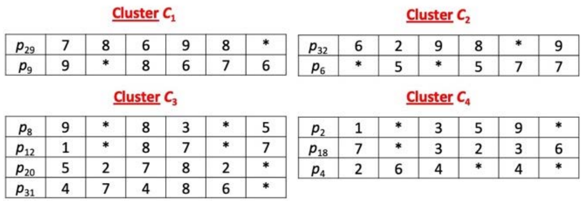
p_8	**9**	*	8	3	*	5
p_{12}	1	*	**8**	7	*	7
p_{20}	5	2	7	**8**	2	*
p_{31}	4	**7**	4	8	6	*
p_{38}	5	5	4	6	5	*
p_{39}	7	*	4	5	5	6
p_{37}	4	*	6	5	4	6
p_{34}	5	6	6	4	*	5

Cluster C_4

p_2	1	*	3	**5**	9	*
p_{18}	**7**	*	3	2	3	6
p_4	2	6	**4**	*	4	*

FIGURE 7. Highlighted superior skylines of each cluster.

The process of choosing superior tuples terminates by eliminating 5 tuples in total which do not have any potential to be part of the final skyline. Out of 16, 5 tuples are removed which is around 31% reduction/elimination of tuples. Which in turn reduces the number of domination tests between the tuples in coming processes. Figure 8 shows the superior skylines of each cluster.



Cluster C_1

p_{29}	7	8	6	9	8	*
p_9	9	*	8	6	7	6

Cluster C_2

p_{32}	6	2	9	8	*	9
p_6	*	5	*	5	7	7

Cluster C_3

p_8	9	*	8	3	*	5
p_{12}	1	*	8	7	*	7
p_{20}	5	2	7	8	2	*
p_{31}	4	7	4	8	6	*

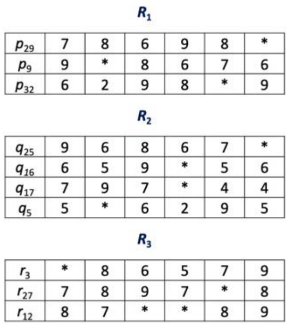
Cluster C_4

p_2	1	*	3	5	9	*
p_{18}	7	*	3	2	3	6
p_4	2	6	4	*	4	*

FIGURE 8. Superior skylines of each cluster.

Stage 5: Retrieval of Final Skylines: This is the final stage of identifying skylines from each data center. In this stage skylines of each data center will be retrieved. In other words, the skylines of all the involved relations from each data center will be retrieved. the process begins by comparing the remaining tuples present in each cluster with other clusters. So, the tuples of C_i will be compared with tuples of $C_{i++} - C_n$ and in the next iteration tuples of C_{i++} will be compared with the tuples of $C_{(i++)+1} - C_n$. this process continues unless all the tuples present in all the clusters are compared with tuples across the clusters.

According to our running example, tuples of C_1 are compared with tuples of C_2, C_3, and C_4. During the first iteration, tuple p_{29} is selected and compared with the tuples of C_2, C_3 and C_4. During this iteration, p_{29} dominates p_6, p_{31}, p_{18} and p_4. These dominated tuples are eliminated immediately from further processing. In the next iteration, p_9 is selected for processing and compared with the remaining tuples C_2, C_3, and C_4. During this iteration, p_9 dominates only p_8 from cluster C_3 an p_8 is eliminated. Now remaining tuples of C_2 are compared with C_3 and C_4. In this iteration, p_{32} is selected for processing and is compared with p_{12}, p_{20}, and p_2. It can be noticed that all these tuples are dominated by p_{32} and are eliminated. This ends the process of comparing the tuples across the clusters because there is not any tuple left in C_3 and C_4. At the end of the process, the remaining tuples (p_{29}, p_9, and p_{32}) are considered the final skylines of relation R_1. Likewise, the entire process of phase 1 (identify local skylines from individual data centers) is applied to all the relations of all the involved clusters. Figure 9 shows the skylines of relation R_1, R_2, and R_3.



R_1

p_{29}	7	8	6	9	8	*
p_9	9	*	8	6	7	6
p_{32}	6	2	9	8	*	9

R_2

q_{25}	9	6	8	6	7	*
q_{16}	6	5	9	*	5	6
q_{17}	7	9	7	*	4	4
q_5	5	*	6	2	9	5

R_3

r_3	*	8	6	5	7	9
r_{27}	7	8	9	7	*	8
r_{12}	8	7	*	*	8	9

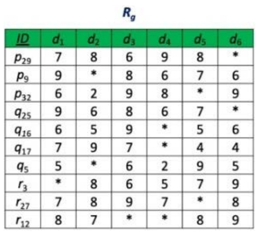
FIGURE 9. **Skylines of relation R_1, R_2, and R_3.**

B. TRANSFERING OF LOCAL SKYLINES AND RETRIEVING GLOBAL SKYLINES

This is the second phase of the CIDS algorithm, in which global skylines are identified from all relations present in different data centers and stored over cloud environment. The aim of this phase is to reduce the amount of data transfer from one data center to another and further eliminate the non-potential tuples that do not have any chance to be part of global skylines. This is achieved by transferring only the skylines of each relation of all data centers to the *query-submitted* data center, rather than transferring all the tuples present in each relation of all data centers to the *query-submitted* data center. Doing so helps in reducing a large amount of data transfer from one data center to another over the network, which in turn reduces the network cost as well. After transferring the skylines of each relation, these skylines are combined into a new relation, R_g. R_g contains only skylines of relation $R_1, R_2, \ldots,$ and R_n. After that global skyline are identified by executing all the stages of phase one all over again (except using *th*). It is done to further reduce the unwanted domination tests by eliminating non-potential tuples before identifying global skylines. It is also important to note that doing so ensures that at the end there is no false negative, or false-positive tuple present in global skylines. At the end of the second phase the remaining tuples in R_g are considered as global skylines of DC_1. R_1, DC_2. $R_2, \ldots,$ and DC_n. R_n.

According to our running example, the skylines of R_2, and R_3 are transferred to *query-submitted* data center DC_1. A new relation R_g is created which contains the skylines of R_1, R_2, and R_3(merged into R_g) as shown in Figure 10.

After combing the skylines of individual relations into one, all the stages of the phase are being repeated and at the

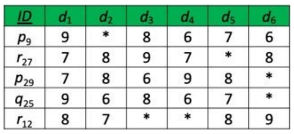


R_g

ID	d_1	d_2	d_3	d_4	d_5	d_6
p_{29}	7	8	6	9	8	*
p_9	9	*	8	6	7	6
p_{32}	6	2	9	8	*	9
q_{25}	9	6	8	6	7	*
q_{16}	6	5	9	*	5	6
q_{17}	7	9	7	*	4	4
q_5	5	*	6	2	9	5
r_3	*	8	6	5	7	9
r_{27}	7	8	9	7	*	8
r_{12}	8	7	*	*	8	9

FIGURE 10. **Combined skylines of all relations after join operation.**

end, the global skylines are identified which are not dominated by any tuple present in any relation of any data center. Figure 11 shows the global skylines of relations R_1, R_2, and R_3, present in data center DC_1, DC_2, and DC_3 over the cloud environment.



ID	d_1	d_2	d_3	d_4	d_5	d_6
p_9	9	*	8	6	7	6
r_{27}	7	8	9	7	*	8
p_{29}	7	8	6	9	8	*
q_{25}	9	6	8	6	7	*
r_{12}	8	7	*	*	8	9

FIGURE 11. **Global skylines.**

At the end of the second phase, CIDS has successfully identified global skylines of all involved data centers. It is important to highlight that incorporating of two optimization techniques have proven to be very efficient and effective in terms of reducing domination tests as well as removing of some non-potential tuples way before applying the skyline technique. Creating of clusters (and groups within the clusters) of relation(s) of all involved data centers not only help in reducing the domination tests between tuples but also expedite the skyline process. Which helps in reducing the execution time of CIDS. Transferring of only skylines of each relation has played a major role in reducing the amount of data transfer over the network, which in turn reduced the network cost.

CIDS algorithm takes partially complete dataset/relations stored in different data centers over cloud and outputs *GlobalSkylines* – the final skylines of entire data centers involved.

The algorithm starts by sorting the tuples present in d_i - d_n of R_i present at DC_i in non-ascending order and stored the ids of sorted tuples present in a *v*, view, which is identical to R_i in structure (step 1 – 3). In step 4 a variable, *TPR* is initialized to number of total tuples in R_i. The view, *v* is scanned in round-robin fashion from $l_i - l_n$ to scan the tuple *ids*. Initially the domination power of all tuples is set to zero. If tuple *id* is read first time, then it is stored in an array, *ADPC* along with its *dp* value. Every time a tuple is scanned its *dp* value is incremented by 1 and updated in *ADPC* (step 5 – 11). The scanning process stops when all the tuples of R_i are read at least once (step 11- 12). At the end of the 13th step the array, *ADPC* contains all the tuples of R_i and their *dp* values. In step 14 a *threshold* variable, *th* is initialized to the number defined number. All the tuples whose *dp* value is less than *th* are removed from *ADPC* (step 15 – 19). At the end of the

19th step with the help of this pruning technique many unwanted tuples have been eliminated. Now remaining tuples present in *ADPC* of R_i are partitioned into different clusters based on their domination power. These clusters are further divided into different groups based on their *bitmap representation* (step 20 – 35). After creating groups (C_i. G_i – C_i. G_m, C_{i++}. G_i – C_{i++}. G_m, …, C_n. G_i – C_n. G_m) within each cluster of R_i, now local skylines of each group are identified by comparing the first tuple, p present in C_i. G_i with the next tuple, q present in C_i. G_i (step 36 – 39). If p dominates q then q is eliminated from group C_i. G_i immediately (step 40). If q dominates p then then a *flag* is set *true* (step 41 – 43). In order to compare p with rest of the tuples in group C_i. G_i steps from 38 to 44 are being repeated. At the end of the iteration p is eliminated from C_i. G_i (step 45 – 47). To compare all the tuples present in group C_i. G_i with one another steps from 36 to 48 are being repeated. At the end of the 48th step all the groups present in all the clusters of R_i contain only tuples that are not being dominated by any tuple present in the same group. The local skylines of group G_i of cluster C_i are compared with the local skylines of group G_{i++} of C_i. This process continues until all the local skylines of group G_i are compared with G_{i++}– G_n of C_i (step 50 to 52). To identify local skylines of $C_i - C_n$ steps from 49 to 53 are being repeated. At the end of the step 53 all the clusters of relation R_i– R_n of data centers DC_i– DC_n contain only those tuples that are not dominated by any other tuple present within the same cluster.

Now another optimization technique is being implemented called "*choosing superior local skylines*" to select only those tuples within each cluster of R_i that are superior to other. The process begins by scanning each cluster $C_i - C_n$ dimension wise. the first dimension d_i of C_i is selected and all those tuples having highest value within d_i of C_i are being marked (step 56). To mark all those tuples who got highest value in each dimension ($d_i - d_n$) step 55 to 57 are being repeated. To identify superior tuples within each cluster and remove all those tuples which do not have highest value present in any one of the dimensions steps from 54 to 59 are getting repeated. In order to identify skylines of each relation ($R_i - R_n$) present in data centers $DC_i - DC_n$, the tuples present in cluster $C_i - C_n$ of relation R_i are being compared with one another. The process starts by choosing tuple of cluster C_i and then compared with the tuples of cluster $C_{i+1} - C_n$ and then remaining tuples in C_{i+1} is compared with the remaining tuples in in cluster $C_{i+2} - C_n$. To compare all the remaining tuples, present in each cluster with one another step 60 to 64 are being repeated.

At the end of the step 64 each relation ($R_i - R_n$) present in data centers $DC_i - DC_n$ contains only skylines which are not dominated by any other tuple present within same relation. In order to identify global skylines for all the data centers involved, the skyline of each relation ($R_i - R_n$) are combined into one relation, R_g(step 65). Then the step from 1-13 and 20 to 64 are repeated. It is important to note that to identify global skylines the *filtering* technique is not repeated (step 14 to 19). At the end of the step 66 the remaining tuples present in relation R_g are considered as global skylines (step 67). Global skylines are tuples that are not being dominated by any other tuple present in any relation of any data center involved, stored over cloud environment. It is also important to note that CIDS is run simultaneously and parallelly for all the relations of involved data centers, so are stages from *clustering and grouping* to *choosing superior local skylines* within each relation.

1) TIME COMPLEXITY ANALYSIS OF CIDS ALGORITHM

In this section, we present and analyze the time complexity analysis of the proposed algorithm, CIDS. To make the complexity analysis easier to understand, each phase of the algorithm is analyzed separately. As quicksort is the most efficient sorting algorithm, it has been used in our proposed algorithm to sort the tuples in relation R_i **(steps 1-3), SORT**. Thus, the running time of the sorting algorithm takes $O(|R_i| \times \log |R_i|)$ time, i.e.,

$$SORT\left(|R_i|, d_j\right) = O(|R_i| \times \log |R_i|)$$

Then for all dimensions $1 \leq d_j \leq d$ in R_i, the sorting takes $O(d \times |R_i| \times \log |R_i|)$ time, i.e.,

$$SORT\,(d, |R_i|) = O(d \times |R_i| \times \log |R_i|)$$

For all relations R_i in *DB*,

$$SORT\,(n, d) = \sum_{i=1}^{|R|} O(d \times |R_i| \times \log |R_i|)$$

The complexity analysis of the second phase, which counts the dominance power, *dp*, for each tuple t_j in relation R_i in the proposed algorithm **(steps 5-13)**, is as follows.

Let $COUNT(k, n)$ indicates the running time of steps 5-13. Based on the complexity analysis of the previous phase (*SORT*), we can assume that for each L_i in View, $|L_i| = O(n)$ where $n = |R_i|$. Since $|d| = k$, we have $|\ View\ | = k$. To compute the *dp* value of each tuple t_j present in view, v, loop scans every L_i in v. Thus, $COUNT(k, n) = k \times O(n) = O(k \times n)$. Since *dp_count* may contain all tuples of R_i, therefore, $|dp\text{-}count| = O(n)$.

Next, we examine the time complexity for the third phase in the proposed algorithm (**steps 20 – 27**). Let $|R_i|$ denotes the size of the database relation R_i, and let $|d_i|$ denotes the number of dimensions in R_i. Since $R_i = \{d_1, d_2, \ldots, d_n\}$ and each tuple is d-dimensional, $|R_i| = n$ and $|d_i| = d$. Similarly, we denote the number of clusters C in R_i and size of C_i (the number of tuples in C_i) by $|C|$ and $|C_i|$, respectively. Since the clusters are created according to the corresponding domination power *dp* of the tuple t_i, the number of clusters is as follows.

$$1 \leq |C| \leq \max(dp) - th$$

Next, the complexity analysis of the (**steps 28-35**) which denote the process of creating groups within each cluster based on different bitmap representations of the tuples. As the bitmap representations of a tuple t_i and a group G_i are order of d, each bitmap comparison takes a constant time. The total number of bitmap comparisons, BC, depends on the number tuples within each cluster. Thus,

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

where q is a positive constant number > 0

The construction of each group G_j takes a constant time, since it is initially an empty list, and the total number of the constructions, CG, is

$$1 \leq CG\,(d) \leq 2^d$$

The insertion of a tuple t_i into a group G_j takes $\mathrm{O}\,(d)$ time (a constant time). The total number of insertions, INS, is an order of n that is

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

Let *COUNT(k, n)* denote the running time of steps **(5-13)**. Then,

$$COUNT\,(k, n) = BC\,(k, n) + CG\,(d) + INS\,(n)$$

In the best case:

$$COUNT_{best}\,(k, n) = \mathrm{O}\,(n) + \mathrm{O}\,(1) + \mathrm{O}\,(n) = \mathrm{O}\,(n)$$

In the worst case:

$$COUNT_{worst}\,(k, n) = \mathrm{O}\left(n \times 2^d\right) + \mathrm{O}\left(2^d\right) + \mathrm{O}\,(n)$$
$$= mathrmO\left(n \times 2^d\right)$$

Thus,

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

Moreover, the time complexity analysis of the next phase described in **steps (36-48)** is illustrated as follows.

For simplicity and without losing generality, we consider the case in which $\left|C_i.G_j\right| = \mathrm{O}\,(n)$. if $\left|C_i.G_j\right| = \mathrm{O}\,(1)$, then all the operations in steps **36 – 48** also take $\mathrm{O}\,(1)$ time.

First, the total number of the assignments of the tuples to t_i is $\left|C_i.G_j\right|$ which means

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

Second, the total number of the assignments of the tuples to t_j is

$$ASSIGNt_j\,(n) = \sum_{i=1}^{|C_i.G_j|-1} \sum_{j=i+1}^{|C_i.G_j|} 1 = \sum_{i=1}^{|C_i.G_j|} \left(\left|C_i.G_j\right| - i\right)$$
$$= \mathrm{O}\left(\left|C_i.G_j\right|^2\right) = \mathrm{O}\left(n^2\right)$$

Similarly, each comparison of t_i with t_j takes $\mathrm{O}\,(d)$ time, and the total number of comparisons is

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

The removal of dominated $t_i s$ and t_js may take at most $\mathrm{O}\,(n)$ time. Thus, the total running time of the comparison process, *COMP* described in steps **(36-48)** is

$$COMP\,(k, n) = ASSIGNt_i\,(n) + ASSIGNt_j\,(n)$$
$$+ COMPt_it_j\,(k, n) = \mathrm{O}\left(k \times n^2\right)$$

Or simply,

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

As we assumed earlier, the number of clusters C in R_i and the total number of dimensions in R_i are denoted by $|C|$ and $|d|$, respectively. Since R_i contains all the constructed clusters, therefore $|C| = \mathrm{O}(n)$ where $n = |C|$. Likewise, and $|d| = \mathrm{O}(p)$, where $p = |d|$. The marking of tuples with highest value in d_i of C_i takes a constant time, therefore, the total number of the marked tuples is:

$1 \leq C\,(d) \leq t$, where t = total number of tuples in C_i

The deletion of a tuple t_i from a C_i takes $\mathrm{O}(t)$ time. Thus, the time complexity of identifying the superior cluster skylines as given in steps (54-59), SCS is

$$SCS = \mathrm{O}\,(C) + \mathrm{O}\,(p) + \mathrm{O}\,(1) + \mathrm{O}\,(t)$$
$$= \mathrm{O}\,(C) + \mathrm{O}\,(p) + \mathrm{O}(t)$$

CIDS Algorithm: Cloud-Based Incomplete Data Skylin
Input: *d-dimensional* partially complete dataset $D = (R_i - R_n)$ present in different Data Centers ($\mathrm{CD_i}$– CD_n)
Output: Global Skylines

```
1+ foreach Ri ∈ DCi do

1. foreach $d_i$ of $R_i \in DC_i$ do
2. sort $R_i$ in non-ascending order and store sorted id of $t_i$. $d_i$ in $v_i$
3. end
4. set $TPR$ = total number of tuples in $R_i$
5. foreach $t_i$ of $v_i \in DC_i$ do
6.    set dp of $t_i = +1$
7.    if $t_i$ not read before then
8.       Set $tuple\_count = +1$
9.       Store $t_i$ in $ADPC$
10.   end
11.   update dp of $t_i$ in dp_count
11.   if $TPR = tuple\_count$ then
12.      exit
13. end
14. set th // as per user preference value
15. foreach $t_i$ in $ADPC$ do
16.   if dp of $t_i < th$ then
17.      eliminate $t_i$ from $ADPC$
18.   end
19. end
20. foreach $t_i \in ADPC$ do
21.   if dp of $t_i == dp$ of any existing Cluster then
22.      insert $t_i$ in $C_k$
23.   else
24.      create cluster $C_k$
25.      insert $t_i$ in $C_k$
26.   end
27. end
```

```
28. foreach $t_i$ of $C_i \in R_i$ do
29.   if BP of $t_i$ == $BP$ of any existing Group $G_i$ then
30.     insert $t_i$ in $C_K$
31.   else
32.     create Group $G_k$
33.     insert $t_i$ in $C_k$
34.   end
35. end
36. foreach $t_i$ of $G_i$ in $C_i \in R_i$ do// Identify skylines within Groups
37.   $isdominated = false$
38.   foreach $t_j$ in $G_i$ do//$j = i{+}{+}$
39.     if $t_i \succ t_j$ then
40.       eliminate $t_j$ from $G_i$
41.   elseif $t_j \succ t_i$ then
42.       set $isdominated = true$
43.     end
44.   end
45.   if $Isdominated$ then
46.     eliminate $t_i$ from $G_i$
47.     end
48.   end
49. foreach $t_i$ of $G_i$ within $C_i \in R_i$ do// identify skylines with cluster
50.   foreach $t_p$ of $G_{i++}$ do
51.     repeat step 39 – 46
52.   end
53. end
54. foreach $C_i \in R_i$ do
55.   foreach $d_i$ of $C_i \in R_i$ do
56.     mark $t_i$ iff it has highest value in $d_i$
57.   end
58.   eliminate all non-marked $t_i$ from $C_i$
59. end
60. foreach $t_i$ of $Ci \in R_i$ do// identify skylines for $R_i$
61.   foreach $t_p$ of $C_{i++}$ do
62.     repeat step 39 – 46
63.   end
64. end
65. combine remaining tuples (skylines) present in $R_i - R_n$ into $R_g$
66. repeat step 1-13 and 20-64
67. return remaining tuples $t_{i-n}$ present in $R_g$ as $GlobalSkylines$
```

V. EXPERIMENTAL ENVIRONMENT

A lot of experiments have been done to measure the efficiency and performance of the proposed approach, CIDS for processing skyline queries in partially complete databases over a cloud environment. Various experiment settings have been designed and developed to measure the performance of CIDS with the most recent proposed approaches, which are designed for incomplete databases such as SCSA [36], ICS [27], SPQ [25], IncoSkyline [35], and SIDS [33]. The proposed approach and existing approaches have been implemented using the $C\#$ programming language. A window 7 32bit machine with 3 GB memory and $i3$ $1.6GHz$ Processor has been used to run a comprehensive and intensive set of experiments. It is a known fact that skyline processing is a CPU exhaustive process [1], [2], [8], [12], [31], [40]–[42], thus the experiments of this research work involved three performance metrics: i) domination tests between tuples, ii) processing time and iii) amount of data transfer from one data center to another. To measure these three parameters two well-known databases have been used (synthetic and real). In terms of real databases, three different datasets have been used such as NBA, MovieLens, and CoIL 2000 insurance company. For the synthetic dataset two datasets have been created (independent and correlated). An Independent dataset is generated where values of one dimension are not related with another relation whereas in a correlated dataset the values of one dimension are related with another dimension positively or negatively. These datasets are more realistic and are frequently used by researchers in the area of evaluating skyline queries in complete or incomplete databases [1], [2], [7], [19], [24], [35], [36], [43]. Furthermore, the performance of the proposed and existing approaches is measured by varying the number of dimensions of a database, the number of partially complete dimensions, and the size of the database. As far as the nature of the Movie Lens and CoIL 2000 insurance company dataset is concerned, they fit our experimental conditions as they are not complete databases. It is important to note that during experimentation the highest value present in dimensions of datasets is considered as a better one. The parameter settings of real and synthetic datasets shown in Table 2 are used to evaluate the proposed algorithm to identify skyline queries in partially incomplete databases.

TABLE 2. Parameter settings of real and synthetic datasets.

Parameter Settings	Datasets				
	Real			Synthetic	
	NBA	CoIL Insurance company 2000	Movie Lens	Independent	Correlated
# dimensions	6, 8, 10, 12, 14, 16, 18	4, 6, 8, 10, 12, 14, 16, 18, 20, 22	4	5, 7, 9, 11, 13	5, 7, 9, 11, 13
# dimensions with missing data	3, 5, 7, 9, 11, 13, 15	3, 5, 7, 9, 11, 13, 15, 17, 19, 21	3	4, 6,8,10, 12	4, 6,8,10, 12
Incompleteness (%)	25-30	25-30	25-30	25-30	25-30
Dataset size	40, 80, 120, 160, 200	50, 100, 150, 200, 250, 300	400, 800, 1200, 1600, 2000	100, 200, 300, 400, 500, 600	100, 200, 300, 400, 500, 600
Data centres	3	3	3	3	3
Dataset size at query submitted data center (%)	40	40	40	40	40
Dataset size atother two data centers (%)	60	60	60	60	60

A. EXPERIMENTAL RESULTS

This section shows the experimental results of our proposed algorithm performed on the datasets (real and synthetic) to identify skylines on partially complete databases stored over a cloud environment. Here, it is attempted to investigate the effect of the number of dimensions, the influence of dataset size on the process of domination tests and processing time for skyline evaluation and the amount of data transfer from one data center to another. From the literature, it can be argued that these are considered crucial parameters and have a huge impact on skyline query processing. In the first set of the experiment, the size of the datasets remains constant whereas the number of the dimensions varies. In the second set of experiments, the number of the dimension of datasets remains constant whereas the size of the dataset varies. In the last part of the experiment set, the amount of data transferred from one data center to another is measured. It is proven from the literature that these parameters are the most crucial parameters that influence skyline query processing [1], [2], [4], [13], [23], [36], [44], [45].

1) EFFECT OF NUMBER OF DIMENSIONS

It is evidenced in literature [2], [19], [46], [47] that the number of dimensions within a dataset has a huge influence on the process of skyline queries. Therefore, in this work, the impact of the varying number of dimensions of incomplete datasets on the processing time taken to identify skylines is examined. In this section, the experimental results of real and synthetic datasets are illustrated throughout the paper. Figure 12 illustrates the impact of a varying number of dimensions on processing time while evaluating skyline queries. Figure 12a and Figure 12b show the execution time taken by the proposed algorithm, CIDS and existing approaches (SCSA, ICS, SPQ, INCOSKYLINE and SIDS) while using real datasets, NBA, and COiL 2000 insurance company dataset respectively. In the NBA dataset, the dataset size is fixed to 120KB, and the number of dimensions varies from 5 dimensions to 17 dimensions. Whereas for COiL 2000 insurance company dataset the dataset size is fixed to 150KB, and the number of dimensions varies from 3 to 21 dimensions. Since the MovieLens dataset consists of only four dimensions it is not used in this experiment.

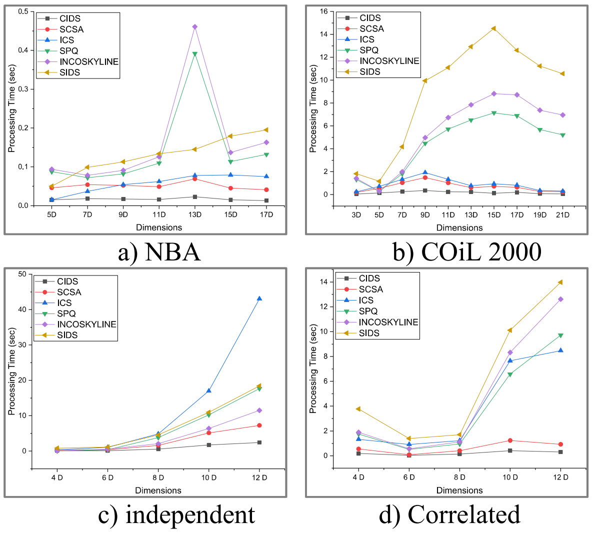


FIGURE 12. Effect of number of dimensions on the processing time.

From the experimental results presented in Figure 12a of the NBA dataset, it can be concluded that CIDS requires less processing time to identify skylines. Its performance remains consistent even though the number of dimensions varies. Whereas the processing time dramatically increases for other approaches such as ICS, SPQ, INCOSKYLINE and SIDS. It can be also noticed that INCOSKYLINE and SIDS take a longer time to identify skylines when the number of dimensions is 11 and above. From Figure 12b it can be concluded that CIDS outruns all the existing approaches in terms of execution time while identifying skylines and the processing time does not increase even when the number of dimensions increases. Whereas SPQ, INCOSKYLINE and SIDS take the longest time to evaluate skylines. However, SCSA and ICS are slightly better than SPQ, INCOSKYLINE and SIDS.

Figure 12c and Figure 12d demonstrate the impact of varying number of dimensions of synthetic datasets (independent and correlated respectively) on processing time to identify skylines by proposed algorithm CIDS and existing approaches (SCSA, ICS, SPQ, INCOSKYLINE and SIDS). For both independent and correlated datasets, the dataset size is fixed to 300KB, whereas the number of dimensions varies from 4 dimensions to 12 dimensions.

From Figure 12c and Figure 12d, it is concluded that CIDS outclasses all other existing approaches (SCSA, ICS, SPQ, INCOSKYLINE and SIDS) in terms of processing time while identifying skylines. It can be also seen that the processing time slightly increases for CIDS when the number of dimensions reaches to 8 and above for the independent dataset set. Whereas processing time for CIDS in correlated datasets remains steady throughout the increase in the number of dimensions. it is also concluded that the processing time dramatically increases for all other approaches especially SPQ, INCOSKYLINE and CIDS when the number of dimensions reaches 8 and above.

The reason behind outperforming all the existing approaches (SCSA, ICS, SPQ, INCOSKYLINE and SIDS) by CIDS is that it uses an optimization technique called pruning in which a large number of tuples are eliminated way before the skyline process starts which in turn minimizes the searching space and reduces the processing time. Another efficient technique implemented in CIDS is that it parallelly executes stages number 2 to 4 in both phases, which has a huge impact on reducing the execution time unlike other approaches such as ICS, SPQ, CIDS where execution occurs step by step.

Figure 13a, 13b, 13c, and 13d show the results of domination tests taken by CIDS, SCSA, ICS, SPQ, INCOSKYLINE, and SIDS approaches to generate skylines on the NBA, CoIL 2000 insurance company, Independent, and Correlated datasets. In this set of experiments, the parameter settings are the same as mentioned in the previous experiment for both real and synthetic datasets. For the NBA dataset in Figure 13a, CIDS takes the least amount of domination tests to compute skylines and outruns all the other approaches. From Figure 13b SPQ, INCOSKYLINE, and SIDS approaches take the highest number of domination tests followed by ICS and SCSA and CIDS outperforms all these approaches in terms of domination tests while identifying skylines.

Figure 13c and 13d show the results of domination tests taken by different approaches in the independent and correlated dataset while identifying skylines. From Figure 13c and 13d, it can be noted that CIDS takes minimum number of domination tests to identify skylines. In Figure 13d the number of domination tests for CIDS remain steady until 8 dimensions but then slowly increase when the number of dimensions also increase. In Figure 13c it can be noted that different approaches take the highest number of domination

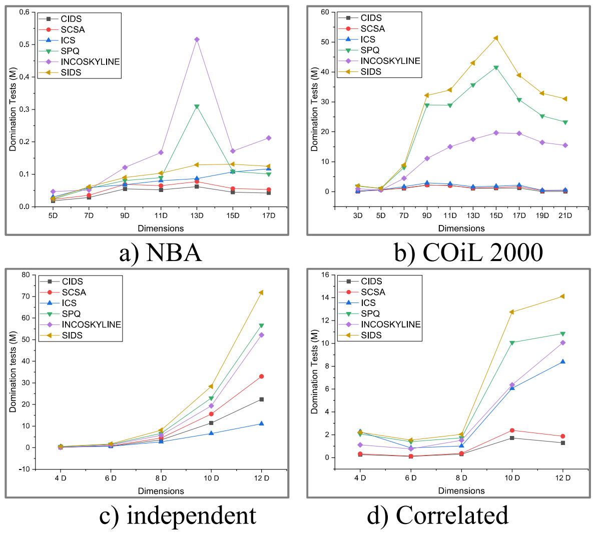


FIGURE 13. **Effect of number of dimensions on the domination tests.**

tests when compared with other datasets. It is because of the nature of the dataset where dimensions are not dependent on each other.

From Figure 13 it can be concluded that CIDS is outperforming all the existing approaches whether it be SCSA, ICS, SPQ, INCOSKYLINE or SIDS. It is achieved due to the fact of using two different optimization techniques, *pruning,* and *selecting superior local skylines*. With the help of *the pruning* technique, a large number of unwanted tuples are eliminated before starting the skyline process. Furthermore, with the help of *selecting superior local skylines* techniques, a good number of tuples are eliminated from further processing. Also, to identify global skylines for all data centers involved, CIDS only processes the local skylines, identified from each data center unlike SCSA, SPQ, INCOSKYLINE or CIDS approach where they process data of all data centers involved to find global skylines.

2) EFFECT OF DATASET SIZE

Figure 14a, 14b, 14c, 14d, and 14e depicts the processing time taken by different approaches to identify skylines on a partially complete database for NBA, COiL 2000 Insurance company, MovieLens, Independent, and Correlated datasets. This set of experiments identifies the impact of dataset size of a partially incomplete database on the process of skyline computation. The parameter settings for this set of experiments are as follows:

For the real dataset, NBA the number of dimensions remains constant (17 dimensions) whereas dataset size increases from 40KB to 200KB. For Coil 2000 Insurance company the number of dimensions remains 13 throughout this experiment whereas the dataset size varies from 50KB to 300KB. In the MovieLens dataset, there are only 4 dimensions including index dimensions and the dataset varies from 400KB to 2000KB. For synthetic dataset, Independent and Correlated the number of dimensions remain constant at 7 (including index dimension) whereas the dataset size varies from 100KB to 600KB.

Figure 14a shows the results conducted on the NBA dataset to identify skylines. From the figure, it can be noticed that CIDS outruns all the existing approaches in all cases. Figure 14b demonstrates the experimental results which were conducted on COiL 2000 Insurance Company dataset. It is noticed that CIDS outruns all the approaches in all cases. SPQ, INCOSKYLINE and SIDS take the longest time in identifying skylines. whereas SCSA and ICS perform slightly better than these approaches. Figure 14c represents the experiment results conducted on the MovieLens dataset. It is concluded that all existing approaches perform worse when compared with the CIDS algorithm. Even though the size of the dataset increases, CIDS performance remains steady and processing time marginally increases along with the dataset size. Figure 14d, 14e shows the experiment results of the synthetic dataset (Independent and Correlated). From Figure 14d and 14e, it is clearly seen that our algorithm is better than SCSA, ICS, SPQ, INCOSKYLINE as well as SIDS in all cases. It requires the least amount of time to identify skylines in partially complete databases.

From Figure 14 it can be concluded that the CIDS algorithm outperforms all the existing approaches by taking least processing time to identify skylines in partially complete databases where dataset size is increasing. It is due to the fact that CIDS uses prominent techniques such as *Filtering, Clustering and Grouping, Finding Local Skylines, Choosing Superior Local Skylines.* All these techniques contribute to reducing processing time as *filtering* helps in eliminating a large number of tuples before the skyline process begins which reduces the searching space. Whereas forming of clustering and groups within clusters and executing them parallelly helps CIDS to process faster and lastly use of optimization technique, *choosing superior skylines* further helps in eliminating those tuples that do not have any potential to be part of the final skyline from each database stored in each data center.

Figure 15a, 15b, 15c, 15d, and 15e describe the experimental results of the domination tests taken by algorithms (CIDS, SCSA, ICS, SPQ, INCOSKYLINE and SIDS), conducted on tuples present in NBA, COiL 2000 Insurance Company, MovieLens, Independent, and Correlated datasets to identify skylines. The parameter settings of this result set are the same as the above one. Figure 15a demonstrates the experiment results of the NBA dataset. From the results shown in the figure, it can be concluded that the number of domination tests gradually increase with the increase in dataset size. SIDS, INCOSKYLINE, SPQ, ICS and SCSA take higher domination tests than CIDS and CIDS outclasses these approaches. SIDS is the worst in all cases. Whereas SCSA performs better than other approaches but is outperformed by CIDS. Figure 15b shows the results of COiL 2000 Insurance Company where it can be seen that SCSA and ICS are marginally outrun by CIDS until the dataset size is 100KB whereas other approaches (SPQ, INCOSKYLINE, SIDS) are performing

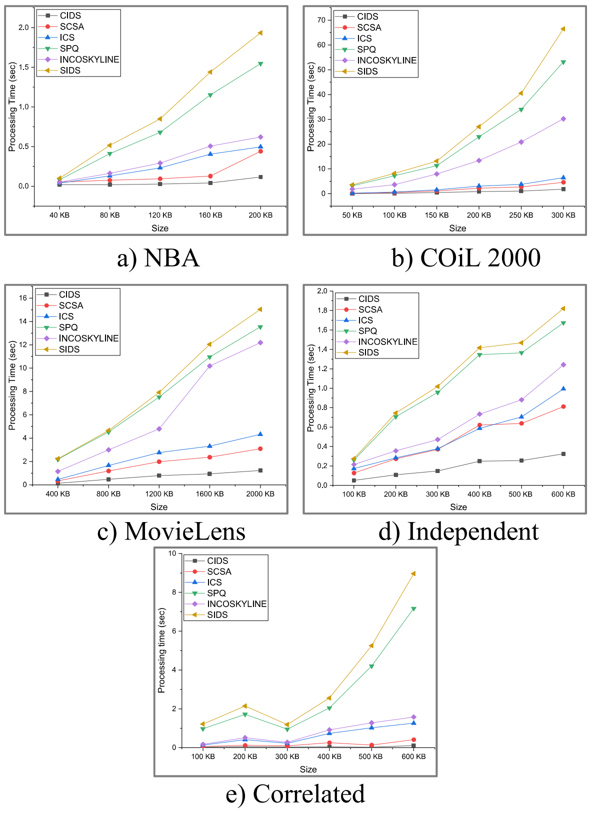


FIGURE 14. Effect of dataset size on the processing time.

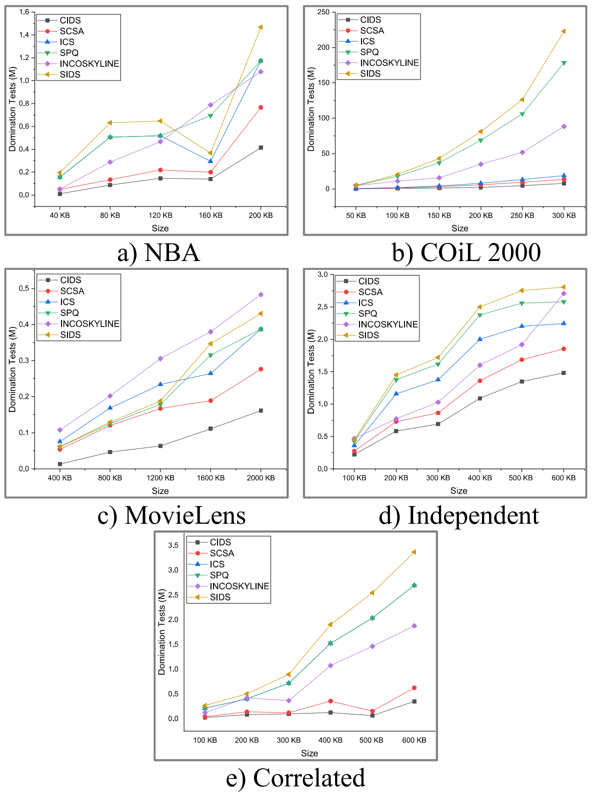


FIGURE 15. Effect of dataset size on the number of domination tests.

worse in all cases and are outperformed by CIDS. Figure 15c displays the experimental results of the MovieLens dataset. From the results presented in the figure, it can be noted that the domination tests gradually increase along with the dataset size, but CIDS takes a minimum amount of domination tests throughout the results and outperforms all the approaches in taking fewer domination tests to identify skylines. Figure 15d and 15e represents the results of the Independent and Correlated dataset. From the results, it is concluded that CIDS outpaces SCSA, ICS, SPQ, INCOSKYLINE and SIDS in taking the least domination tests to generate final skylines.

From the results presented in Figure 15, it is concluded that CIDS outruns all the existing approaches (SCSA, ICS, SPQ, INCOSKYLINE and SIDS) in terms of the number of domination tests while identifying skylines in partially complete databases. Incorporating of two optimization techniques (pruning and choosing superior local skylines) have proven that they eliminate a lot of dominated tuples before the skyline process starts and stops many non-dominant tuples from further processing. These two techniques play a vital role in reducing domination tests. Moreover, the concept of creating clusters based on domination power helps in putting tuples with higher *dp* value other tuples in similar clusters. due to that technique, the tuples having higher *dp* value have potential to dominate and eliminate tuples in other clusters (with lesser *dp* value) when a cross-comparison is done. That helps in reducing domination by eliminating such non-dominant tuples way before getting compared with other tuples present in other clusters with less domination power. Creating groups based on bitmap representation helps CIDS to avoid losing transitive property and stops cyclic dominance from occurring. Comparing tuples with the same bitmap representation helps CIDS to avoid false negative and false positive cases to occur. Lastly comparing tuples within groups saves a lot of domination tests by eliminating many non-dominant tuples within groups before getting compared with other tuples present in other groups.

3) EFFECT OF AMOUNT OF DATA TRANSFER

In this set of experiments, the amount of data transferred from one data center to another data center is evaluated. In other words, data transfer means the number of tuples transferred from one data center to another. The existing approaches are designed for single databases except (ICS) whereas CIDS is designed to handle multiple databases located at remote locations over the cloud. When identifying skylines over is cloud it is important to identify those tuples which are not being dominated by any tuple present in any data center. For that data needs to be transferred from one data center to another. Transferring data from one data center to another is considered an expensive process. To avoid transferring all the data present at one data center to another CIDS implemented a cost-effective technique that only transfers the local skylines of each relation present at each data center rather than transferring all the data unlike SCSA, SPQ, INCOSKYLINE and SIDS. Figure 16a, 16b,16c, 16d, and 16e depicts the results of the amount of data transferred from one data center to another of NBA, COiL 2000 Insurance Company, MovieLens, Independent and Correlated datasets.

From figure 16 it can be concluded that around 98-99% of data transfer is reducing in CIDS whereas SCSA, SPQ, INCOSKYLINE and SIDS transfers the whole dataset in order to identify global skylines. CIDS also outruns ICS as ICS does not use any crusting or grouping technique to filter out some non-dominant tuples whereas CIDS uses two optimization techniques to remove many unwanted tuples during the skyline process. That has an impact on reducing the number of tuples to be transferred from one data center to another.

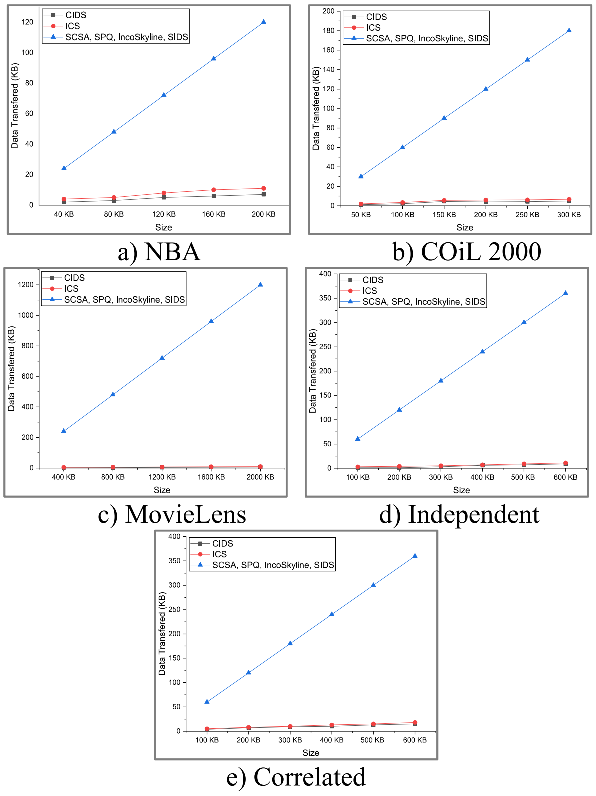


FIGURE 16. **Effect of dataset size on the number of domination tests.**

VI. CONCLUSION

The skyline process is generally considered an expensive process, given the extensive domination tests performed during the process of identifying skylines. The parameters such as the number of dimensions and dataset size are considered as an important factor that shows the baneful impact on the process of the skyline in terms of searching space and computation process. This research work proposed a novel algorithm (CIDS) for processing skyline queries in partially complete databases stored over a cloud environment. CIDS implements two optimization techniques, *pruning* and *choosing superior skylines* to prune some unwanted tuples that do not have the potential to be part of final skylines. CIDS also implements a novel approach of identifying the domination power of individual tuples which represents their domination power over other tuples. Identifying the domination power of each tuple helps in creating different clusters and based on that the most dominant tuples are a productive technique. To make CIDS efficient clusters are further divided into groups and then local skylines are identified simultaneously and parallelly for each cluster within a relation and the same technique is executed simultaneously and parallelly for each data center involved. These techniques have played a vital role in reducing domination tests between tuples and reducing execution time. Furthermore, transferring only skylines of each relation from one data center to another has reduced a large amount of data from being transferred. Which in turn reduced network cost. A different set of experiments have been conducted to prove the efficiency and effectiveness of CIDS over existing approaches such as SCSA, ICS, SPQ, INCOSKYLINE and SIDS. The results have proven that CIDS is outclassing all the existing approaches in identifying the skyline for partially complete data stored over cloud environment.

REFERENCES

[1] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. 17th Int. Conf. Data Eng.*, Apr. 2001, pp. 421–430, doi: 10.1109/ICDE.2001.914855.

[2] 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, doi: 10.1109/ICDE.2008.4497464.

[3] 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," *Proc. Comput. Sci.*, vol. 94, pp. 191–198, Jan. 2016, doi: 10.1016/j.procs.2016.08.030.

[4] 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, doi: 10.1145/1142473.1142530.

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

[6] 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 Databases (VLDB Endowment)*, Hong Kong, 2002, pp. 275–286.

[7] 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.

[8] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, and A. K. H. Tung, "On high dimensional skylines," in *Proc. 10th Int. Conf. Extending Database Technol. (EDBT)* Berlin, Germany: Springer, 2006, pp. 478–495.

[9] 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, doi: 10.11591/ijeecs.v10.i2.pp798-806.

[10] 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.

[11] L. Ding, X. Zhang, H. Zhang, L. Liu, and B. Song, "CrowdSJ: Skyline-join query processing of incomplete datasets with crowdsourcing," *IEEE Access*, vol. 9, pp. 73216–73229, 2021.

[12] M. B. Swidan, A. A. Alwan, S. Turaev, H. Ibrahim, A. Zaid Abualkishik, and Y. Gulzar, "Skyline queries computation on crowdsourced- enabled incomplete database," *IEEE Access*, vol. 8, pp. 106660–106689, 2020, doi: 10.1109/ACCESS.2020.3000664.

[13] 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.

[14] M. Morse, J. M. Patel, and W. I. Grosky, ''Efficient continuous skyline computation,'' *Inf. Sci.*, vol. 177, no. 17, pp. 3411–3437, Sep. 2007.

[15] C. Kalyvas, T. Tzouramanis, and Y. Manolopoulos, ''Processing skyline queries in temporal databases,'' in *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 893–899.

[16] Y. Gulzar, A. A. Alwan, A. Z. Abualkishik, and A. Mehmood, ''A model for computing skyline data items in cloud incomplete databases,'' *Proc. Comput. Sci.*, vol. 170, pp. 249–256, Jan. 2020, doi: 10.1016/j.procs.2020.03.037.

[17] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, ''Skyline with presorting,'' in *Proc. 19th Int. Conf. Data Eng.*, Mar. 2003, pp. 717–719, doi: 10.1109/ICDE.2003.1260846.

[18] 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)*, 2003, pp. 467–478, doi: 10.1145/872757.872814.

[19] Y. Gulzar, A. A. Alwan, H. Ibrahim, S. Turaev, S. Wani, A. B. Soomo, and Y. Hamid, ''IDSA: An efficient algorithm for skyline queries computation on dynamic and incomplete data with changing states,'' *IEEE Access*, vol. 9, pp. 57291–57310, 2021.

[20] Y. Gulzar, ''Skyline query approaches in static and dynamic incomplete databases,'' Int. Islamic Univ. Malaysia, Selangor, Malaysia, Tech. Rep., 2018.

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

[22] 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, doi: 10.3844/jcssp.2017.647.658.

[23] J. Lee, H. Im, and G.-W. You, ''Optimizing skyline queries over incomplete data,'' *Inf. Sci.*, vols. 361–362, pp. 14–28, Sep. 2016, doi: 10.1016/j.ins.2016.04.048.

[24] 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.

[25] 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, doi: 10.1109/ACCESS.2016.2639558.

[26] A. A. Alwan, H. Ibrahim, N. I. Udzir, and F. Sidi, ''Processing skyline queries in incomplete distributed databases,'' *J. Intell. Inf. Syst.*, vol. 48, no. 2, pp. 399–420, Apr. 2017.

[27] Y. Gulzar, A. A. Alwan 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.

[28] A. A. Alwan, H. Ibrahim, N. I. Udzir, and F. Sidi, ''Estimating missing values of skylines in incomplete database,'' in *Proc. 12th Int. Conf. Digit. Enterprise Inf. Syst.*, 2013, pp. 220–229.

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

[30] N. H. M. Saad, H. Ibrahim, F. Sidi, R. Yaakob, and A. A. Alwan, ''Computing range skyline query on uncertain dimension,'' in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2016, pp. 377–388.

[31] M. Shamsul Arefin, ''Skyline sets queries for incomplete data,'' *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 5, pp. 67–80, Oct. 2012.

[32] X. Miao, Y. Gao, L. Chen, G. Chen, Q. Li, and T. Jiang, ''On efficient K-skyband query processing over incomplete data,'' in *Proc. 18th Int. Conf. Database Syst. Adv. Appl. (DASFAA)*, Wuhan, China, Apr. 2013, pp. 424–439, doi: 10.1007/978-3-642-37487-6_32.

[33] R. Bharuka and P. S. Kumar, ''Finding skylines for incomplete data,'' in *Proc. 24th Australas. Database Conf.*, vol. 137. Adelaide, QLD, Australia: Australian Computer Society, 2013, pp. 109–117.

[34] 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, doi: 10.1007/978-3-319-32055-7_26.

[35] 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, doi: 10.1007/s13369-016-2048-z.

[36] Y. Gulzar, A. A. Alwan, R. M. Abdullah, Q. Xin, and M. B. Swidan, ''SCSA: Evaluating skyline queries in incomplete data,'' *Int. J. Speech Technol.*, vol. 49, no. 5, pp. 1636–1657, May 2019, doi: 10.1007/s10489-018-1356-2.

[37] Y. Gulzar, A. A. Alwan, and S. Turaev, ''Optimizing skyline query processing in incomplete data,'' *IEEE Access*, vol. 7, pp. 178121–178138, 2019, doi: 10.1109/ACCESS.2019.2958202.

[38] H.-C. Ryu and S. Jung, ''MapReduce-based skyline query processing scheme using adaptive two-level grids,'' *Cluster Comput.*, vol. 20, no. 4, pp. 3605–3616, Dec. 2017, doi: 10.1007/S10586-017-1203-Y.

[39] P. Ezatpoor, J. Zhan, J. M.-T. Wu, and C. Chiu, ''Finding top-K dominance on incomplete big data using mapreduce framework,'' *IEEE Access*, vol. 6, pp. 7872–7887, 2018.

[40] M. Haddache, A. Hadjali, and H. Azzoune, ''Skyline refinement exploiting fuzzy formal concept analysis,'' *Int. J. Intell. Comput. Cybern.*, vol. 14, no. 3, pp. 333–362, Jul. 2021.

[41] R. Chi-Wing Wong, A. Wai-chee Fu, J. Pei, Y. Sing Ho, T. Wong, and Y. Liu, ''Efficient skyline querying with variable user preferences on nominal attributes,'' 2007, *arXiv:0710.2604*.

[42] M. A. Soliman, I. F. Ilyas, and S. Ben-David, ''Supporting ranking queries on uncertain and incomplete data,'' *Int. J. Very Large Data Bases*, vol. 19, no. 4, pp. 477–501, 2010.

[43] Y. Gulzar, A. A. Alwan, H. Ibrahim, and Q. Xin, ''D-SKY: A framework for processing skyline queries in a dynamic and incomplete database,'' in *Proc. 20th Int. Conf. Inf. Integr. Web-Based Appl. Services*, 2018, pp. 164–172, doi: 10.1145/3282373.3282389.

[44] 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, VA, USA, 2006, pp. 405–414, doi: 10.1145/1183614.1183674.

[45] 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, doi: 10.1166/asl.2018.10745.

[46] G. B. Dehaki, H. Ibrahim, A. A. Alwan, F. Sidi, and N. I. Udzir, ''Efficient skyline computation over an incomplete database with changing states and structures,'' *IEEE Access*, vol. 9, pp. 88699–88723, 2021.

[47] N. H. M. Saad, H. Ibrahim, F. Sidi, R. Yaakob, and A. A. Alwan, ''Efficient skyline computation on uncertain dimensions,'' *IEEE Access*, vol. 9, pp. 96975–96994, 2021.

YONIS GULZAR received the bachelor's degree in computer science from the University of Kashmir, in 2010, the master's degree in computer science from Bangalore University, India, in 2013, and the Ph.D. degree in computer science from the International Islamic University Malaysia, Malaysia, in 2018.

He worked as a Lecturer (PT), a Teaching Assistant, and a Research Assistant at the Department of Computer Science, International Islamic University Malaysia, till 2018. He has been an Assistant Professor at King Faisal University (KFU), Saudi Arabia, since 2019. His research interests include preference queries, skyline queries, query processing, machine learning, deep learning, image processing, and computer vision.

ALI A. ALWAN received the master's and Ph.D. degrees in computer science from Universiti Putra Malaysia (UPM), Malaysia, in 2009 and 2013, respectively. He is currently an Assistant Professor at the School of Theoretical and Applied Science, Ramapo College of New Jersey, USA. His research interests include databases (mobile, distributed, and parallel), preference queries, web databases, probabilistic, incomplete and uncertain databases, query processing and optimization, data management, data integration, location-based social networks (LBSN), recommendation systems, data mining, database in cloud, big data management, and crowd-sourced database.