

Received June 26, 2020, accepted July 15, 2020, date of publication July 24, 2020, date of current version August 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3011652

# Efficient Computation of Skyline Queries Over a Dynamic and Incomplete Database

GHAZALEH BABANEJAD DEHAKI<sup>1</sup>, HAMIDAH IBRAHIM<sup>1</sup>, FATIMAH SIDI<sup>1</sup>,  
NUR IZURA UDZIR<sup>1</sup>, ALI A. ALWAN<sup>2</sup>, AND YONIS GULZAR<sup>3</sup>

<sup>1</sup>Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang 43400, Malaysia

<sup>2</sup>Department of Computer Science, Kuliyah of Information and Communication Technology, International Islamic University Malaysia, Kuala Lumpur 53100, Malaysia

<sup>3</sup>Department of Management Information Systems, College of Business Administration, King Faisal University, Hofuf 31982, Saudi Arabia

Corresponding author: Hamidah Ibrahim (hamidah.ibrahim@upm.edu.my)

This work was supported by the Malaysian Ministry of Science, Technology, and Information (MOSTI) under the Fundamental Research Grant Scheme (Grant No. 08-01-16-1853FR) and the Universiti Putra Malaysia.

**ABSTRACT** Skyline queries rely on the notion of Pareto dominance, filter the data items by keeping only those data items that are the best, most preferred, also known as skylines, from a database to meet the user's preferences. Skyline query has been studied extensively and a significant number of skyline algorithms have been proposed, mostly attempt to resolve the optimisation problem that is mainly associated with a reduction in the processing time of skyline computations. In today's era, the presence of incomplete data in a database is inevitable. Furthermore, databases are dynamic in nature in which their states change throughout the time to reflect the current and latest information of the applications. The skylines derived before changes are made towards the initial database are no longer valid in the new state of the database. Blindly examining the entire database to identify the new set of skylines is unwise as not all data items are affected by the changes made towards the database. Hence, in this paper we propose a solution, named *DyIn-Skyline*, which is capable of deriving skylines over a dynamic and incomplete database, by exploiting only those data items that are affected by the changes. Several experiments have been conducted and the results show that our proposed solution outperforms the previous works with regard to the number of pairwise comparisons and processing time.

**INDEX TERMS** Skyline queries, incomplete database, dynamic database, pairwise comparisons.

## I. INTRODUCTION

Query processing which extracts data items<sup>1</sup> from a database according to a set of access criteria, also known as conditions, and presents these data items to the user for use, has achieved tremendous success at both research and industry levels. There are many types of queries that have been introduced mainly to accommodate the different needs of applications or systems. For instance, a temporal query (based on temporal query language) retrieves time-referenced or temporal data for applications that require information relating to the past, present, and future time. On the other hand, a spatial query which uses geometry data types such as points, lines, and polygons and considers the spatial relationships between

these geometries; is useful in Geographic Information System (GIS), Multimedia Information System (MIS), or Computer Aided-Design (CAD).

The traditional query processing operates either by retrieving data items from a database that strictly satisfy each condition specified in the query or returning an empty result if otherwise. The recent developments in query processing attempt to relax these stringent requirements, by retrieving the best, most preferred data items from a database according to the conditions specified in the query, also known as user-defined preferences. These preference queries employ preference evaluation techniques, have achieved significant success, as they are widely used in applications related to multi-criteria decision support. During the two past decades, several preference evaluation techniques have been introduced, among them are: top- $k$  [30], skyline [8], [10]–[13], [20], [25], [29], [33], [35],  $k$ -dominance [5], top- $k$  dominating [19], and  $k$ -frequency [6].

The associate editor coordinating the review of this manuscript and approving it for publication was Genoveffa Tortora<sup>1</sup>.

<sup>1</sup>Without loss of generality, we use the term data item throughout this paper to be in line with other research works in similar area. The terms *data*, *object*, *record*, and *tuple* can also be used in this context.

Skyline queries rely on the notion of Pareto dominance filter the data items from a database by keeping only those data items that are not worse than any other. It is a well-known technique that is utilised to identify the best, most preferred data items, also known as skylines, from a database to meet the user's preferences. Consider a user who wanted to go for a holiday with the following preferences: (i) hotel that is nearest to the beach (minimum distance) and (ii) hotel with the cheapest price (minimum price). Generally, hotels that are near to a beach are expensive as compared to those which are far away from a beach, which implies that the chances to find a hotel that meets both preferences are nil. Taking this into consideration, the user is left with three choices: (i) hotel that is nearest to the beach (minimum distance) while the price is not the cheapest, (ii) hotel with the cheapest price (minimum price) while it is not the nearest hotel to the beach, and (iii) hotel(s) with price cheaper than hotel (i) and distance nearer than (ii). Eventually, the user will have to make the final decision by choosing a hotel from these filtered hotels. Unlike the traditional query which will obviously return an empty result since there is no hotel with minimum price and minimum distance, the skyline query which relies on the powerful skyline operator introduced by [29] managed to return results that are not dominated by any other based on the user-defined preferences.

Since the introduction of skyline queries, there are a lot of research works that have been conducted mainly to solve the optimisation problem in computing the skylines [7], [8], [10], [12], [18], [25]. The skyline operator introduced by [29] only works with the assumption that data items in the database are comparable. However, in today's era, the presence of incomplete data in a database is inevitable. The skyline algorithms in such situation will have to deal with several issues besides the optimisation problem. The missing values in databases give a negative influence on the number of pairwise comparisons that needs to be performed between the data items. Moreover, the transitivity property of skylines is no longer hold. Cyclic dominance is another issue that needs to be tackled as it yields empty skyline results [23]. Furthermore, databases need to frequently change their state to reflect the current and latest information of the applications. The changes are normally achieved through data manipulation operations like insert, delete, or update operations. The skylines derived before changes are made towards the initial database are no longer valid in the new state of the database. Utilising the existing skyline algorithms would require computing the skylines over the entire database after changes are made which is unwise as not all the data items are affected by the changes. Hence, the incompleteness and dynamism nature of data make the process of identifying skylines no longer a trivial task. This paper takes the challenge to solve the problem associated to identifying skylines over a dynamic and incomplete database. In general, the main contributions of this work are briefly described as follows:

- We have proposed a solution, named *DyIn-Skyline*, that is capable of deriving a set of skylines over a dynamic

and incomplete database. *DyIn-Skyline* consists of two main phases, that are simply named as *Phase I* and *Phase II*. *Phase I* focuses on the processing of skyline queries over the initial incomplete database while *Phase II* focuses on the processing of skyline queries over a dynamic and incomplete database, in which the changing state of the database is due to a data manipulation operation(s) (insert, delete or update a data item(s)).

- We have introduced and designed three main lists, namely: *Domination History (DH)*, *Bucket Dominating (BDG)*, and *Bucket Dominated (BDD)*, that keep track of the domination relationships, dominating data items, and dominated data items, respectively. These lists are crucial as they assist in identifying the data items that are affected by the changes made towards a database, hence excluding the unaffected data items from the process of computing skylines.
- The experimental results of the proposed solution are presented to prove *DyIn-Skyline*'s capability in deriving a set of skylines after changes are made towards a database.

The rest of the paper is organized as follows. Section II reviews the approaches proposed by previous studies that are related to the issues highlighted in this paper. Section III presents the definitions and introduces the notations used throughout this paper. Section IV presents our proposed solution *DyIn-Skyline*. Part A of Section IV presents the *Phase I* of *DyIn-Skyline* while Part B of Section IV presents the *Phase II* of *DyIn-Skyline*. Section V evaluates the performance of our proposed solution, *DyIn-Skyline*, which is compared to other previous works. The last section concludes this work and sheds light on some directions which can be used in the future.

## II. RELATED WORK

There are various techniques that have been proposed for preference queries, which include top- $k$  [30],  $k$ -dominance [4], top- $k$  dominating [20],  $k$ -frequency [7], skylines [29], multi-objective skyline [3], and ranked skylines [15]. These techniques attempt to find the best results that meet the user's preferences. Besides, several skyline algorithms have been introduced such as *Block-Nested-Loop (BNL)* [29], *Divide and Conquer (D&C)* [29], *Bitmap* and *Index* [18], *Sort-Filter-Skyline (SFS)* [12], *Nearest Neighbor (NN)* [8], *Branch-and-Bound Skyline (BBS)* [7], *Linear Elimination Sort for Skyline (LESS)* [10], and *Sort and Limit Skyline algorithm (SaLSa)* [10]. The *BNL* [29] retrieves skyline points by reading a set of data items and comparing them to each other; the dominated data items after the comparisons are removed while the remaining ones are the final skylines. Meanwhile, the *D&C* algorithm [29] divides the database into smaller partitions with an attempt to reduce the searching space. For each partition, the local skylines are retrieved. These local skylines are then compared to each other to remove the dominated ones while the remaining data items are the final skylines. The *Bitmap* method introduced by [18] assigns a bit (0 or 1) to each dimension of the data items and performs the AND

operation over them while the *Index* method proposed by the same authors [18] splits the database into smaller sections. In each section, the data items are categorised based on the smallest value in the same dimension. The local skylines of each section are then computed and these local skylines are compared to derive the global skylines. The *SFS* algorithm proposed by [12] sorts the data items based on a specific sorting function in a way that the data items which are stored at the beginning of the relation are delivered instantly as preliminary skylines. In addition, the *NN* algorithm [8] which utilises the *D&C* scheme to split the indexed database into small regions in order to prune the searching space, attempts to retrieve the first skylines while the algorithm is running. Nevertheless, the *BBS* algorithm [7] aims at returning skylines in a progressive manner with fewer amount of data needed to be accessed. *BBS* employs the *R-tree* indexes to compute the skylines based on the best first nearest neighbour search. *LESS* [10] is an algorithm that works on non-indexed data and does not require any additional pre-processing steps. It adopts the benefits of *BNL* [29] and *SFS* [12] while avoiding their disadvantages to provide a substantial enhancement on the process of preference evaluation. The *SaLSa* algorithm [10] exploits the concept of *SFS* [12] method to pre-sort the data items and progressively select a subset of the data items to derive the skylines. However, all of the above algorithms focus on the issues of skyline computations with the assumption that the database is complete and static.

In recent past, focus has been given to resolving issues related to the incompleteness of data in a database, in which several preference query approaches have been established [1], [2], [9], [14], [16], [17], [23], [24], [27], [31], [32], [34]. Most of these works focus on top- $k$  and  $k$ -dominant queries [31], [32] while others like [14], [16], and [17] concentrate on techniques to rank the skyline results. The *Top-k dominating (TKD)* algorithm [31] utilises the upper bound score pruning and bitmap binning strategies while *k-dominant Skyline Queries on Incomplete Data (IkDS)* [32] rely on the conventional strategies, namely: local skyline, dominance ability, bitmap index in retrieving skyline results. Moreover, pruning strategy and probabilistic model have been used by [16] and [17] in ranking the skyline results. Meanwhile, the works by [24] and [34] combine both top- $k$  and skylines to gain a better set of skyline results. To the best of our knowledge only the works by [2], [23], and [27] have made attempts to tackle the issues of processing skyline queries over an incomplete database which are further elaborated below.

The early research work on processing skyline queries over incomplete database is conducted by [23] in which two algorithms are proposed, namely: *Bucket* and *Iskyline*. The *Bucket* algorithm divides the data items of the database into distinct buckets based on their bitmap representation. Each bucket contains the data items which have missing values in the same dimensions. Then, the conventional skyline algorithm is utilised on each bucket to identify the local skylines. Finally, the local skylines of each bucket are compared to each other

to derive the final skylines. The *Iskyline* algorithm handles the skyline queries over an incomplete relational database by dividing the initial database into distinct nodes depending on the missing values of the dimensions and then applying the conventional skyline technique to retrieve the local skylines in every node. *Iskyline* method conducts two optimisation techniques that reduce the number of local skylines in every node. However, *Iskyline* is time consuming as in each node many pairwise comparisons need to be performed to find the local skylines.

In *SIDS* which is proposed by [27], the input data set is pre-sorted in non-increasing order for each dimension, to determine the processing order of the points. The proposed approach chooses one of the dimensions in a round-robin fashion and then, the point having the next best value in that dimension is chosen for processing. There are two variables that are *processedCount* and *dimCount*. *processedCount* keeps track of the number of times a point has been processed from the sorted arrays while *dimCount*, stores the number of complete dimensions. There are also two sets, namely: *CandidateSet* and *ResultSet*. *CandidateSet* stores the candidate skyline points. *ResultSet* stores the points which are determined to be the final skyline points. *CandidateSet* is initialised to the data set size. Also, there is an array pointer, *ptr1*. The intuition behind this is to process relatively dominant points early, so that non-skyline points can be pruned as early as possible. Consequently, a lesser number of comparisons would be required for determining the skylines and this would, in turn, reduce the execution time of the algorithm. The algorithm initially considers all points in the data set as candidate skyline points and then iteratively removes dominated points from the candidate set. If a point has not been pruned yet and has been processed  $k$  times, where  $k$  is the count of complete dimensions for the point, then it is determined to be a skyline point and can be returned immediately. The reason is any point with  $k$  complete dimensions can be dominated in at most  $k$  dimensions.

In *Incoskyline* [2], first the input data set is categorised in the related bucket like the *Iskyline* algorithm. Then grouping and finding local skylines are performed. The groups are created based on the highest value of one of the data items in the cluster. Those data items that shared the same highest value are grouped in the same group. Besides that, the lowest value of the data items in each created group is determined as well. The upper bound of a cluster, *ubc*, and the lower bound of a group, *lbg*, help to stop examining the remaining data items of a cluster if *ubc* is less than or equal to *lbg* of a recently created group. The next phase removes the local skylines of a cluster which are dominated by the local skylines of other cluster(s). This is achieved by deriving a set of virtual skylines named  $k$ -dom from the local skylines. These  $k$ -dom are then merged to derive only one global  $k$ -dom skyline by considering the highest value in each dimension of the  $k$ -dom skylines. Furthermore, this global  $k$ -dom skyline is inserted at the top of each cluster to eliminate the dominated local skylines. The last phase of their proposed approach attempts

to identify the data items that are not dominated by any other data items in the entire database. The final result is retrieved after conducting pairwise comparisons among the candidate skylines.

Nevertheless, there are algorithms which are proposed specifically for a dynamic database; most of them are based on top- $k$ . For instance, the work by [21] focuses on top- $k$  and top- $k$  dominating and reviews algorithms for evaluating continuous preference queries under the sliding window streaming model. Also, a  $k$ -dominant skyline algorithm has been presented in [6]. In their work when the data set is changed, the existing  $k$ -dominant skylines are compared to the new  $k$ -dominant data items to derive the results [6].

Table 1 summarises the works presented in this section.

**TABLE 1. Summary of the related works.**

Algorithm	Database	Query Type	Approach
Block-Nested-Loop (BNL) [29]	Complete	Skyline	Block-nested-loops algorithm
Divide and Conquer (D&C) [29]	Complete	Skyline	Divide-and-conquer algorithm
Bitmap and Index [18]	Complete	Skyline	Bitmap representation, Indexing
Sort-Filter-Skyline (SFS) [12]	Complete	Skyline	Presorting
Nearest Neighbor [8]	Complete	Skyline	Progressive processing
Branch-and-Bound Skyline (BBS) [7]	Complete	Skyline	Nearest-neighbor search, Progressive processing
Linear Elimination Sort for Skyline (LESS) [10]	Complete	Skyline	Non-indexing
Sort and Limit Skyline algorithm (SaLSa) [10]	Complete	Skyline	Sorting
Top- $k$ dominating (TKD) [31]	Incomplete	Top- $k$	Upper bound score pruning, Bitmap binning strategy
$k$ -dominant Skyline Queries on Incomplete Data (IkDS) [32]	Incomplete	$k$ -dominant skyline	Local skyline, Dominance ability, Bitmap index
Sorting-based Bucket Skyline [14]	Incomplete	Skyline	Bucket- and point-level orders
PISkyline [16]	Incomplete	Skyline	Probabilistic model, Pruning
ISSA [17]	Incomplete	Skyline	Bitmap Representation (Bucket technique), Pruning
RBSSQ [24]	Incomplete	Skyline	Replacement-based approach
[34]	Incomplete	Skyline	Sorting
Incoskyline [2]	Incomplete	Skyline	Bitmap representation, Clustering, Optimisation technique ( $k$ -dom skylines)
Bucket and Iskyline [23]	Incomplete	Skyline	Bitmap representation (Bucket technique), Optimisation technique
Sort-based Incomplete Data Skyline (SIDS) [27]	Incomplete	Skyline	Sorting
$k$ -dominant skyline [6]	Dynamic data set	$k$ -dominant skyline	Filtering
DynIn-Skyline	Incomplete Dynamic	Skyline	Bitmap representation (Bucket technique), Domination list

### III. PRELIMINARIES

This section presents the definitions and introduces the necessary notations that are used throughout this paper.

**Definition 1 (Incomplete Database):** A database,  $D$ , with  $m$  dimensions,  $d = \{d_1, d_2, \dots, d_m\}$  and  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$  is incomplete denoted as  $D_I$  if and only if it contains at least a data item  $p_i$  with missing value in one

or more of its dimensions,  $d_j$ , where  $d_j \in d$ ; otherwise, it is complete. We use the symbol ‘-’ to denote a missing value. For example, the data item  $p_i(-, 6, 2, -)$  demonstrates that the first and fourth dimensions have missing values.

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

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

**Definition 4 (Skylines):** Given a database  $D$  with  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$ ,  $p_i$  is a skyline of  $D$  if there are no other data items  $p_j \in D$  that dominates  $p_i$ . We use the symbol  $S$  to denote the set of skylines of a database  $D$ .

**Definition 5 (Transitivity property of skylines):** Skylines hold the transitivity property that means given a database  $D$  with  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$ , if  $p_i \succ p_j$  and  $p_j \succ p_k$ , then this implies that  $p_i \succ p_k$  and  $p_k \not\succ p_i$  [29]. However, in an incomplete database the transitivity property of skylines no longer holds due to cyclic dominance where none of the data items is considered as skyline as every data item is dominated by at least one other data item. This means, if  $p_i \succ p_j$  and  $p_j \succ p_k$ , then it is possible that  $p_k \succ p_i$  and  $p_i \not\succ p_k$ . For example, consider the following incomplete data items,  $p_i(4, 3, 4, -)$ ,  $p_j(2, 1, -, 5)$ , and  $p_k(-, -, 5, 2)$ . Here,  $p_i$  dominates  $p_j$  as  $p_i$  is greater than  $p_j$  in the common dimensions (i.e. first and second dimensions). Also,  $p_j$  dominates  $p_k$  as the only common dimension is the fourth dimension in which  $p_j$  is greater than  $p_k$ . However, when comparing  $p_i$  to  $p_k$ , the third dimension is the only common dimension in which  $p_k$  is greater than  $p_i$ . Thus,  $p_i$  does not dominate  $p_k$  which means that the dominance relationship is non-transitive. Moreover,  $p_k$  dominates  $p_i$  which means that the dominance relationship ends to be cyclic. As a result, none of the data items can be considered as a skyline.

**Definition 6 (Revised Bitwise):** Given a database  $D$  with  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$ ,  $p_i$  and  $p_j$  with different bitmap representations, i.e.  $p_i$  and  $p_j$  are incomparable as



defined in *Definition 2*. Nevertheless,  $p_i$  and  $p_j$  are comparable on their *revised bitwise* if it is not equal to 0, which is obtained by performing an AND operation on the bitmap representations of  $p_i$  and  $p_j$ . For instance, the bitmap representations of the data items  $p_i(-, 6, 2, -)$  and  $p_j(7, -, -, 9)$  are 0110 and 1001, respectively. Based on *Definition 2*, these two data items are incomparable. They are also incomparable based on their *revised bitwise* which is obtained by performing the AND operation on the 0110 and 1001; which results in 0000 or simply 0.

**Definition 7 (Dominance Relationship on the Revised Bitwise):** Given a database,  $D$ , with  $m$  dimensions,  $d = \{d_1, d_2, \dots, d_m\}$  and  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$ ; and two data items  $p_i$  and  $p_j \in D$  with different bitmap representations. If their *revised bitwise* is not equal to 0, then  $p_i$  is said to dominate  $p_j$  on the *revised bitwise* denoted by  $p_i > p_j$  if and only if the following condition holds:  $\forall d_k \in d', p_i.d_k \geq p_j.d_k \wedge \exists d_l \in d', p_i.d_l > p_j.d_l$  where  $d'$  is a set of dimensions whose revised bitwise representation is 1 and  $d' \subset d$ . For instance, consider the data items  $p_i(-, 6, 2, -)$  and  $p_j(-, 5, -, 5)$ ,  $p_i > p_j$  as  $p_i$  is better than  $p_j$  in the second dimension.

**Definition 8 (Dynamic Database):** A database,  $D$ , is said to be dynamic denoted as  $D_D$  if the data items in the database keep on changing in which a new data item(s) is inserted into the database, while an existing data item(s) of the database is deleted or updated.

**Definition 9 (Database State):** Given an incomplete database,  $D_I$  (refer to *Definition 1*), its state is changed to a new state,  $D_{new}$ , due to the following operations:

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

**Definition 10 (Skylines of  $D_{new}$ ):** Given a database  $D_{new}$  with  $l$  data items  $D = \{p_1, p_2, \dots, p_l\}$ ,  $p_i$  is a skyline of  $D_{new}$  if there are no other data items  $p_j \in D_{new}$  that dominates  $p_i$ . Since the focus of this paper is on incomplete and dynamic database, without loss of generality the database  $D_{new}$  is also an incomplete database as well as a dynamic database as defined in *Definition 1* and *Definition 8*, respectively and hence  $p_i \in D_I$  and  $p_i \in D_D$ .

**Definition 11 (Skylines of  $D_{new}$  due to  $D_{<insert>}$ ):** Given a database  $D$  with  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$ ,  $p_i$  is a skyline of  $D$  if there are no other data items  $p_j \in D$  that dominates  $p_i$ . Assume that the set of skylines derived based on  $D$  is  $S$ . Given the  $D_{<insert>} = \{q_1, q_2, \dots, q_q\}$ , the  $D_{new} = \{p_1, p_2, \dots, p_n, q_1, q_2, \dots, q_q\}$ , the set  $S$  is still valid if and only if for every  $p_l \in S$  there are no data items  $q_k \in D_{<insert>}$  that dominates  $p_l$ , otherwise a new set of skylines  $S'$  needs to be computed as defined in *Definition 10*.

**Definition 12 (Skylines of  $D_{new}$  due to  $D_{<delete>}$ ):** Given a database  $D$  with  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$ ,  $p_i$  is a skyline of  $D$  if there are no other data items  $p_j \in D$  that dominates  $p_i$ . Assume that the set of skylines derived based on  $D$  is  $S$ . Given  $D_{<delete>} = \{p_i, p_j, \dots, p_l\}$ , the  $D_{new} = \{p_1, p_2, \dots, p_n\} - \{p_i, p_j, \dots, p_l\}$ , the set  $S$  is still valid if and only if for every  $p_l \in D_{<delete>}$ ,  $p_l \notin S$ , otherwise a new set of skylines  $S'$  needs to be computed as defined in *Definition 10*.

**Definition 13 (Skylines of  $D_{new}$  due to an update operation):** Given a database  $D$  with  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$ ,  $p_i$  is a skyline of  $D$  if there are no other data items  $p_j \in D$  that dominates  $p_i$ . Assume that the set of skylines derived based on  $D$  is  $S$ . Since an update operation is considered as a delete operation followed by an insert operation, thus the set of skylines over  $D_{new}$  is based on the definitions given in *Definition 12* and *Definition 11*.

As defined in *Definition 9*, a database  $D$  changed its state to a new state,  $D_{new}$ , due to the following operations: (i) insert operation, (ii) delete operation, and (iii) update operation. In the following we show the significant of avoiding unnecessary skyline computations by analysing the number of pairwise comparisons after each of the above operation is performed. Here, we assume a worst-case scenario while both the best-case and average-case scenarios are presented in Table 2. Given a database,  $D$ , with  $m$  dimensions,  $d = \{d_1, d_2, \dots, d_m\}$  and  $n$  data items  $D = \{p_1, p_2, \dots, p_n\}$ , to derive a set of skylines,  $S$ , over  $D$  requires  $m \left[ \frac{n(n-1)}{2} \right]$  pairwise comparisons.

**TABLE 2. Number of pairwise comparisons based on the type of data manipulation operation.**

Case	Insert	Delete	Update
(i) Best-Case:			
$S$	$m(n-1)$	$m(n-1)$	$m(n-1)$
$S'$	$m(n+r-1)$	$m(n-l-1)$	$m(n-1)$
(ii) Average Case:			
$S$	$\frac{m(n-1)(2+n)}{4}$	$\frac{m(n-1)(2+n)}{4}$	$\frac{m(n-1)(2+n)}{4}$
$S'$	$\frac{m(n+r-1)(2+n+r)}{4}$	$\frac{m(n-l-1)(2+n-l)}{4}$	$\frac{m(n-1)(2+n)}{4}$
(iii) Worst-Case:			
$S$	$m \left[ \frac{n(n-1)}{2} \right]$	$m \left[ \frac{n(n-1)}{2} \right]$	$m \left[ \frac{n(n-1)}{2} \right]$
$S'$	$m \left[ \frac{(n+r)((n+r)-1)}{2} \right]$	$m \left[ \frac{(n-l)((n-l)-1)}{2} \right]$	$m \left[ \frac{n(n-1)}{2} \right]$

Given the  $D_{<insert>} = \{q_1, q_2, \dots, q_r\}$ , the  $D_{new} = \{p_1, p_2, \dots, p_n, q_1, q_2, \dots, q_r\}$  and to derive the set of skylines,  $S'$ , based on  $D_{new}$  requires  $m \left[ \frac{(n+r)((n+r)-1)}{2} \right]$  pairwise comparisons. However, if  $S$  has been derived based on the initial database  $D$ , this implies that the comparisons between the data items  $p_1, p_2, \dots$ , and  $p_n$  have been performed with  $m \left[ \frac{n(n-1)}{2} \right]$  pairwise comparisons. The new inserted data items incur an additional of  $\frac{r^2+2nr-r}{2}$  pairwise comparisons. For instance, if  $n = 10$ , then  $45m$  pairwise comparisons are performed to derive  $S$ , while inserting a data item say  $q_1$  with

$n + r = 11$  will involve  $55m$  pairwise comparisons with  $10m$  additional pairwise comparisons. However, the  $45m$  pairwise comparisons have been performed earlier; which in this paper are considered as the unnecessary skyline computations which should be avoided by mainly identifying those data items that are not affected by the changes.

Given the  $D_{<delete>} = \{p_i, p_j, \dots, p_l\}$  with  $|D_{<delete>}| = l$ , the  $D_{new} = \{p_1, p_2, \dots, p_n\} - \{p_i, p_j, \dots, p_l\}$  and to derive the set of skylines,  $S'$ , based on  $D_{new}$  requires  $m \left[ \frac{(n-l)(n-l-1)}{2} \right]$  pairwise comparisons. However, if  $S$  has been derived based on the initial database  $D$ , this implies that the comparisons between the data items  $p_1, p_2, \dots$ , and  $p_n$  have been performed. Hence, deriving the set of skylines,  $S'$ , based on  $D_{new}$  would incur  $m \left[ \frac{(n-l)(n-l-1)}{2} \right]$  unnecessary pairwise comparisons. For instance, if  $n = 10$ , then  $45m$  pairwise comparisons are performed to derive  $S$ , while deleting a data item say  $q_1$  will involve  $36m$  pairwise comparisons which have been performed earlier; these unnecessary pairwise comparisons should be avoided.

In this paper, an update operation is considered as a delete operation followed by an insert operation, thus the above two cases are applied here.

Our proposed solution, named *DyIn-Skyline*, is capable of deriving skylines over a database for both the complete and incomplete databases. Fig. 1 shows an example of a database with incomplete data, which is used throughout this paper to clarify the steps of our proposed solution. The database consists of five dimensions, namely: *Data Item*,  $d_1$ ,  $d_2$ ,  $d_3$ , and  $d_4$  where *Data Item* represents the unique identifier of a data item while  $d_1$ ,  $d_2$ ,  $d_3$ , and  $d_4$  are the dimensions that are to be considered in computing the set of skylines of the database. An example of a real-world database reflected by the example is the hotel or apartment database where  $d_1$ ,  $d_2$ ,  $d_3$ , and  $d_4$  are the price, distance from the city centre, service speed, and dining rate, respectively. A user might want to seek for a hotel with the minimum price, the shortest distance to the city centre, the fastest service, and the lowest dining rate per head among all the available hotels in the database. For simplicity, for each data item we limit the number of missing values to one. However, the proposed solution can handle various number of missing values. This is clearly shown in the experiments that we have conducted. Finding the set of skylines in  $D_{new}$  should incur the least number of pairwise comparisons between the data items, which will indirectly incur the least processing time. The data item  $p_i \in D_{new}$  may have missing values in one or more of its dimensions.

Table 3 summarises the symbols and notations used throughout the paper.

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$x_1$	7	—	6	6
$x_2$	3	—	5	4
$w_1$	—	5	3	3
$y_1$	2	3	—	2
$x_3$	5	—	7	7
$y_2$	6	4	—	6
$w_2$	—	1	3	1
$w_3$	—	3	2	1
$y_3$	4	4	—	6
$w_4$	—	6	6	6
$w_5$	—	3	6	6
$z_5$	3	2	2	—
$x_8$	5	—	5	5
$x_9$	1	—	3	3
$x_{10}$	3	—	3	3
$w_6$	—	4	3	5
$z_1$	5	3	5	—
$z_2$	4	5	5	—
$z_4$	7	7	6	—
$z_6$	1	1	3	—
$w_7$	—	3	2	2
$w_8$	—	5	5	5
$x_6$	2	—	1	2
$x_7$	7	—	4	6
$z_7$	2	2	2	—
$z_8$	7	4	4	—
$z_9$	1	3	2	—
$z_{10}$	1	2	3	—
$y_7$	5	4	—	5
$y_8$	3	3	—	2
$y_9$	3	6	—	7
$y_{10}$	2	1	—	2
$x_4$	5	—	3	3
$x_5$	1	—	1	5
$z_3$	6	7	5	—
$y_4$	1	3	—	3
$y_5$	3	5	—	5
$y_6$	6	3	—	4
$w_9$	—	6	4	4
$w_{10}$	—	2	1	1

FIGURE 1. An example of an incomplete database.

changes are made in order to identify the new set of skylines, is unwise as some parts of the database, i.e. those that are not affected by the changes, have been analysed. Thus, to avoid unnecessary skyline computations, it is important to capture the domination analysis that has been performed before the database is changed. Our proposed solution, named *DyIn-Skyline*, consists of two main phases, namely: *Phase I* – processing skyline queries over the initial incomplete database and *Phase II* – processing skyline queries over a dynamic and incomplete database. In *Phase I*, we introduce and design three main lists, namely: *Domination History (DH)*, *Bucket Dominating (BDG)*, and *Bucket Dominated (BDD)*, that keep track of the domination relationships, dominating data items, and dominated data items, respectively. These lists are crucial as they assist in identifying the data items that are affected by the changes made towards a database, hence excluding the unaffected data items from the process of computing skylines. In *Phase II*, we show how these lists are utilised to avoid

**TABLE 3.** List of symbols/notations.

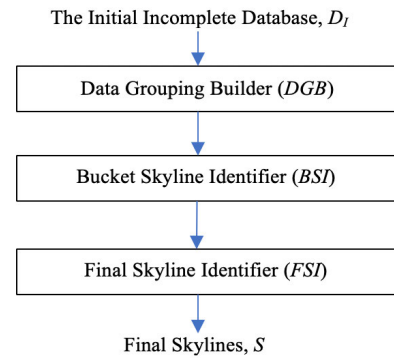
Symbols/Notations	Remarks
$D = \{p_1, p_2, \dots, p_n\}$	A database with $n$ data items
$d = \{d_1, d_2, \dots, d_m\}$	A set of $m$ dimensions
$D_I$	An incomplete database
$S = \{S_1, S_2, \dots, S_f\}$	A set of $f$ skylines, i.e. final skylines derived based on $D_I$
$S' = \{S_1, S_2, \dots, S_g\}$	A set of $g$ skylines, i.e. final skylines derived based on $D_{new}$
$D_D$	A dynamic database
$D_{new}$	The new state of a database
$D^{<insert>}$	A set of data items to be inserted into the initial database
$D^{<delete>}$	A set of data items to be deleted from the initial database
$DH$	<i>Domination History</i>
$BDG$	<i>Bucket Dominating</i>
$BDD$	<i>Bucket Dominated</i>
$DGB$	<i>Data Grouping Builder</i>
$BSI$	<i>Bucket Skyline Identifier</i>
$FSI$	<i>Final Skyline Identifier</i>
$B = \{B_1, B_2, \dots, B_s\}$	A set of $s$ buckets
$BS = \{BS_1, BS_2, \dots, BS_s\}$	A set of <i>Bucket Skylines</i>
$S-II$	<i>Skyline-Insert Identifier</i>
$S-DI$	<i>Skyline-Delete Identifier</i>
$S-UI$	<i>Skyline-Update Identifier</i>
$CSI$	<i>Candidate Skyline Identifier</i>
$TBS$	<i>Temp Bucket Skyline</i>
$TBDG$	<i>Temp Bucket Dominating</i>
$TBDD$	<i>Temp Bucket Dominated</i>
$CS = \{p_1, p_2, \dots, p_c\}$	A set of $c$ candidate skylines
$SI$	<i>Skyline Insert algorithm</i>
$DDIA$	<i>Dominated Data Items Analyser</i>
$CD$	<i>Candidate Data Items</i>
$SD$	<i>Skyline Delete algorithm</i>

unnecessary computation of skyline queries when changes are made towards the database. Part A presents the *Phase I* while Part B presents the *Phase II*.

### A. PHASE I

The *Phase I* aims at deriving a set of skylines given the initial incomplete database. This phase is performed only once. Our proposed approach is briefly explained below:

Given an initial incomplete database,  $D_I$ , a set of skylines,  $S$ , is derived. Note that  $S$  is a subset of  $D_I$ , i.e.  $S \subseteq D_I$ , and  $S$  is the set of data items that are not dominated by other data items in  $D_I - S$ , say  $\neg S$ . When there are changes in the  $D_I$ , then performing pairwise comparisons on the entire new state of  $D_I$  is unwise as comparisons between  $S$  and the initial state of  $D_I - S$  (or  $\neg S$ ) has been performed. Thus, keeping track of the data items that dominate other data items (i.e. the *dominating* data items) as well as the data items that are dominated by other data items (i.e. the *dominated* data items) can reduce the number of pairwise comparisons and processing time in the subsequent phase. Fig. 2 presents the *Phase I* of *DyIn-Skyline*. It consists of three main components, namely: *Data Grouping Builder (DGB)*, *Bucket Skyline Identifier (BSI)*, and

**FIGURE 2.** The *Phase I* of *DyIn-Skyline*.

*Final Skyline Identifier (FSI)*. Each component is explained in detail in the following paragraphs.

**Data Grouping Builder (DGB)** – We assume that the initial database is incomplete, thus each data item might have missing values in different dimensions. This will cause cyclic dominance and difficulty in preserving the transitivity property of skylines. In order to resolve the issue due to the incompleteness of data in the database, the data items should be grouped into groups in which all data items in the same group will have missing values in the same dimension(s). The *Data Grouping Builder (DGB)* groups the data items into buckets based on the missing values. This approach has been used by [2] and [23]. In this approach, each data item is represented as a bitmap representation (refer to *Definition 2 Comparable*).

Fig. 3 presents the *Data Grouping Builder (DGB)* algorithm which is embedded into the *Data Grouping Builder (DGB)* component. Each data item,  $p_i$ , of the initial incomplete database,  $D_I$ , is analysed (step 2). If the bitmap representation of  $p_i$  is the same as the bitmap representation of any existing buckets,  $B_j$ , then the data item is inserted into the bucket  $B_j$  (step 4). Otherwise, a new bucket,  $B_k$ , is created based on the bitmap representation of the data item (steps 7 – 8). Fig. 4 shows the results of the *DGB* algorithm over the initial incomplete database given in Fig. 1. After

**Input:** Initial Database,  $D_I = \{p_1, p_2, \dots, p_n\}$   
**Output:** Bucket,  $B = \{B_1, B_2, \dots, B_s\}$

1. **Begin**
2. **For** each data item  $p_i \in D_I$  **do**
3.   **If** bitmap representation of  $p_i$  = bitmap representation of existing Bucket  $B_j$  **Then**
4.      $B_j = B_j \cup p_i$
5.   **Else**
6.     **Begin**
7.       Create a Bucket  $B_k$
8.        $B_k = B_k \cup p_i$
9.     **End**
10. **End**

**FIGURE 3.** The *Data Grouping Builder (DGB)* algorithm.

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_1$	—	5	3	3
$w_2$	—	1	3	1
$w_3$	—	3	2	1
$w_4$	—	6	6	6
$w_5$	—	3	6	6
$w_6$	—	4	3	5
$w_7$	—	3	2	2
$w_8$	—	5	5	5
$w_9$	—	6	4	4
$w_{10}$	—	2	1	1

Bucket 1 (0111)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$x_1$	7	—	6	6
$x_2$	3	—	5	4
$x_3$	5	—	7	7
$x_4$	5	—	3	3
$x_5$	1	—	1	5
$x_6$	2	—	1	2
$x_7$	7	—	4	6
$x_8$	5	—	5	5
$x_9$	1	—	3	3
$x_{10}$	3	—	3	3

Bucket 2 (1011)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$y_1$	2	3	—	2
$y_2$	6	4	—	6
$y_3$	4	4	—	6
$y_4$	1	3	—	3
$y_5$	3	5	—	5
$y_6$	6	3	—	4
$y_7$	5	4	—	5
$y_8$	3	3	—	2
$y_9$	3	6	—	7
$y_{10}$	2	1	—	2

Bucket 3 (1101)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$z_1$	5	3	5	—
$z_2$	4	5	5	—
$z_3$	6	7	5	—
$z_4$	7	7	6	—
$z_5$	3	2	2	—
$z_6$	1	1	3	—
$z_7$	2	2	2	—
$z_8$	7	4	4	—
$z_9$	1	3	2	—
$z_{10}$	1	2	3	—

Bucket 4 (1110)

FIGURE 4. The results of the Data Grouping Builder (DGB).

applying the algorithm, all data items with the same bitmap representation are grouped into the same bucket.

The number of created buckets is based on the number of distinct bitmap representations of the data items which may lie between 1 where all the data items have the same bitmap representation and  $2^m - 1$  where  $m$  is the number of dimensions in which all distinct bitmap representations are possible. For instance, if  $m = 3$ , then the number of created buckets is 1 if all the data items have the same bitmap representation, else if all distinct bitmap representations are possible then 7 buckets are created (001, 010, 011, 100, 101, 110, 111). Note that the bitmap representation 000 for the above example will not occur as it represents data items with missing values in all dimensions which are impossible. The number of data items in the whole buckets is equal to the number of data items in the initial incomplete database,  $D_I$ , i.e.  $n = \sum_{i=1}^s b_i$  where  $n$  is the number of data items in  $D_I$ ,  $s$  is the number of buckets, and  $b_i$  is the number of data items in the  $i$ th bucket,  $B_i$ . This ensures that no data items are missing during this process.

**Bucket Skyline Identifier (BSI)** – After the data items have been grouped into the appropriate buckets based on their bitmap representations, then pairwise comparisons are performed on each bucket to derive the skylines of the bucket. We use the term *intra-bucket* comparisons to mean pairwise comparisons that are performed between data items in the same bucket. Note that the *Definition 2 Comparable* and *Definition 3 Dominance Relationship* are applied here. The skylines derived at this stage are the potential candidate

skylines. The *Bucket Skyline Identifier (BSI)* component has two main aims that are:

- 1) To derive the bucket skylines of each bucket by performing the intra-bucket comparisons. These bucket skylines are kept in the *Bucket Skyline (BS)*.
- 2) To keep track of the dominance relationships, a list called the *Dominance History (DH)* is created. The structure of the *DH* is as follows:  $DH = \langle \text{Dominating}, \text{Dominated} \rangle$  where *Dominating* indicates the data items that dominate other data items while *Dominated* indicates the data items that are dominated by other data items. For instance, if  $p_i > p_j$ , then  $p_i$  is the *dominating* data item while  $p_j$  is the *dominated* data item.

Fig. 5 shows the *Bucket Skyline Identifier (BSI)* algorithm which is embedded into the *Bucket Skyline Identifier (BSI)* component. It aims at deriving the bucket skylines for each bucket. It starts by analysing each bucket,  $B_k$  (step 2). The data item  $p_i$  of a bucket,  $B_k$ , is then compared to another data item,  $p_j$ , of the same bucket where the dominance relationship among them is identified based on the *Definition 3 Dominance Relationship* (step 4). In regard to the *Definition 2 Comparable*, these data items are comparable as they have the same bitmap representation. If the data item  $p_i$  dominates the data item  $p_j$ ,  $p_i > p_j$ , then  $p_i$  will be inserted into the *Bucket Skyline*,  $BS_k$  (step 6) while  $p_j$  that is dominated by  $p_i$  is removed from the bucket,  $B_k$  (step 7). We use the notation

```

Input: Bucket,  $B = \{B_1, B_2, \dots, B_s\}$ 
Output: Bucket Skylines,  $BS = \{BS_1, BS_2, \dots, BS_s\}$ 
1. Begin
2. For each Bucket  $B_k$  do
3.   Begin
4.     If  $p_i > p_j$  Then
5.       Begin
6.          $BS_k = BS_k \cup p_i$ 
7.          $B_k = B_k - p_j$ 
8.       If  $p_j \in BS_k$  Then
9.          $BS_k = BS_k - p_j$ 
10.        Insert  $\langle p_i, p_j \rangle$  into  $DH$ 
11.      End
12.    Else
13.      If  $p_j > p_i$  Then
14.        Begin
15.           $BS_k = BS_k \cup p_j$ 
16.           $B_k = B_k - p_i$ 
17.        If  $p_i \in BS_k$  Then
18.           $BS_k = BS_k - p_i$ 
19.        Insert  $\langle p_j, p_i \rangle$  into  $DH$ 
20.      End
21.    If  $p_j \nless p_i$  and  $p_i \nless p_j$  Then
22.      Begin
23.         $BS_k = BS_k \cup p_i$ 
24.         $BS_k = BS_k \cup p_j$ 
25.      End
26.    End
27. End

```

FIGURE 5. The *Bucket Skyline Identifier (BSI)* algorithm.



$BS_k$  to represent the *Bucket Skyline* of the bucket  $B_k$ . If the dominated data item  $p_j$  already exists in  $BS_k$ , then the data item  $p_j$  is removed from  $BS_k$  (steps 8 – 9). Both  $p_i$  and  $p_j$  are then inserted into the *Domination History* ( $DH$ ) (step 10) where  $p_i$  is the *dominating* data item and  $p_j$  is the *dominated* data item. However, if the data item  $p_j$  dominates the data item  $p_i$ ,  $p_j \succ p_i$ , then  $p_j$  will be inserted into the *Bucket Skyline*,  $BS_k$  (steps 13 – 15) while  $p_i$  that is dominated by  $p_j$  is removed from the bucket,  $B_k$  (step 16). If the dominated data item  $p_i$  already exists in  $BS_k$  then the data item  $p_i$  is removed from  $BS_k$  (steps 17 – 18). Both  $p_i$  and  $p_j$  are then inserted into the  $DH$  where  $p_j$  is the *dominating* data item and  $p_i$  is the *dominated* data item (step 19). However, if  $p_i$  and  $p_j$  do not dominate each other, then both data items are inserted into  $BS_k$  (steps 21 – 25).

Based on the buckets that are shown in Fig. 4, the *Bucket Skyline*,  $BS_k$ , created for each bucket,  $B_k$ , is as shown in Fig. 6(a). The  $DH$  created based on the pairwise comparisons performed on these data items is as shown in Fig. 6(b). For example, referring to the *Bucket 1* of Fig. 4, the data item  $w_1$  dominates the data items  $w_2$  and  $w_3$ . Here,  $w_1$ ,  $w_2$  and  $w_3$  are inserted into the  $DH$ , the data item  $w_1$  is inserted into the *Bucket Skyline*, and the data items  $w_2$  and  $w_3$  are deleted from the bucket. Next,  $w_1$  is compared to  $w_4$ . Here,  $w_4$  dominates  $w_1$  and both  $w_1$  and  $w_4$  are inserted into the  $DH$ .  $w_4$  is inserted into the *Bucket Skyline* while  $w_1$  which exists in the *Bucket Skyline* is removed from it. Performing the subsequent comparisons, we noticed that  $w_4$  also dominates the other data items in the *Bucket 1*, i.e.  $w_5$ ,  $w_6$ ,  $w_7$ ,  $w_8$ ,  $w_9$ , and  $w_{10}$ . The dominance relationships are kept in the  $DH$  as shown in Fig. 6(b). The same process is applied for buckets 2, 3, and 4.

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_4$	–	6	6	6

$BS_1$  (0111)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$x_1$	7	–	6	6
$x_3$	5	–	7	7

$BS_2$  (1011)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$y_2$	6	4	–	6
$y_9$	3	6	–	7

$BS_3$  (1101)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$z_4$	7	7	6	–

$BS_4$  (1110)

(a)

Dominating	Dominated
$w_1$	$\{w_2, w_3\}$
$w_4$	$\{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}\}$
$x_1$	$\{x_2, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$
$y_2$	$\{y_1, y_3, y_4, y_6, y_7, y_8, y_{10}\}$
$y_9$	$\{y_5\}$
$z_3$	$\{z_1, z_2\}$
$z_4$	$\{z_3, z_5, z_6, z_7, z_8, z_9, z_{10}\}$

(b)

**FIGURE 6.** Example of (a) *Bucket Skyline* ( $BS$ ) and (b) *Domination History* ( $DH$ ).

*Final Skyline Identifier* ( $FSI$ ) – The bucket skylines produced by the previous component only present the results of intra-bucket comparisons. Further pairwise comparisons between the data items from different buckets, i.e. *inter-bucket* comparisons, should be performed in order to derive the final skylines. Since these data items have different bitmap representations, hence the *Definition 6 Revised Bitwise* and the *Definition 7 Dominance Relationship on the Revised Bitwise* are applied here. The dominance relationships need to be kept to be used in the *Phase II*. The *Final Skyline Identifier* ( $FSI$ ) component performs the following tasks in deriving the final skylines:

- 1) Perform the inter-bucket comparisons. The pairwise comparisons performed at this stage are not straightforward as intra-bucket comparisons; inter-bucket comparisons are performed on the data items with different bitmap representations. Therefore, before these data items are compared, it is important to ensure that they are comparable. Here, the bitmap representations of the buckets are analysed and if the AND operation of the bitwise representations produced 0 then these buckets are not comparable; i.e. the *Definition 6 Revised Bitwise* is applied.
- 2) Keep track of the dominating and dominated data items. As the focus of this paper is not only on incomplete database but also on dynamic database, therefore keeping track of the produced results of this phase to be used later in the *Phase II* is important to ensure that the same processes are not repeated. The data items that dominate other data items are saved into a list named *Bucket Dominating* ( $BDG$ ). While the data items that are dominated by other data items are saved into a list named *Bucket Dominated* ( $BDD$ ). In addition, the data items that are not dominated by other data items are kept in the  $BDG$ . Since the above lists only capture the dominated data items and the dominating data items without the associated data items that dominate and data items that are dominated, respectively, then these relationships are kept in the *Domination History* ( $DH$ ) to be used in the *Phase II*.
- 3) Derive the final skylines. This task derives the final skylines by comparing the data items in the  $BDG$  to the data items in the  $BDD$ . The final skylines are the data items that appear in the  $BDG$  but do not appear in the  $BDD$ , i.e.  $\{S | S \in BDG \wedge S \notin BDD\}$ . This is because the data item that appears in both lists, i.e.  $BDG$  and  $BDD$ , is the data item that has both relationships dominating and dominated. This means although the data item dominates another data item but at the same time it is dominated by the other data items and thus should not be considered as one of the final skylines.

Fig. 7 presents the *Skyline* algorithm, which is embedded into the *Final Skyline Identifier* ( $FSI$ ) component. It aims at deriving the final skylines from the bucket skylines. It starts by checking the AND operation between the bitmap representations of a bucket skyline  $BS_k$  and another bucket skyline

**Input:** Bucket Skylines,  $BS = \{BS_1, BS_2, \dots, BS_s\}$   
**Output:** Final Skylines,  $S = \{S_1, S_2, \dots, S_f\}$

1. **Begin**
2. **If** (bitmap representation of  $BS_k$ )  $\wedge$  (bitmap representation of  $BS_l$ )  $\neq 0$  **Then**
3.   **For each** data item  $p_i \in BS_k$  **do**
4.    **For each** data item  $q_j \in BS_l$  **do**
5.     **Begin**
6.      **If**  $p_i > q_j$  **Then**
7.       **Begin**
8.          $BDG = BDG \cup p_i$
9.          $BDD = BDD \cup q_j$
10.        Insert  $\langle p_i, q_j \rangle$  into  $DH$
11.      **End**
12.     **If**  $q_j > p_i$  **Then**
13.      **Begin**
14.         $BDG = BDG \cup q_j$
15.         $BDD = BDD \cup p_i$
16.        Insert  $\langle q_j, p_i \rangle$  into  $DH$
17.      **End**
18.     **If**  $p_i \nlessdot q_j$  **and**  $q_j \nlessdot p_i$  **Then**
19.      **Begin**
20.         $BDG = BDG \cup p_i$
21.         $BDD = BDD \cup q_j$
22.      **End**
23.    **End**
24.   **For each** data item  $p_i \in BDG$  **do**
25.     **If**  $p_i \notin BDD$  **Then**
26.       $S = S \cup p_i$
27. **End**

FIGURE 7. The Skyline algorithm.

$BS_l$ . If the result of the AND operation is not equal to 0 (step 2) (refer to the *Definition 6 Revised Bitwise*), then each data item,  $p_i$ , of the bucket skyline,  $BS_k$ , and each data item,  $q_j$ , of the bucket skyline,  $BS_l$  (steps 3 – 4) are compared (refer to the *Definition 7 Dominance Relationship on the Revised Bitwise*). If the data item,  $p_i$ , dominates the data item,  $q_j$ , i.e.  $p_i > q_j$ , then  $p_i$  is inserted into the  $BDG$  while  $q_j$  is inserted into the  $BDD$  and both  $p_i$  and  $q_j$  are inserted into the  $DH$  where  $p_i$  is the *dominating* data item and  $q_j$  is the *dominated* data item (steps 6 – 11). Likewise, if the data item,  $q_j$ , dominates the data item,  $p_i$ , i.e.  $q_j > p_i$ , then  $q_j$  is inserted into the  $BDG$  while  $p_i$  is inserted into the  $BDD$  and  $q_j$  and  $p_i$  are inserted into the  $DH$  where  $q_j$  is the *dominating* data item and  $p_i$  is the *dominated* data item (steps 12 – 17). However, if  $p_i$  and  $q_j$  do not dominate each other, then both data items are inserted into the  $BDG$  (steps 18 – 22). Next for each data item  $p_i$  in the  $BDG$ , if it is not a member of the  $BDD$  then  $p_i$  is identified as one of the final skylines,  $S$  (steps 24 – 26).

Based on the bucket skylines,  $BS_i$ , shown in Fig. 6(a), the *Bucket Dominating (BDG)*, *Bucket Dominated (BDD)*, and *Dominance History (DH)* that are generated by the *Final Skyline Identifier (FSI)* component are as shown in Fig. 8. In this example, the data items  $w_4$ ,  $x_1$ ,  $x_3$ ,  $y_2$ ,  $y_9$ , and  $z_4$  are the bucket skylines of  $BS_1$ ,  $BS_2$ ,  $BS_3$ , and  $BS_4$  as shown in Fig. 6(a). These bucket skylines are compared to each other. Here,  $w_4$  dominates  $y_2$ . Thus,  $w_4$  is saved into the  $BDG$  while  $y_2$  is saved into the  $BDD$ . Also,  $x_3$  dominates  $w_4$ . Therefore,

Dominating	Dominated
$w_1$	$\{w_2, w_3\}$
$w_4$	$\{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}, y_2\}$
$x_1$	$\{x_2, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, y_2\}$
$x_3$	$\{y_9, w_4\}$
$y_2$	$\{y_1, y_3, y_4, y_6, y_7, y_8, y_{10}\}$
$y_9$	$\{y_5, w_4\}$
$z_3$	$\{z_1, z_2\}$
$z_4$	$\{z_3, z_5, z_6, z_7, z_8, z_9, z_{10}, y_9, y_2, w_4\}$

(a)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_4$	—	6	6	6
$x_3$	5	—	7	7
$y_9$	3	6	—	7
$y_2$	6	4	—	6
$z_4$	7	7	6	—
$x_1$	7	—	6	6

(b)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_4$	—	6	6	6
$y_2$	6	4	—	6
$y_9$	3	6	—	7

(c)

FIGURE 8. Example of (a) *Dominance History (DH)*, (b) *Bucket Dominating (BDG)*, and (c) *Bucket Dominated (BDD)*.

$x_3$  is saved into the  $BDG$  while  $w_4$  is saved into the  $BDD$ . On the other hand, the data item  $y_9$  dominates  $w_4$ , thus,  $y_9$  is saved into the  $BDG$ . Since  $w_4$  already exists in the  $BDD$  no further action is needed. Also,  $x_3$  dominates  $y_9$ . Hence,  $x_3$  is saved into the  $BDG$  while  $y_9$  is saved into the  $BDD$ . The data item  $z_4$  dominates  $w_4$ ,  $y_2$ , and  $y_9$  which leads to saving  $z_4$  in the  $BDG$  and as  $w_4$ ,  $y_2$ , and  $y_9$  already exist in the  $BDD$  no further action is needed. The bucket skyline  $x_1$ , does not dominate any data items and it is not being dominated by any other data items, hence  $x_1$  is saved into the  $BDG$ . After pairwise comparisons between these data items have been performed, the two lists  $BDG$  and  $BDD$  are compared and those data items that appeared in both lists will not appear in the final skylines while the remaining data items of  $BDG$  are the final skylines. These dominance relationships are kept in the  $DH$  (Fig. 8(a)). Based on this example,  $w_4$ ,  $y_2$ , and  $y_9$  that appeared in both lists are not the final skylines while  $x_1$ ,  $x_3$ , and  $z_4$  are identified as the final skylines as highlighted in Fig. 8(c).

**Theorem 1:** For every data item,  $p_i \in D_I$ , not dominated by other data items,  $p_j \in D_I$ , the *Skyline* algorithm will identify  $p_i$  as a skyline.

**Proof:** Assume a data item  $p_i \in D_I$  which is a skyline but not identified as a skyline by the *Skyline* algorithm. Based on the *Definition 3 Dominance Relationship*, *Definition 4 Skylines*, and *Definition 7 Dominance Relationship on the Revised Bitwise*,  $p_i$  is a skyline if it is not dominated by other data items. Hence, if  $p_i$  is not identified as a skyline by the *Skyline* algorithm then there is a data item, say  $p_j \in D_I$ , that dominates  $p_i$ . Based on the *Skyline* algorithm the data item  $p_j$  is either (i) a bucket skyline that dominates  $p_i$  based on the intra-bucket comparisons (*Definition 3 Dominance Relationship*) or (ii) a data item of the  $BDG$ , in which  $p_i$  is

dominated by  $p_j$  based on the inter-bucket comparisons (*Definition 7 Dominance Relationship on the Revised Bitwise*). Thus, we have two cases:

*Case 1:*  $p_j$  is a *Bucket Skyline*. Bucket skylines are produced by performing the intra-bucket comparisons. If  $p_j$  dominates  $p_i$ , then based on the *Definition 3 Dominance Relationship*,  $\forall d_k \in d, p_j.d_k \geq p_i.d_k \wedge \exists d_l \in d, p_j.d_l > p_i.d_l$  hold and  $p_j$  is kept in the *BS* while  $p_i$  is not. Since  $p_i$  is a skyline, then  $p_i$  is a bucket skyline which is kept in the *BS*; which contradicts the earlier statement.  $\square$

*Case 2:*  $p_j$  is a data item of the *BDG*. In deriving a skyline, the inter-bucket comparisons are performed. If  $p_j$  dominates  $p_i$ , then based on the *Definition 7 Dominance Relationship on the Revised Bitwise*,  $\forall d_k \in d', p_j.d_k \geq p_i.d_k \wedge \exists d_l \in d', p_j.d_l > p_i.d_l$  where  $d'$  is a set of dimensions whose revised bitwise representation is 1 hold,  $p_j$  is inserted into the *BDG* while  $p_i$  is inserted into the *BDD*. Based on the *Skyline* algorithm, the data item which is in the *BDD* is not a skyline. As  $p_i$  is a skyline, thus  $p_i$  does not appear in the *BDD*; which contradicts the earlier statement.

From *Case 1* and *Case 2*, the assumption that  $p_i$  is a skyline but not identified by the *Skyline* algorithm is invalid. Thus, the *Skyline* algorithm identifies all the skylines,  $S$ , of  $D_I$ .

## B. PHASE II

Fig. 9 shows the *Phase II* of *DyIn-Skyline*. Three components have been identified as listed below depending on the type of modification made towards a database:

(i) *Skyline-Insert Identifier (S-II)* derives a new set of skylines after a data item(s) is inserted into a database.

(ii) *Skyline-Delete Identifier (S-DI)* derives a new set of skylines after an existing data item(s) is deleted from a database.

(iii) *Skyline-Update Identifier (S-UI)* produces a new set of skylines after an existing data item(s) of a database is updated.

These components are to be invoked independently depending on the type of modify operation and there is no specific sequence among them. This means that if changes towards a database are due to a delete operation, then the *S-DI* component will be invoked without a necessity to perform the *S-II*. Similarly, if a set of mixed operations is to be performed, then each component is invoked according to the

sequence of the operations in the set. For instance, if the set of operations begins with an insert operation(s) then followed by an update operation(s), and lastly with a delete operation(s), then the following will be invoked in sequence, *S-II*, *S-UI*, and *S-DI*. The following sections present each of these components in detail.

**Skyline-Insert Identifier (S-II)**– This component derives a set of skylines when a database is changed due to an insert operation(s). Instead of performing pairwise comparisons on the new state of the database, the component identifies the skylines by analysing the information that has been established in the *Phase I*.

Given an initial incomplete database,  $D_I$ , a set of skylines derived from the *Phase I*,  $S$ , and a set of data items to be inserted  $D_{<insert>}$  into the  $D_I$ , the new state of the database is written as  $D_{new} = D_I \cup D_{<insert>}$ . As explained in Part A,  $\neg S \cup S = D_I$  and pairwise comparisons have been performed between the data items of  $\neg S \cup S$  to derive  $S$ . Given the  $D_{<insert>}$ , it is unwise to perform pairwise comparisons between all the data items of  $D_{new}$ , i.e.  $\neg S \cup S \cup D_{<insert>}$  as this means performing pairwise comparisons over  $\neg S \cup S$  again.

In our work, to derive a set of skylines given the  $D_{<insert>}$ , the *BDG* generated in the *Phase I* is utilised. *BDG* is a list of data items that is derived from the bucket skylines (*BS*). The data items in *BDG* are the data items that dominate other data items despite if they are dominated by other data items. While the data items in  $S$  are the data items that are not dominated by other data items, i.e.  $S \subseteq BDG$ . Given a database  $D_I$  with  $m$  dimensions  $d = \{d_1, d_2, \dots, d_m\}$  and a data item  $p_i \in D_I$ , where  $p_i \in BDG$  and  $p_i \notin BDD$ . When a set of data items,  $D_{<insert>}$ , is inserted into  $D_I$ , there are three cases to be considered:

*Case 1:* If a data item  $p_j \in D_{<insert>}$  dominates the data item  $p_i$ , then  $p_j$  has a potential to be a skyline.

*Case 2:* If the data item  $p_i$  dominates a data item  $p_j \in D_{<insert>}$ , then  $p_j$  is not a skyline.

*Case 3:* If the data items  $p_i$  and  $p_j \in D_{<insert>}$  do not dominate each other, then both  $p_i$  and  $p_j$  have a potential to be a skyline.

These cases are visualised in Fig. 10.

Fig. 11 presents the subcomponents of the *Skyline-Insert Identifier (S-II)*. The subcomponents are *Data Grouping Builder (DGB)*, *Bucket Skyline Identifier (BSI)*, *Candidate Skyline Identifier (CSI)*, and *Final Skyline Identifier (FSI)*. These subcomponents are explained in details in the following paragraphs. Meanwhile, Fig. 12 shows a sample of data items to be inserted,  $D_{<insert>}$ , into the database,  $D_I$ . We will use this sample as an example to clarify the steps of each subcomponent of the *S-II*.

**Data Grouping Builder (DGB)** – This subcomponent groups the data items into buckets based on the bitmap representation of the data items. It uses the same approach and applies the *Data Grouping Builder (DGB)* algorithm presented in Fig. 3. Thus, we will not further elaborate this subcomponent. Based on the  $D_{<insert>}$  given in Fig. 12 and

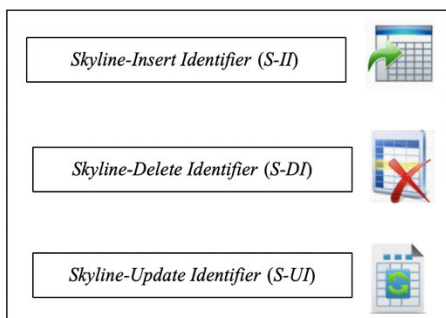


FIGURE 9. The *Phase II* of *DyIn-Skyline*.

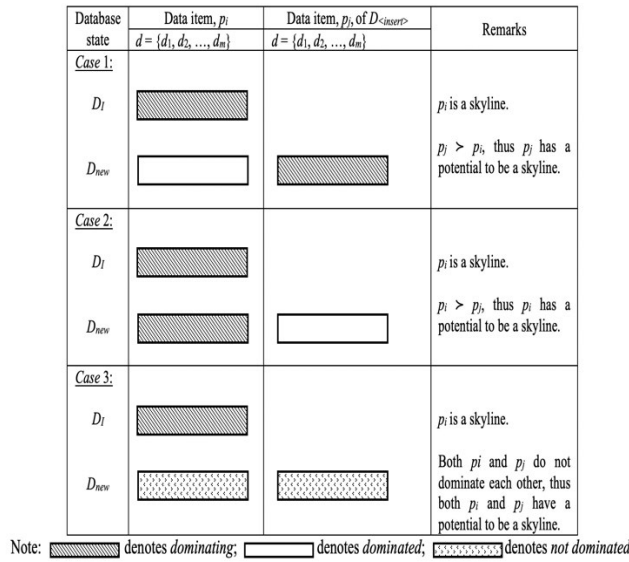
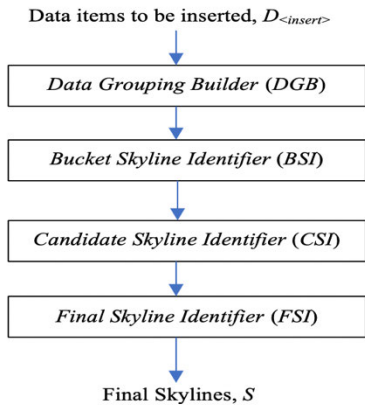
FIGURE 10. The three cases of  $D_{<insert>}$ .

FIGURE 11. The subcomponents of the Skyline-Insert Identifier (S-II).

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$z_{11}$	5	2	5	—
$y_{11}$	8	3	—	5
$w_{13}$	—	1	3	1
$y_{14}$	3	2	—	1
$z_{12}$	3	5	7	—
$x_{15}$	6	—	5	5
$y_{13}$	6	5	—	3
$z_{13}$	6	7	5	—
$w_{12}$	—	7	8	6

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$x_{14}$	5	—	2	3
$z_{14}$	3	1	4	—
$x_{13}$	3	—	7	7
$y_{12}$	7	1	—	4
$x_{12}$	6	—	3	4
$w_{11}$	—	4	6	7
$x_{11}$	8	—	1	6
$w_{14}$	—	4	6	3

FIGURE 12. Example of  $D_{<insert>}$ .

applying the *DGB* algorithm, four buckets are produced as shown in Fig. 13. Each bucket contains the data items with the same bitmap representation. For example, *Bucket 1* with bitmap representation 0111 has data items of  $D_{<insert>}$  with missing values in the first dimension,  $d_1$ .

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_{11}$	—	4	6	7
$w_{12}$	—	7	8	6
$w_{13}$	—	1	3	1
$w_{14}$	—	4	6	3

Bucket 1 (0111)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$x_{11}$	8	—	1	6
$x_{12}$	6	—	3	4
$x_{13}$	3	—	7	7
$x_{14}$	5	—	2	3
$x_{15}$	6	—	5	5

Bucket 2 (1011)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$y_{11}$	8	3	—	5
$y_{12}$	7	1	—	4
$y_{13}$	6	5	—	3
$y_{14}$	3	2	—	1

Bucket 3 (1101)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$z_{11}$	5	2	5	—
$z_{12}$	3	5	7	—
$z_{13}$	6	7	5	—
$z_{14}$	3	1	4	—

Bucket 4 (1110)

FIGURE 13. The results of the Data Grouping Builder (DGB) based on  $D_{<insert>}$ .

*Bucket Skyline Identifier (BSI)* – Similar to the *Bucket Skyline Identifier (BSI)* component presented in Part A, the subcomponent *Bucket Skyline Identifier (BSI)* in this phase aims at deriving the bucket skylines of each bucket and keeping track of the list of dominating and dominated data items in the *Domination History (DH)*. It uses the same approach and applies the *Bucket Skyline Identifier (BSI)* algorithm presented in Fig. 5. Thus, we will not further elaborate this subcomponent. Based on the buckets produced in Fig. 13 and applying the *BSI* algorithm, the bucket skylines of each bucket are saved into the *Temp Bucket Skyline (TBS)* as shown in Fig. 14 while the *DH* created during the process of pairwise comparisons is presented in Fig. 15(a).

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_{11}$	—	4	6	7
$w_{12}$	—	7	8	6

TBS1 (0111)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$x_{11}$	8	—	1	6
$x_{13}$	3	—	7	7
$x_{15}$	6	—	5	5

TBS2 (1011)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$y_{11}$	8	3	—	5
$y_{13}$	6	5	—	3

TBS3 (1101)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$z_{12}$	3	5	7	—
$z_{13}$	6	7	5	—

TBS4 (1110)

FIGURE 14. Temp Bucket Skyline (TBS).

*Candidate Skyline Identifier (CSI)* – This subcomponent performs similar tasks as the *Final Skyline Identifier (FSI)* component presented in Part A. Three tasks are performed that are: (i) performing the inter-bucket comparisons, (ii) keeping track of the dominating and dominated data items, and (iii) deriving the candidate skylines.

Based on the bucket skylines,  $TBS_i$ , that are shown in Fig. 14, the *Temp Bucket Dominating (TBDG)*, *Temp Bucket Dominated (TBDD)*, and *Domination History (DH)* that are generated by the *Candidate Skyline Identifier (CSI)* subcomponent are as shown in Fig. 15. The characteristic and the



Dominating	Dominated
$w_1$	$\{w_2, w_3\}$
$w_4$	$\{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{12}\}$
$x_1$	$\{x_2, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, y_2\}$
$x_3$	$\{y_9, w_4\}$
$y_2$	$\{y_1, y_3, y_4, y_6, y_7, y_8, y_{10}\}$
$y_9$	$\{y_5, w_4\}$
$z_3$	$\{z_1, z_2\}$
$z_4$	$\{z_3, z_5, z_6, z_7, z_8, z_9, z_{10}, y_2, y_9, w_4\}$
$y_{11}$	$\{y_{12}, y_{14}, x_{15}\}$
$w_{11}$	$\{w_{13}, w_{14}, x_{11}, x_{15}, y_{11}\}$
$w_{12}$	$\{z_{13}, z_{12}, x_{11}, x_{15}, y_{11}, y_{13}\}$
$x_{12}$	$\{x_{14}\}$
$x_{15}$	$\{x_{12}, y_{13}\}$
$x_{13}$	$\{w_{11}\}$
$x_{11}$	$\{y_{11}, y_{13}\}$
$y_{13}$	$\{z_{12}\}$
$z_{13}$	$\{z_{11}, y_{13}\}$
$z_{12}$	$\{z_{14}, w_{11}\}$

(a)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_{12}$	—	7	8	6
$y_{11}$	8	3	—	5
$x_{13}$	3	—	7	7
$w_{11}$	—	4	6	7
$x_{11}$	8	—	1	6
$y_{13}$	6	5	—	3
$z_{12}$	3	5	7	—
$z_{13}$	6	7	5	—
$x_{15}$	6	—	5	5

(b)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$y_{11}$	8	3	—	5
$y_{13}$	6	5	—	3
$w_{11}$	—	4	6	7
$x_{11}$	8	—	1	6
$z_{12}$	3	5	7	—
$z_{13}$	6	7	5	—
$x_{15}$	6	—	5	5

(c)

**FIGURE 15. (a) Domination History (DH), (b) Temp Bucket Dominating (TBDG), and (c) Temp Bucket Dominated (TBDD).**

process of producing the TBDG, TBDD are the same as the BDG and BDD presented in Part A but the only difference is that the TBDG and TBDD are produced based on the inserted data items,  $D_{<insert>}$ . The aim of having the TBDG and TBDD lists is to hasten the process of identifying the candidate skylines of  $D_{<insert>}$ . We do not append the results of dominance relationships produced in this step into the existing BDG and BDD although both lists have the same role as the TBDG and TBDD, as doing this will incur unnecessary comparisons between the data items of BDG and BDD. These comparisons have been performed in the Phase I when the set of skylines is derived. However, once the candidate skylines have been identified, i.e. those data items that are listed in the TBDG but not in the TBDD, the contents of TBDG and TBDD are emptied, ready to be used by the next modify operation, if any.

**Final Skyline Identifier (FSI)** – This subcomponent derives the set of skylines by comparing the data items of BDG against the data items of Candidate Skyline (CS) produced in the previous step. The BDG is derived in the Phase I. Fig. 16 depicts the Skyline Insert (SI) algorithm, which is embedded into the Final Skyline Identifier (FSI)

**Input:** Candidate Skylines,  $CS = \{p_1, p_2, \dots, p_c\}$ ; *Bucket Dominating*,  $BDG = \{q_1, q_2, \dots, q_b\}$   
**Output:** Final Skylines,  $S$

```

1. Begin
2.   For each data item  $p_i \in CS$  do
3.     For each data item  $q_j \in BDG$  do
4.       Begin
5.         If  $p_i > q_j$  Then
6.           Begin
7.              $BDG = BDG \cup p_i$ 
8.              $BDD = BDD \cup q_j$ 
9.             Insert  $\langle p_i, q_j \rangle$  into  $DH$ 
10.          End
11.         Else
12.           If  $q_j > p_i$  Then
13.             Begin
14.                $BDG = BDG \cup q_j$ 
15.                $BDD = BDD \cup p_i$ 
16.               Insert  $\langle q_j, p_i \rangle$  into  $DH$ 
17.            End
18.           Else
19.             If  $q_j \neq p_i$  and  $p_i \neq q_j$  Then
20.               Begin
21.                  $BDG = BDG \cup p_i$ 
22.                  $BDG = BDG \cup q_j$ 
23.               End
24.             End
25.           For each data item  $p_i \in BDG$  do
26.             If  $p_i \notin BDD$  then
27.                $S = S \cup p_i$ 
28.         End

```

**FIGURE 16. The Skyline Insert (SI) algorithm.**

subcomponent. It aims at deriving the set of skylines,  $S$ . The SI algorithm has similar steps to the Skyline algorithm shown in Fig. 7 and explained in the Part A. The difference between these two algorithms is the inputs to the SI algorithm are the list of candidate skylines derived by the Candidate Skyline Identifier (CSI) subcomponent and the BDG and BDD lists.

Fig. 15(b) shows the CS which is derived by the CSI subcomponent. It contains two data items that are  $w_{12}$  and  $x_{13}$ . These two data items are compared to the data items of BDG that are  $w_4, x_1, x_3, z_4$ , and  $y_9$  as shown in Fig. 8(b). Here,  $w_{12}$  dominates  $w_4, z_4$ , and  $x_1$ ;  $x_{13}$  dominates  $x_{13}$  while  $x_{13}$  dominates  $w_4$ . This will result in  $w_{12}$  and  $x_{13}$  being saved into the BDG while  $x_1$  and  $x_{13}$  are saved into the BDD. These dominance relationships are saved into the DH. Figs 17(a), (b), and (c) present the updated DH, BDG, and BDD, respectively. The set of skylines are derived based on the updated BDD and BDG by excluding the common data items that appeared in both the BDG and BDD. The set of skylines,  $S$ , for this example is  $S = \{w_{12}, x_3\}$  as highlighted in Fig. 17(b).

**Theorem 2:** If the data item  $p_i \in D_{new}$  is a skyline, then  $p_i$  is derived as a skyline by the Skyline Insert (SI) algorithm.

**Proof:** Assume that  $p_i$  is a skyline of  $D_{new}$ , but  $p_i$  is not identified as a skyline by the SI algorithm. The data item  $p_i$  is not a skyline if it is dominated by other data items,  $p_j \in D_{new}$ ,

Dominating	Dominated
$w_1$	$\{w_2, w_3\}$
$w_4$	$\{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}, y_2\}$
$x_1$	$\{x_2, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, y_2\}$
$x_3$	$\{y_9, w_4, x_{13}\}$
$y_2$	$\{y_1, y_3, y_4, y_6, y_7, y_8, y_{10}\}$
$y_9$	$\{y_5, w_4\}$
$z_3$	$\{z_1, z_2\}$
$z_4$	$\{z_3, z_5, z_6, z_7, z_8, z_9, z_{10}, y_2, y_9, w_4\}$
$y_{11}$	$\{y_{12}, y_{14}, x_{15}\}$
$w_{11}$	$\{w_{13}, w_{14}, x_{11}, x_{15}, y_{11}\}$
$w_{12}$	$\{w_{13}, w_{14}, z_{13}, z_{12}, x_{11}, x_{15}, y_{11}, y_{13}, x_1, w_4, z_4\}$
$x_{12}$	$\{x_{14}\}$
$x_{15}$	$\{x_{12}, x_{14}, y_{13}\}$
$x_{13}$	$\{w_{11}, w_4\}$
$x_{11}$	$\{y_{11}, y_{13}\}$
$y_{13}$	$\{y_{14}, z_{12}\}$
$z_{11}$	$\{z_{14}\}$
$z_{13}$	$\{z_{11}, z_{14}, y_{13}\}$
$z_{12}$	$\{z_{14}, w_{11}\}$
$w_{14}$	$\{w_{13}\}$

(a)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_4$	—	6	6	6
$x_1$	7	—	6	6
$x_3$	5	—	7	7
$y_9$	3	6	—	7
$z_4$	7	7	6	—
$w_{12}$	—	7	8	6
$x_{13}$	3	—	7	7

(b)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_4$	—	6	6	6
$y_2$	6	4	—	6
$y_9$	3	6	—	7
$z_4$	7	7	6	—
$x_1$	7	—	6	6
$x_{13}$	3	—	7	7

(c)

**FIGURE 17. (a) Domination History (DH), (b) Bucket Dominating (BDG), and (c) Bucket Dominated (BDD).**

either during the intra-bucket or inter-bucket comparisons. Thus, we have two cases:

**Case 1:** Assume that  $p_i$  is not a skyline and  $p_i$  is dominated by a data item say  $p_j \in D_{<insert>}$  during the intra-bucket comparisons, i.e.  $p_i \in D_{<insert>}$ . According to the *Definition 3 Dominance Relationship*,  $\forall d_k \in d, p_j.d_k \geq p_i.d_k \wedge \exists d_l \in d, p_j.d_l > p_i.d_l$ . However, based on the *Definition 4 Skylines*, if  $p_i$  is a skyline, then  $p_i$  is not dominated by  $p_j$ , which contradicts the earlier statement.  $\square$

**Case 2:** Assume that  $p_i$  is not a skyline and  $p_i$  is dominated by a data item say  $p_j \in D_{<insert>}$  during the inter-bucket comparisons. Thus, either  $p_j \in CS$  or  $p_j \in BDG$ . According to the *Definition 7 Dominance Relationship on the Revised Bitwise*,  $\forall d_k \in d', p_j.d_k \geq p_i.d_k \wedge \exists d_l \in d', p_j.d_l > p_i.d_l$  where  $d'$  is a set of dimensions whose revised bitwise representation is 1. However, based on the *Definition 4 Skylines*, if  $p_i$  is a skyline, then  $p_i$  is not dominated by  $p_j$ , which contradicts the earlier statement.  $\square$

From *Case 1* and *Case 2*, the assumption that  $p_i$  is a skyline but not identified by the *SI* algorithm is invalid. Hence, the *SI* algorithm derives all the skylines,  $S$ , of  $D_{new}$ .

**Skyline-Delete Identifier (S-DI)**— This component derives a set of skylines when a database is changed due to a delete operation(s). Instead of performing pairwise comparisons on the new state of a database, the component identifies the set of skylines by analysing the information that has been established in the *Phase I*.


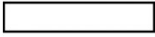
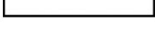
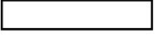


Given the initial incomplete database,  $D_I$ , the set of skylines derived from the *Phase I*,  $S$ , and a set of data items to be deleted  $D_{<delete>}$  from the  $D_I$ , the new state of the database is written as  $D_{new} = D_I - D_{<delete>}$ . This means  $D_{new} \subset D_I$  and  $D_I = D_{new} \cup D_{<delete>}$ . As explained in Part A,  $D_I = \neg S \cup S$  which also means that  $D_I = D_{new} \cup D_{<delete>}$ . Hence, pairwise comparisons have been performed between the data items of  $\neg S \cup S$  or  $D_{new} \cup D_{<delete>}$  to derive  $S$ . Given the  $D_{<delete>}$ , it is unwise to perform pairwise comparisons between all the data items of  $D_{new}$ , i.e.  $\neg S \cup S - D_{<delete>}$ , as this means performing pairwise comparisons over  $\neg S \cup S - D_{<delete>}$  again.



In our work, to derive a set of skylines,  $S$ , given the  $D_{<delete>}$ , the recent *DH* generated by the previous process is utilised. *DH* is the list of dominated and dominating data items, which keeps track of the dominance relationships. The data items which are dominated by the data items in  $D_{<delete>}$  are examined. Given a database  $D_I$  with  $m$  dimensions  $d = \{d_1, d_2, \dots, d_m\}$  and a data item,  $p_i \in D_I$ . When a set of data items,  $D_{<delete>}$ , is deleted from  $D_I$ , if the data item  $p_i \in D_{<delete>}$ , there are two cases to be considered:

**Case 1:** If  $p_i \in BDG$  and  $p_i \notin BDD$ , i.e.  $p_i$  is a skyline of  $D_I$ , and  $p_i$  dominates a data item  $p_j \in D_I$  while  $p_j \notin D_{<delete>}$  then  $p_j$  has a potential to be a skyline.

**Case 2:** If  $p_i$  is not a skyline of  $D_I$  and is dominated by a data item  $p_j \in D_I$  while  $p_j \notin D_{<delete>}$  then  $p_j$  has a potential to be a skyline.

These cases are visualised in Fig. 18.

Database state	Data item, $p_i$ $d = \{d_1, d_2, \dots, d_m\}$	Data item, $p_j$ $d = \{d_1, d_2, \dots, d_m\}$	Remarks
<b>Case 1:</b>			
$D_I$			$p_i > p_j$ , $p_i$ is a skyline.
$D_{new}$			$p_j$ has a potential to be a skyline, when $p_i$ is deleted.
<b>Case 2:</b>			
$D_I$			$p_j > p_i$ .
$D_{new}$			$p_j$ has a potential to be a skyline.

Note:  denotes dominating;  denotes dominated

**FIGURE 18. The two cases of  $D_{<delete>}$ .**

Fig. 19 presents the subcomponents of the *Skyline-Delete Identifier (S-DI)*. The subcomponents are *Dominated Data Items Analyser (DDIA)* and *Final Skyline Identifier (FSI)*. These subcomponents are explained in details in the following subsections.

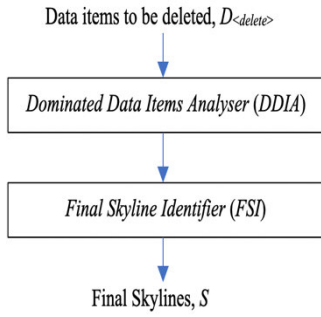


FIGURE 19. The subcomponents of the Skyline-Delete Identifier (S-DI).

Fig. 20 shows a sample of data items to be deleted,  $D_{<delete>}$ , from the database,  $D_I$ . We will use this sample as an example to clarify the steps of each subcomponent of S-DI.

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_4$	—	6	6	6
$z_4$	7	7	6	—
$x_1$	7	—	6	6

FIGURE 20. Example of  $D_{<delete>}$ .

**Dominated Data Items Analyser (DDIA)** – As explained above, the effect of deleting a data item will affect the present skylines. This is due to the fact that the data items dominated by the deleted data items have chances to be skylines. The aim of this subcomponent is to identify these dominated data items and perform pairwise comparisons between them instead of analysing the entire database to avoid unnecessary skyline computations, i.e. to reduce the number of pairwise comparisons to be performed. In our work, the *Domination History* (DH) is analysed to identify the dominated data items. Since the data items that are dominated by the deleted data items have the potential to be skylines, thus these data items are saved into a temporary list called  $CD$ . The data items to be deleted are then removed from the database,  $BDD$ ,  $BDG$ , and  $DH$ . The data items that are in the  $CD$  are compared to each other. The data items that dominate other data items are stored in the  $TBDG$  while the dominated data items are removed from the  $CD$  and saved into the  $TBDD$ . The  $TBDD$  and  $TBDG$  are used by the next subcomponent.

Fig. 21 presents the *Restore* algorithm which is embedded into the *Dominated Data Items Analyser (DDIA)* subcomponent. It aims at identifying the data items that have potential to be skylines when some data items are deleted from a database. It starts by analysing each data item to be deleted  $p_i$  (step 2). It also analyses the  $DH$  to identify the list of data items,  $L$ , that are dominated by  $p_i$  (step 3) which are then saved into the  $CD$  (step 4). The data item  $p_i$  is then removed from the database,  $DH$ ,  $BDD$ , and  $BDG$  (steps 5–7). The data items saved in the  $CD$  are then compared to each other. If  $q_i$  dominates  $q_j$ ,  $q_i > q_j$ , then  $q_i$  is inserted into the  $TBDG$  and  $DH$  while  $q_j$  is removed from the  $CD$  and inserted into the  $TBDD$  and  $DH$

```

Input: Data Items to be Deleted,  $D_{<delete>} = \{p_1, p_2, \dots, p_d\}$ 
Output: Candidate Data Items,  $CD = \{q_1, q_2, \dots, q_c\}$ 
1. Begin
2. For each data item  $p_i \in D_{<delete>}$  do
3.   Let  $L = \{q_1, q_2, \dots, q_l\}$ , the dominated data items of  $p_i$  from  $DH$ 
4.    $CD = CD \cup L$ 
5.   Remove  $p_i$  from  $DH$ 
6.    $BDD = BDD - p_i$ 
7.    $BDG = BDG - p_i$ 
8.   For each data item  $q_i \in CD$  do
9.     For each data item  $q_j \in CD$  do
10.      Begin
11.      If  $q_i > q_j$  Then
12.        Begin
13.         $CD = CD - q_j$ 
14.         $TBDG = TBDG \cup q_i$ 
15.         $TBDD = TBDD \cup q_j$ 
16.        Insert  $\langle q_i, q_j \rangle$  into  $DH$ 
17.      End
18.      If  $q_j > q_i$  Then
19.        Begin
20.         $CD = CD - q_i$ 
21.         $TBDG = TBDG \cup q_j$ 
22.         $TBDD = TBDD \cup q_i$ 
23.        Insert  $\langle q_j, q_i \rangle$  into  $DH$ 
24.      End
25.    End
26.  End
  
```

FIGURE 21. The Restore algorithm.

(steps 11–17). Likewise, if  $q_j$  dominates  $q_i$ ,  $q_j > q_i$ , then  $q_j$  is inserted into the  $TBDG$  and  $DH$  while  $q_i$  is removed from the  $CD$  and inserted into the  $TBDD$  and  $DH$  (steps 18–24).

Assume that we want to delete the data items  $w_4$ ,  $z_4$ , and  $x_1$  as shown in Fig. 20 from the initial database,  $D_I$ . The  $DH$  is checked to identify the data items that are dominated by  $w_4$ ,  $z_4$ , and  $x_1$ . Based on the  $DH$ , the data items that are dominated by  $w_4$  are  $w_1$ ,  $w_5$ ,  $w_6$ ,  $w_7$ ,  $w_8$ ,  $w_9$ ,  $w_{10}$ , and  $y_2$ ; the data items that are dominated by  $z_4$  are  $z_3$ ,  $z_5$ ,  $z_6$ ,  $z_7$ ,  $z_8$ ,  $z_9$ ,  $z_{10}$ ,  $y_9$ ,  $y_2$ , and  $w_4$ ; while the data items that are dominated by  $x_1$  are  $x_2$ ,  $x_4$ ,  $x_5$ ,  $x_6$ ,  $x_7$ ,  $x_8$ ,  $x_9$ ,  $x_{10}$ , and  $y_2$ . These data items that are dominated and not in the list of deleted data items are saved in the  $CD$  as shown in Fig. 23(a).  $w_4$ ,  $z_4$ , and  $x_1$  are then deleted from the  $DH$ ,  $BDG$ ,  $BDD$ , and *skyline*,  $S$ . The updated  $DH$  and  $BDG$  are shown in Figs 22(a) and 22(b), respectively. The data items in  $CD$  are then compared to each other. The data items that are dominated are removed from the  $CD$ , i.e. all the data items in the  $CD$  except for  $x_7$  and  $z_3$ . Those data items that dominate other data items which are  $w_8$ ,  $w_9$ ,  $y_2$ ,  $x_7$ ,  $x_8$ ,  $z_8$ , and  $z_3$  are saved in the  $TBDG$  as shown in Fig. 23(b) while the dominated data items are saved in the  $TBDD$  as shown in Fig. 23(c).

**Final Skyline Identifier (FSI)** – This subcomponent identifies the set of skylines when data items are deleted from the database. This is done by performing pairwise comparisons between the data items in the  $CD$  and  $BDG$ . Fig. 24 presents the steps of the *Skyline Delete (SD)* algorithm, which aims at deriving a set of skylines,  $S$ , given a set of data items to

Dominating	Dominated
$w_1$	$\{w_2, w_3\}$
$x_3$	$\{y_9\}$
$y_2$	$\{y_1, y_3, y_4, y_6, y_7, y_8, y_{10}\}$
$y_9$	$\{y_5\}$
$z_3$	$\{z_1, z_2\}$

(a)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$x_3$	5	—	7	7
$y_9$	3	6	—	7

(b)

**FIGURE 22.** Updated (a) *Domination History (DH)* and (b) *Bucket Dominating (BDG)*.

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_1$	—	5	3	3
$w_5$	—	3	6	6
$w_6$	—	4	3	5
$w_7$	—	3	2	2
$w_8$	—	5	5	5
$w_9$	—	6	4	4
$w_{10}$	—	2	1	1
$y_2$	7	4	—	6
$z_3$	6	7	5	—
$z_5$	3	2	2	—
$z_6$	1	1	3	—
$z_7$	2	2	2	—

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$z_8$	7	4	4	—
$z_9$	1	3	2	—
$z_{10}$	1	2	3	—
$y_9$	3	6	—	7
$x_2$	3	—	5	4
$x_4$	5	—	3	3
$x_5$	1	—	1	5
$x_6$	2	—	1	2
$x_7$	7	—	4	6
$x_8$	5	—	5	5
$x_9$	1	—	3	3
$x_{10}$	3	—	3	3

(a)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_8$	—	5	5	5
$w_9$	—	6	4	4
$x_8$	5	—	5	5
$y_2$	6	4	—	6
$x_7$	7	—	4	6
$z_3$	6	7	5	—
$z_8$	7	4	4	—

(b)

Data Item	$d_1$	$d_2$	$d_3$	$d_4$
$w_8$	—	5	5	5
$w_9$	—	6	4	4
$x_8$	5	—	5	5
$y_2$	6	4	—	6
$z_8$	7	4	4	—

(c)

**FIGURE 23.** (a) The Candidate Data Items, (b) Temp Bucket Dominating (TBDG), and (c) Temp Bucket Dominated (TBDD).

be deleted,  $D_{<delete>}$  from the database  $D_I$ . The algorithm starts by analysing the data items  $p_i$  of  $CD$  and  $q_j$  of  $BDG$  (steps 2 – 3). If the data item,  $p_i$ , dominates the data item,  $q_j$ ,  $p_i > q_j$ , then  $p_i$  is inserted into the  $BDG$  and  $DH$  while  $q_j$  is inserted into the  $BDD$  and  $DH$  (steps 5 – 10). Likewise, if the data item,  $q_j$ , dominates the data item,  $p_i$ ,  $q_j > p_i$ , then  $q_j$  is inserted into the  $BDG$  and  $DH$  while  $p_i$  is inserted into the  $BDD$  and  $DH$  (steps 12 – 17). However, if  $p_i$  and  $q_j$  do not dominate each other, then both data items are inserted into the  $BDG$  (steps 19 – 23). Then the  $TBDG$  list is merged with the  $BDG$  while the  $TBDD$  with the  $BDD$  (steps 25– 26). Next for each data item  $p_i$  in  $BDG$  if it is not a member of  $BDD$ , then  $p_i$  is inserted into the skylines,  $S$  (steps 27 – 29).

The running example for the Skyline Delete ( $SD$ ) algorithm is shown in Fig. 25. The data items in the updated  $CD$  shown in Fig. 23(a) (data items that are being highlighted) are

**Input:** Candidate Data Items,  $CD = \{p_1, p_2, \dots, p_c\}$ ; *Bucket Dominating*,  $BDG = \{q_1, q_2, \dots, q_b\}$   
**Output:** Final Skylines,  $S$

1. **Begin**
2. **For** each data item  $p_i \in CD$  **do**
3.   **For** each data item  $q_j \in BDG$  **do**
4.     **Begin**
5.       **If**  $p_i > q_j$  **Then**
6.         **Begin**
7.            $BDG = BDG \cup p_i$
8.            $BDD = BDD \cup q_j$
9.           Insert  $\langle p_i, q_j \rangle$  into  $DH$
10.       **End**
11.       **Else**
12.         **If**  $q_j > p_i$  **Then**
13.         **Begin**
14.            $BDG = BDG \cup q_j$
15.            $BDD = BDD \cup p_i$
16.           Insert  $\langle q_j, p_i \rangle$  into  $DH$
17.         **End**
18.       **Else**
19.         **If**  $p_i \neq q_j$  and  $q_j \neq p_i$  **Then**
20.         **Begin**
21.            $BDG = BDG \cup p_i$
22.            $BDG = BDG \cup q_j$
23.         **End**
24.       **End**
25.        $BDG = BDG \cup TBDG$
26.        $BDD = BDD \cup TBDD$
27.       **For** each data item  $p_i \in BDG$  **do**
28.         **If**  $p_i \notin BDD$  **Then**
29.            $S = S \cup p_i$
30. **End**

**FIGURE 24.** The Skyline Delete ( $SD$ ) algorithm.

compared to the data items in  $BDG$  presented in Fig. 22(b). After comparing the data items, the updated  $BDG$  and  $BDD$  are as shown in Figs 25(b) and 25(c), respectively. The set of skylines for this example after deleting the data items  $w_4$ ,  $z_4$ , and  $x_1$  is  $S = \{x_3, x_7, z_3\}$ .

**Theorem 3:** If the data item  $p_i \in D_{new}$  is a skyline, then  $p_i$  is identified as a skyline by the Skyline Delete ( $SD$ ) algorithm.

**Proof:** Assume that  $p_i$  is a skyline of  $D_{new}$ , but  $p_i$  is not identified as a skyline by the  $SD$  algorithm. The data item  $p_i$  is not a skyline if it is dominated by other data items,  $p_j \in D_{new}$ . Three cases need to be considered as follows:

**Case 1:** If  $p_i$  is a skyline of  $D_{new}$ , then  $p_i \in BDG$  and  $p_i \notin BDD$  as presented in steps 27 – 29 of the  $SD$  algorithm. Based on Definition 3 Dominance Relationship,  $\forall d_k \in d, p_j.d_k \geq p_i.d_k \wedge \exists d_l \in d, p_j.d_l > p_i.d_l$ , which implies that  $p_j \in BDG$  and  $p_j \notin BDD$  while  $p_i \in BDD$  which contradicts with the earlier statement.  $\square$

**Case 2:** If  $p_i$  is a skyline of  $D_{new}$ , then  $p_i \in$  Bucket Skyline. Based on Definition 3 Dominance Relationship,  $\forall d_k \in d, p_j.d_k \geq p_i.d_k \wedge \exists d_l \in d, p_j.d_l > p_i.d_l$  and steps 27 – 29 of the  $SD$  algorithm,  $p_j \in BDG$  and  $p_j \notin BDD$  while  $p_i \in BDD$ . Since  $p_i \in BDD$  and based on steps 27 – 29 of the  $SD$  algorithm  $p_i$  is not a skyline which contradicts with the earlier statement.  $\square$



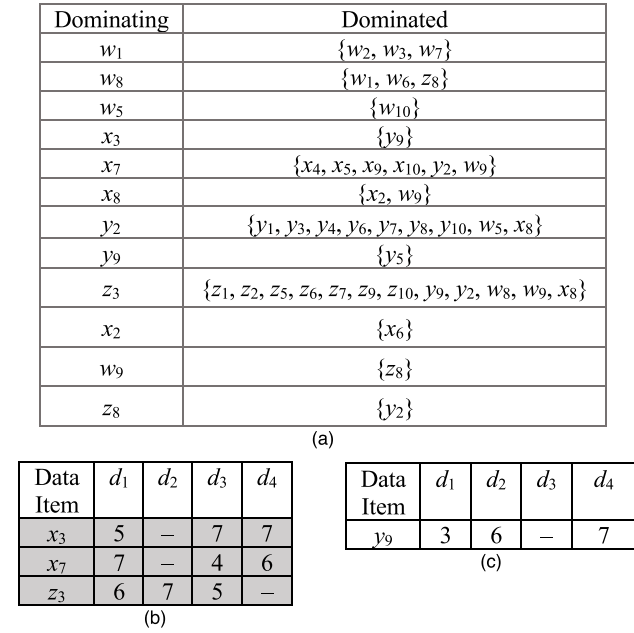


FIGURE 25. (a) Updated Domination History (DH) (b) Bucket Dominating (BDG), and (c) Bucket Dominated (BDD).

Case 3: Assume that  $p_j \in CD$ , based on steps 2 – 3 of the  $SD$  algorithm,  $p_j$  is compared to each data item of  $BDG$ , in which  $p_j \in BDG$  as it dominates  $p_i$  and  $p_i \in BDD$ , hence  $p_i$  is not a skyline which contradicts with the earlier statement.  $\square$

From the above cases, the assumption that  $p_i$  is a skyline but is not derived by the  $SD$  algorithm is invalid. Hence,  $SD$  algorithm derives all the skylines,  $S$  of  $D_{new}$ .

**Skyline-Update Identifier (S-UI)**– This component derives a set of skylines when a database is changed due to an update operation(s). An update operation in our work is achieved by performing a delete operation followed by an insert operation. Hence, the component  $S-DI$  is performed followed by the component  $S-II$  as explained earlier. In other words, The components  $S-DI$  and  $S-II$  are part of the  $S-UI$  as shown in Fig. 26.

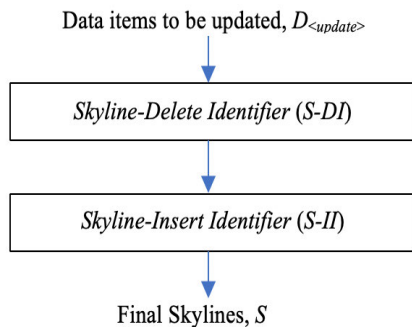


FIGURE 26. The subcomponents of the Skyline-Update Identifier (S-UI).

## V. RESULTS AND DISCUSSION

### A. EXPERIMENTAL SETTINGS

To fairly evaluate the performance and prove the efficiency of our proposed solution, *DyIn-Skyline*, in processing skyline queries over a dynamic and incomplete database, several

extensive experiments are designed. These experiments are conducted on Intel Core i7 3.6GHz processor with 32GB of RAM and Windows 8 professional. The implementation of *DyIn-Skyline* was done in VB.NET 2013. Since this study is the first attempt that investigates the issues of deriving skylines in a dynamic and incomplete database, thus we compare our proposed solution to other existing works that are the closest to this research, namely: *ISkyline* [23], *SIDS* [27], and *IncOSkyline* [2]. These algorithms are mainly designed to deal with the issue of incompleteness of data in a database. [23] has clearly stated in his work that to derive a set of skylines when new data items are inserted into a database, would require the *ISkyline* algorithm to be run over the new state of the database, i.e. the entire database needs to be analysed.

In conducting the experiment, we first run each of the algorithms, namely: *ISkyline*, *SIDS*, and *IncOSkyline* over the initial incomplete database and derive a set of skylines,  $S_p$ . When the database changes its states, we run again each of the above algorithms over the new state of the database and derive a new set of skylines,  $S'_p$ . While for our proposed solution, we first run the *Phase I* of *DyIn-Skyline* solution over the initial incomplete database and derive a set of skylines,  $S$ . When the database changes its states, depending on the type of modification made, we run the *Phase II* to derive a set of skylines,  $S'$ . We compare the set of skylines,  $S$  ( $S'$ ), produced by our proposed solution against the set of skylines produced by the previous algorithms,  $S_p$  ( $S'_p$ , respectively), to validate the correctness of our proposed solution. Intuitively,  $S$  and  $S_p$  as well as  $S'$  and  $S'_p$  should produce the same set of skylines.

Two types of data sets are used in the experiments, namely: synthetic and real data sets. Table 4 presents the parameter settings for the synthetic and real data sets. The real data sets, namely: *NBA*, *MovieLens*, and *stock market*, are the data sets mostly used by the previous works [2], [5], [20], [22], [23], [27], [28]. Each experiment is run 10 times and we report the average value of these runs. In deriving the set of skylines, we assume that greater values are preferable compared to lesser ones. The performance measurements used in our experiments are number of pairwise comparisons and processing time as they are the most commonly used measurements in evaluating the performance of skyline algorithms [2], [23], [27]. These measurements are evaluated on different parameter settings that are data set size, number of dimensions, number of dimensions with missing values, and changing rate. The initial sizes of the *NBA*, *stock market*, and *MovieLens* data sets are 120K, 500K, and 1200K, respectively. As shown in Table 4, the size of these data sets including the synthetic data set is varied with 40K as the minimum size (*NBA*) and 1200K as the maximum size (*MovieLens*). The initial numbers of dimensions for *NBA*, *stock market*, and *MovieLens* data sets are 13, 16, and 4, respectively. In some experiments, this parameter is varied as shown in Table 4. We have also varied the number of dimensions with missing values, with minimum 1 dimension (*MovieLens*) and maximum 6 dimensions (*synthetic*, *NBA*, *stock market*). The incompleteness rate is fixed to 20% but the

**TABLE 4.** The parameter settings of the synthetic and real data sets.

Parameter Settings	Data Sets			
	<i>Synthetic</i>	<i>NBA</i>	<i>stock market</i>	<i>MovieLens</i>
Data Set Size ( <i>K</i> )	50, <b>100</b> , 150, 200, 250, 300	40, 60, 80, 100, <b>120</b>	200, 300, <b>500</b>	300, 500, 700, 900, <b>1200</b>
Number of Dimensions	3, 5, 7, 9, 11, 13, <b>15</b>	5, 7, 9, 11, <b>13</b>	5, 8, 10, 14, <b>16</b>	2, 3, <b>4</b>
Number of Dimensions with Missing Values	3, 4, 5, 6			1, 2
Incompleteness Rate (%)	<b>20</b>			
Changing Rate (%)	5, 10, <b>20</b> , 30			

size of the data with incomplete values varied, for example if the size of the data set is 40K, thus 20% from 20K = 4K is incomplete while 20% from 1200K = 240K; which gives a different size of incompleteness of data. Changing rate reflects how much of changes is made towards a data set. For instance, 20% changing rate indicates the amount of changes made based on the size of the data set. The changes to be made towards a data set are randomly selected among the insert, delete, and update operations. The bold values in Table 4 are the default values.

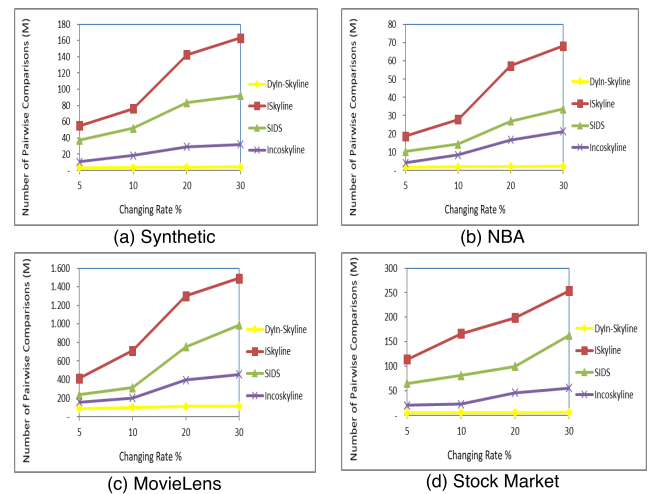
## B. THE EXPERIMENTAL RESULTS

This section presents the experimental results of the *DyIn-Skyline* solution in processing skyline queries over a dynamic and incomplete database, in which the changing state of the database is due to a data manipulation operation(s) (insert, delete or update a data item(s)). The *Phase I* of *DyIn-Skyline* is run to prepare the initial set of skylines as well as other necessary information that are captured by the *Bucket Dominating (BDG)*, *Bucket Dominated (BDD)*, and *Dominance History (DH)* that keep track of the dominating data items, dominated data items, and dominance relationships, respectively. Depending on the type of data manipulation, the appropriate component of the *Phase II* is invoked, namely: *Skyline-Insert Identifier (S-II)*, *Skyline-Delete Identifier (S-DI)*, and *Skyline-Update Identifier (S-UI)*. The number of pairwise comparisons and processing time are measured with different parameter settings as discussed in Part A and presented in Table 4. These results are compared to the results of *ISkyline* [23], *SIDS* [27], and *Incoskyline* [2], based on the synthetic and real data sets.

**Effect of Changing Rate**— One of the factors that has significant effect on the performance of skyline algorithms in processing skyline queries over a dynamic database is the changing rate of the database. In this section, we illustrate the experimental results of our proposed solution and the previous algorithms for both the synthetic and real data sets with respect to the number of pairwise comparisons and processing time, by varying the changing rate from 5% – 30% as applied in the previous studies [26], [35] with 20% incompleteness rate. The number of dimensions is fixed to 15, 13, 4,

and 16 for the *synthetic*, *NBA*, *MovieLens*, and *stock market* data set, respectively.

Figs 27(a) – (d) present the number of pairwise comparisons achieved by the *DyIn-Skyline*, *ISkyline* [23], *SIDS* [27], and *Incoskyline* [2], based on the *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets, respectively. From these figures, *DyIn-skyline* shows a steady performance with changing rate from 5% to 30%. Also, *DyIn-skyline* achieved better performance as compared to *ISkyline*, *SIDS*, and *Incoskyline*. Unlike *ISkyline*, *SIDS*, and *Incoskyline* which derive skylines over the entire data set after changes are made towards the data set, i.e. the new state of the data set, *DyIn-skyline* avoids unnecessary skyline computations. It relies on the information captured in the *Phase I*. Specifically, the *BDG*, *BDD*, and *DH* that keep track of the dominating data items, dominated data items, and dominance relationships, respectively; are utilised. *ISkyline* shows the worst performance even when the changing rate is only 5%. The nearest performance to *DyIn-Skyline* is *Incoskyline*, however *DyIn-Skyline* gained on average 24% improvement compared to *Incoskyline*.

**FIGURE 27.** The results of number of pairwise comparisons with varying changing rate.

Figs 28(a) – (d) present the processing time achieved by the *DyIn-Skyline*, *ISkyline* [23], *SIDS* [27], and *Incoskyline* [2], based on the *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets, respectively. From these figures, *DyIn-skyline* shows a steady performance with various changing rate, i.e. 5% – 30%. Also, *DyIn-skyline* attained less processing time as compared to *ISkyline*, *SIDS*, and *Incoskyline*. Similar trends as presented in Figs 27(a) – (d) can be seen in Figs 28(a) – (d). This is due to the fact that reducing the number of pairwise comparisons would reduce the processing time. The number of pairwise comparisons is reduced as the comparisons are performed between the data items retrieved from the *DH*, *BDG*, and *BDD*, as well as the data items inserted/deleted/updated into/from a data set, which is less than performing comparisons between the data items of the entire data set.

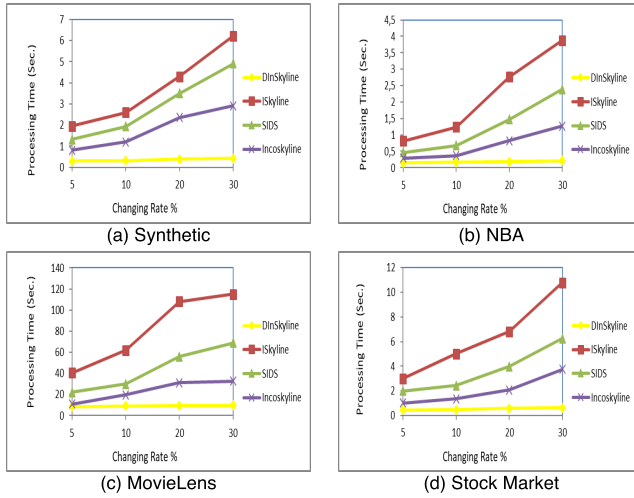


FIGURE 28. The results of processing time with varying changing rate.

**Effect of Data Set Size** – In this study, we have investigated the effect of data set size on the performance of *DyIn-Skyline*, which is one of the important factors that has high impact on the process of deriving skylines. Since the data sets have different number of dimensions and initial size, thus we use the following parameter settings: the numbers of dimensions for *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets are fixed to 15, 13, 4, and 16, respectively; while the incompleteness rate and changing rate are fixed to 20%. The data set sizes are varied as follows: 50K – 300K for the *synthetic* data set, 40K – 120K for the *NBA* data set, 300K – 1200K for the *MovieLens* data set, and 200K – 500K for the *stock market* data set.

Figs 29(a) – (d) present the number of pairwise comparisons achieved by the *DyIn-Skyline*, *ISkyline* [23], *SIDS* [27], and *Incoskyline* [2], based on the *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets, respectively. It is apparent that increasing the size of a data set is achieved by increasing

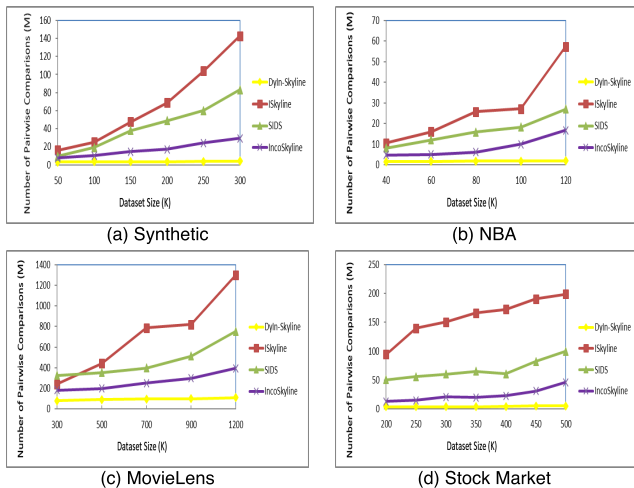


FIGURE 29. The results of number of pairwise comparisons with varying data set size.

the number of data items in the data set, thus the number of pairwise comparisons needed to be performed also increases. From the figures, it is obvious that varying the size of a data set does not have a significant impact on the performance of *DyIn-Skyline* while its performance outperforms the *ISkyline*, *SIDS*, and *Incoskyline* algorithms. This is because, *ISkyline*, *SIDS*, and *Incoskyline* algorithms perform pairwise comparisons between the data items of the entire data set, even though if the data items are unaffected by the changes. While, *DyIn-skyline* avoids unnecessary skyline computations by utilising the *BDG*, *BDD*, and *DH* that keep track of the dominating data items, dominated data items, and dominance relationships, respectively. *DyIn-skyline* focuses on the data items that are affected by the changes made towards the data set and make pairwise comparisons between them. For instance, if the data manipulation operation is an insert operation, then pairwise comparisons are performed between the inserted data items of the data set, which are then compared between the data items saved in the *DH*, *BDG*, and *BDD*. While, if the data manipulation is a delete operation, then each data item deleted from the data set is analysed to identify its role, either it is a dominating data item or a dominated data item to decide on the appropriate pairwise comparison to be performed. By doing this, *DyIn-skyline* managed to reduce the number of pairwise comparisons needed significantly.

Figs 30(a) – (d) present the processing time achieved by the *DyIn-Skyline*, *ISkyline* [23], *SIDS* [27], and *Incoskyline* [2], based on the *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets, respectively. From these figures, *DyIn-skyline* shows a steady performance with varying sizes of data sets. Also, *DyIn-skyline* achieved less processing time as compared to *ISkyline*, *SIDS*, and *Incoskyline*. Similar trends as presented in Figs 29(a) – (d) can be seen in Figs 30(a) – (d). This is due to the fact that reducing the number of pairwise comparisons would reduce the processing time. The number of pairwise comparisons is reduced as the comparisons are performed between the data items retrieved

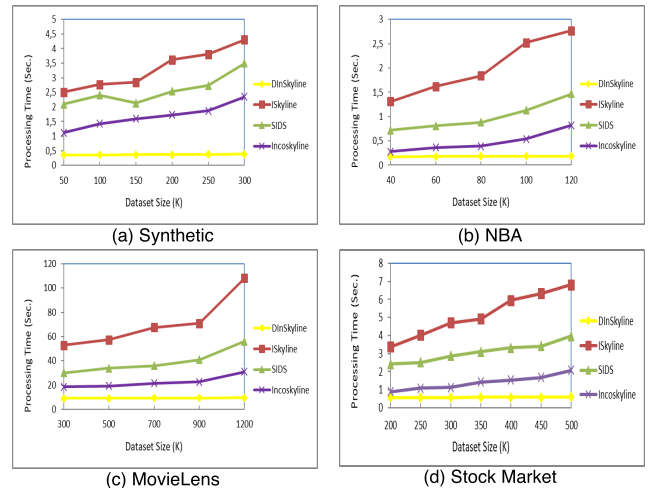


FIGURE 30. The results of processing time with varying data set size.

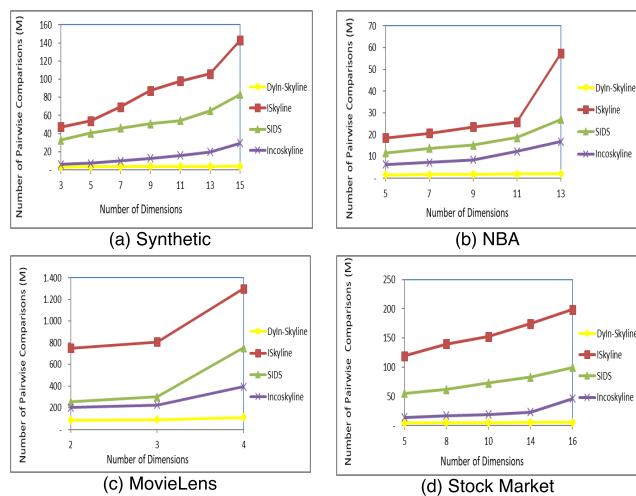
from the *DH*, *BDG*, and *BDD*, as well as the data items inserted/deleted/updated into/from a data set, which is less than performing comparisons between the data items of the entire data set. For both performance measurements, i.e. number of pairwise comparisons and processing time, *ISkyline* shows the worst performance. The nearest performance to *DyIn-Skyline* is *Incoskyline*.

**Effect of Data Dimensionality** – Besides changing rate and data set size, data dimensionality is also one of the factors that has significant effect on the performance of skyline algorithms in processing skyline queries. In this section, we illustrate the experimental results of our proposed solution and the previous algorithms, for both the synthetic and real data sets with respect to the number of pairwise comparisons and processing time. Since the data sets have different number of dimensions and initial size, thus we use the following parameter settings: the data set size for *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets is fixed to 300K, 120K, 1200K, and, 500K, respectively; while the incompleteness rate and changing rate are fixed to 20%. The numbers of dimensions are varied as follows: 3 – 15 dimensions for the *synthetic* data set, 5 – 13 dimensions for the *NBA* data set, 2 – 4 dimensions for the *MovieLens* data set, and 5 – 16 dimensions for the *stock market* data set.

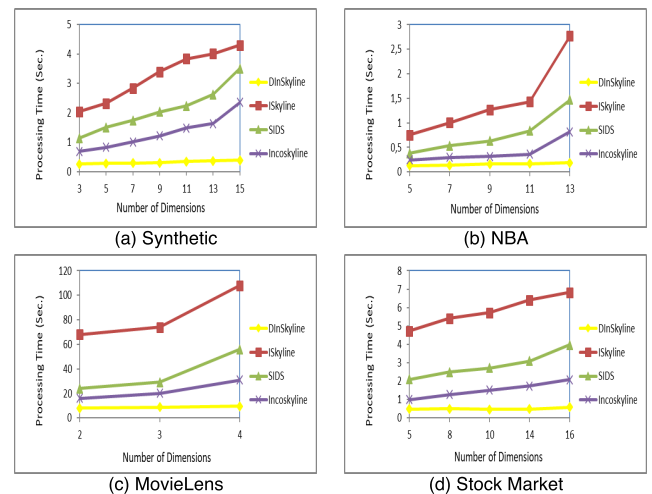
Comparing the results of number of pairwise comparisons presented in Figs 31(a) – (d), reveals that the performance of our proposed approach, *DyIn-skyline*, is better than the performance of *ISkyline*, *SIDS*, and *Incoskyline* algorithms for both the synthetic and real data sets. In fact, *DyIn-skyline* shows a steady performance even when the number of dimensions is increased, while a slight increment in number of pairwise comparisons can be seen in the other algorithms. Similar explanation given in the previous sections, applies here. Intuitively, *ISkyline*, *SIDS*, and *Incoskyline* algorithms perform pairwise comparisons between the data items of the entire data set, even though the data items are not affected by

the changes. While, *DyIn-skyline* avoids unnecessary skyline computations by utilising the *BDG*, *BDD*, and *DH*, that keep track of the dominating data items, dominated data items, and dominance relationships, respectively. *DyIn-skyline* focuses on the data items that are affected by the changes made towards the data set and make pairwise comparisons between them. By doing this, *DyIn-skyline* managed to reduce the number of pairwise comparisons needed significantly.

Figs 32(a) – (d) present the processing time achieved by the *DyIn-Skyline*, *ISkyline* [23], *SIDS* [27], and *Incoskyline* [2], based on the *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets, respectively. From these figures, *DyIn-skyline* shows a steady performance with varying number of dimensions. Also, *DyIn-skyline* achieved less processing time as compared to *ISkyline*, *SIDS*, and *Incoskyline*. Similar trends as presented in Figs 31(a) – (d) can be seen in Figs 32(a) – (d). This is due to the fact that reducing the number of pairwise comparisons would reduce the processing time. The number of pairwise comparisons is reduced as the comparisons are performed between the data items retrieved from the *DH*, *BDG*, and *BDD*, as well as the data items inserted/deleted/updated into/from a data set, which is less than performing comparisons between the data items of the entire data set. Similar to the other factors, *ISkyline* shows the worst performance with regard to both the number of pairwise comparisons and processing time, while the nearest performance to *DyIn-Skyline* is *Incoskyline*.



**FIGURE 31.** The results of number of pairwise comparisons with varying number of dimensions.



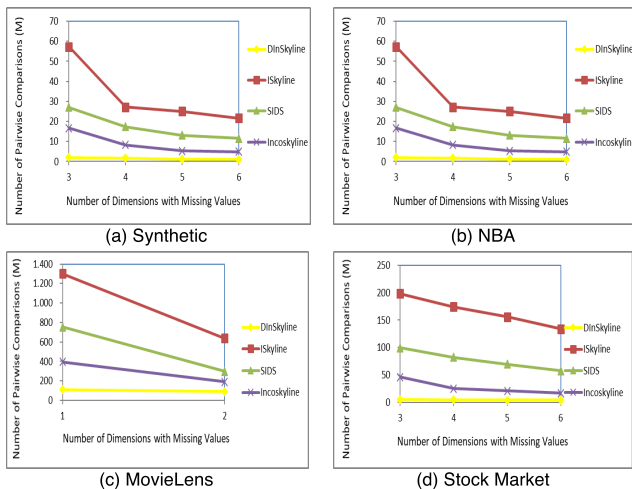
**FIGURE 32.** The results of processing time with varying number of dimensions.

**Effect of Number of Dimensions with Missing Values** – Since, incompleteness of data is one of the issues tackle by this study, thus investigating the performance of *DyIn-Skyline* in processing skyline queries with various number of dimensions with missing values is inevitable. In this section, we illustrate the experimental results of our proposed solution and the previous algorithms, for both the synthetic and real data sets with respect to the number of pairwise comparisons and processing time, by varying the number of dimensions



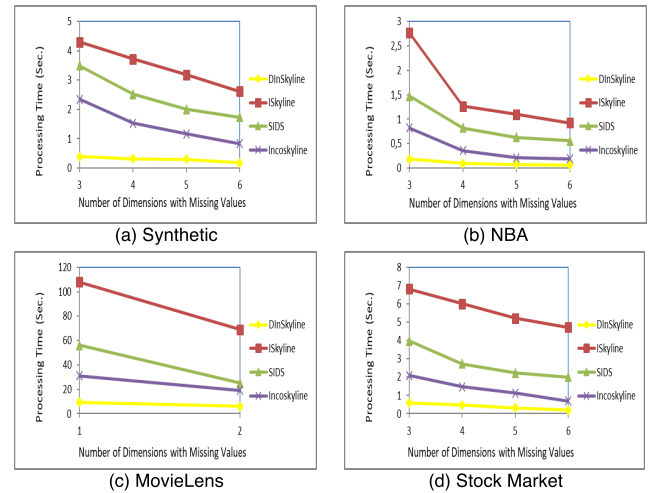
with missing values as follows: 3 – 6 dimensions for the *synthetic*, *NBA*, and *stock market* data sets, while 1 – 2 dimensions for the *MovieLens* data set, with changing rate and incompleteness rate fixed to 20%. The numbers of dimensions are fixed to 15, 13, 4, and 16 for the *synthetic*, *NBA*, *MovieLens*, and *stock market* data set, respectively.

Figs 33(a) – (d) present the number of pairwise comparisons achieved by the *DyIn-Skyline*, *ISkyline* [23], *SIDS* [27], and *Incoskyline* [2], based on the *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets, respectively. From these figures, it is obvious that *DyIn-skyline* shows a steady performance which reflects that the number of dimensions with missing values has no significant impact on the performance of *DyIn-skyline*. When the number of dimensions with missing values increases, the number of pairwise comparisons performed decreases, as reflected in the results of all the algorithms. Although the reduction shown by *DyIn-skyline* is small but *DyIn-skyline* achieved better performance as compared to *ISkyline*, *SIDS*, and *Incoskyline*, since it avoids unnecessary skyline computations by utilising the information captured in the *BDG*, *BDD*, and *DH* that keep track of the dominating data items, dominated data items, and dominance relationships, respectively.



**FIGURE 33. The results of number of pairwise comparisons with varying number of dimensions with missing values.**

Figs 34(a) – (d) present the processing time achieved by the *DyIn-Skyline*, *ISkyline* [23], *SIDS* [27], and *Incoskyline* [2], based on the *synthetic*, *NBA*, *MovieLens*, and *stock market* data sets, respectively. From these figures, *DyIn-skyline* shows a steady performance with varying number of dimensions with missing values. Also, *DyIn-skyline* gained less processing time as compared to *ISkyline*, *SIDS*, and *Incoskyline*. Similar trends as presented in Figs 33(a) – (d) can be seen in Figs 34(a) – (d). This is due to the fact that reducing the number of pairwise comparisons would reduce the processing time. When the number of dimensions with missing values increases, the number of pairwise comparisons performed decreases and consequently the processing time also

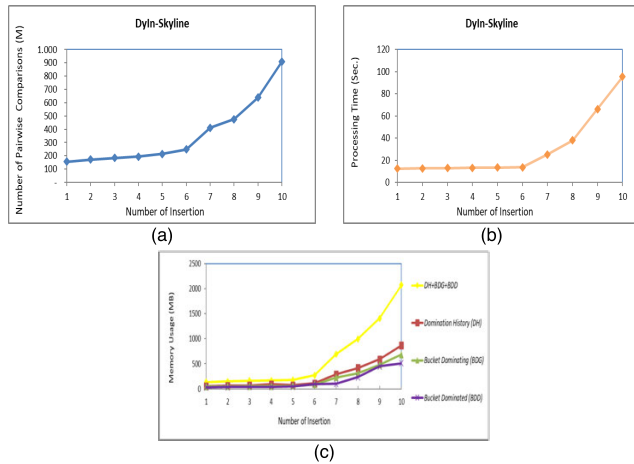


**FIGURE 34. The results of processing time with varying number of dimensions with missing values.**

decreases, as reflected in the results of all the algorithms. However, the processing time achieved by *DyIn-skyline* is always lesser than the other algorithms since comparisons in *DyIn-skyline* are performed between the data items retrieved from the *DH*, *BDG*, and *BDD*, as well as the data items inserted/deleted/updated into/from a data set, which is less than performing comparisons between the data items of the entire data set. Both sets of Figs 33 and 34 clearly depict that the worst performance with regard to the number of pairwise comparisons and processing time is shown by *ISkyline*, while the nearest performance to *DyIn-Skyline* is *Incoskyline*.

**Effect of Continuous Insertions** – In this study, we have investigated the effect of continuous insertions on the performance of *DyIn-Skyline* and the capability of *DyIn-Skyline* in handling huge data set. In this section, we illustrate the experimental results of our proposed solution, *DyIn-Skyline*, on the *synthetic* data set with respect to the number of pairwise comparisons, processing time, and memory usage. Here, the incompleteness rate is fixed to 20% while the number of dimensions is fixed to 10. The initial size of the data set is 1000K, and we performed 10 sets of insertion operations, in which each set of insertion operations involves an increment of 30% of changing rate.

Fig. 35(a) shows the performance of *DyIn-Skyline* with regard to the number of pairwise comparisons. It shows a steady performance with a slight increment between the iterations of the sets of insertion operations and starts to increase drastically at the 7th. iteration which reflects a double increment compared to the 6th. iteration. At this stage, the size of the *synthetic* data set has reached to around 5000K. The effect of continuous insertion on the processing time of *DyIn-Skyline* is shown in Fig. 35(b). Similar trends as shown in Fig. 35(a) can be seen in Fig. 35(b), in which starting at the 7th. iteration the processing time starts to show a drastic increment. Similarly, Fig. 35(c) shows a drastic increment of memory usage when the size of data set increases due to

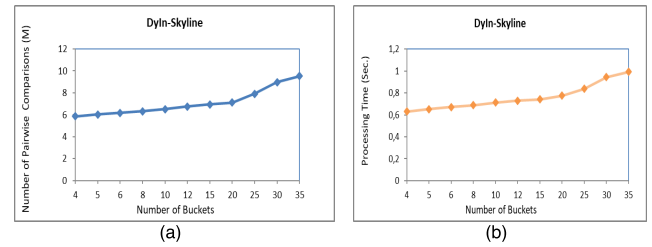


**FIGURE 35.** The results of (a) number of pairwise comparisons, (b) processing time, and (c) memory usage; with continuous insertions.

the continuous insertion of data items, particularly after the 7th. iteration. This is presented by the line  $DH+BDG+BDD$  which denotes the total memory usage of *DyIn-Skyline*. This is due to the introduction of three main lists, namely: *Domination History (DH)*, *Bucket Dominating (BDG)*, and *Bucket Dominated (BDD)*, that keep track of the domination relationships, dominating data items, and dominated data items, respectively. Comparing between these lists, *DH* consumes more memory than *BDG* and *BDD* as it stores all the domination relationships between data items in the data set. While *BDG* is a list of data items that dominates other data items regardless if they are being dominated by others consumes a slightly more memory compared to *BDD* that only stores data items that are being dominated. Based on this analysis, the *DyIn-Skyline* shows a good performance for data sets with the size of less than 5000K.

**Effect of Number of Buckets** – In this study, we have investigated the effect of number of buckets on the performance of *DyIn-Skyline*. Intuitively, the higher the number of buckets created implies the higher the number of inter-bucket comparisons that need to be performed. In this section, we illustrate the experimental results of our proposed solution, *DyIn-Skyline* on the *synthetic* data set with respect to the number of pairwise comparisons and processing time. Here, the incompleteness rate and the changing rate are fixed to 20% while the number of dimensions is fixed to 20, to ensure that a high number of distinct buckets can be created with a well-balanced number of data items in each bucket. The number of buckets is varied from 4 – 35. The initial size of the data set is 500K.

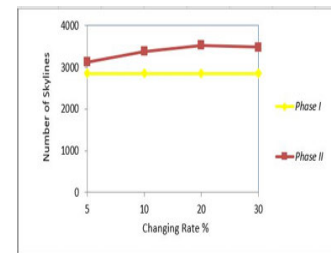
Fig. 36(a) shows the performance of *DyIn-Skyline* with regard to the number of pairwise comparisons. It shows a steady performance with a slight increment when the number of buckets is increased. However, the performance of *DyIn-Skyline* starts to show a drastic increment when the number of buckets is 20 until it reaches to 35. Similar trends as shown in Fig. 36(a) can be seen in Fig. 36(b), in which starting at 20 number of buckets, the processing time starts to show a



**FIGURE 36.** The results of (a) number of pairwise comparisons and (b) processing time with varying number of buckets.

drastic increment. This is due to the fact that with the same amount of data items and various bitmap representations, more buckets are created with each bucket having small number of data items. Filtering at the intra-bucket comparisons is less while more inter-bucket comparisons are performed.

**Effect of Changing Rate on the Number of Skylines of Phase I and Phase II** – In this section, we illustrate the experimental results of our proposed solution, *DyIn-Skyline*, on the *synthetic* data set with respect to the number of skylines produced by *Phase I* denoted as  $S$  and *Phase II* denoted as  $S'$ , by varying the changing rate from 5% – 30% with 20% incompleteness rate. The initial size of the *synthetic* dataset is 300K and the number of dimensions is fixed to 15. The experiment was run 10 times and we report the average value of these runs. Intuitively, there are two cases, namely: (i)  $S' = S$  – this indicates that the changes made towards the data set have no effect on the set of skylines produced in *Phase I* as all the skylines in  $S$  are also the skylines of  $S'$ ; and (ii)  $S' \neq S$  – this case is more realistic and it indicates that the changes made towards the data set effect on the set of skylines produced in *Phase I*. Note that  $S' \neq S$  can either be (i)  $S' \cap S = \emptyset$ , i.e. some of the skylines produced in *Phase I* are also the skylines of *Phase II* or (ii)  $S' \cap S = \emptyset$ , i.e. none of the skylines of  $S$  are in  $S'$ .



**FIGURE 37.** The results of number of skylines of Phase I and Phase II.

Fig. 37 shows the results of this experiment, for all of the runs the number of skylines produced in *Phase II* is slightly higher than those produced in *Phase I* with all runs showing  $S' \neq S$  and  $S' \cap S = \emptyset$ .

## VI. CONCLUSION

In this paper, we proposed a solution named *DyIn-Skyline* that is capable of deriving skylines over a dynamic and incomplete database. *DyIn-Skyline* which consists of two main phases

works by utilising the *DH*, *BDG*, and *BDD* that keeps track of the domination relationships, dominating, and dominated data items, respectively. A huge amount of pairwise comparisons can be avoided as clearly shown by the results of the experiments. Further enhancement to the proposed solution can be done by investigating the following area: (i) *Distributed Databases*— In our work, we assumed that the database is centralised and all data items for deriving the set of skylines are populated into a single table. However, it is interesting to investigate issues related to data items that are populated into several distinct databases which are scattered throughout the network. Apart from the issues as elaborated in this paper that relate to incompleteness and dynamism of data of a database, issue of reducing the amount of data transferred across the network in an attempt to derive a set of skylines of a distributed database, is worth to be investigated. (ii) *Uncertain Database* – another important and interesting area that can be explored is processing skyline queries in a dynamic and incomplete of an uncertain database. Skyline queries in an uncertain database are mainly probabilistic skylines as the exact values of the data items are not known during the process of deriving the set of skylines owing to either the values of the data items are missing or the values of the data items are in continuous forms (range of values). Thus, deriving an accurate set of skylines based on the uncertainty of data is challenging and worth to be investigated. (iii) *Cloud Environment*— In a cloud environment, data are massive and users are charged based on the amount of data transferred via the cloud environment. These massive data are dynamic and some of them are with missing information. Thus, handling massive incomplete and dynamic data over the cloud in deriving a set of skylines that will benefit the users is another issue that is worth investigating.

## ACKNOWLEDGMENT

All opinions, findings, conclusions and recommendations in this article are those of the authors and do not necessarily reflect the views of the funding agencies. The authors would like to thank the anonymous reviewers for their constructive comments.

## REFERENCES

- [1] 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.
- [2] 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.
- [3] W. T. Balke and U. Güntzer, "Multi-objective query processing for database systems," in *Proc. 13th Int. Conf. Very Large Data Bases*, 2004, pp. 936–947.
- [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.
- [5] 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.*, 2006, pp. 478–495.
- [6] X. W. Cui, L. G. Dong, H. Zou, and X. M. An, "Finding k-dominant skyline in dynamic dataset," in *Proc. 7th IEEE Int. Conf. Natural Comput.*, May 2011, pp. 1247–1250.
- [7] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, Mar. 2005.
- [8] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *Proc. 28th Int. Conf. Very Large Data Bases (VLDB)*, 2002, pp. 275–286.
- [9] G. B. Dehaki, H. Ibrahim, N. I. Udzir, F. Sidi, and A. A. Alwan, "Efficient skyline processing algorithm over dynamic and incomplete database," in *Proc. 20th Int. Conf. Integr. Web-Based Appl. Services (iiWAS2018)*, 2018, pp. 190–199.
- [10] B. Ilaria, P. Ciaccia, and M. Patella, "SaLSa: Computing the skyline without scanning the whole sky," in *Proc. 15th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, 2006, pp. 405–414.
- [11] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting: Theory and optimizations," *Intell. Inf. Syst. J.*, vol. 31, pp. 595–604, May 2005.
- [12] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. 19th Int. Conf. Data Eng. (ICDE)*, 2003, pp. 717–816.
- [13] J. Pei, W. Jin, M. Ester, and Y. Tao, "Catching the best views of skyline: A semantic approach based on decisive subspaces," in *Proc. 31st Int. Conf. Very Large Data Bases (VLDB)*, 2005, pp. 253–264.
- [14] J. Lee, H. Im, and G.-W. You, "Optimizing skyline queries over incomplete data," *Inf. Sci.*, vols. 361–362, pp. 14–28, Sep. 2016.
- [15] J. Lee, G. You, and S. Hwang, "Personalized top-k skyline queries in high-dimensional space," *Inf. Syst.*, vol. 34, no. 1, pp. 45–61, 2009.
- [16] K. Zhang, H. Gao, X. Han, Z. Cai, and J. Li, "Probabilistic skyline on incomplete data," in *Proc. ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 427–436.
- [17] 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.
- [18] K.-L. Tan, E. Pin-Kwang, and C. O. Beng, "Efficient progressive skyline computation," in *Proc. 27th Int. Conf. Very Large Data Bases (VLDB)*, 2001, pp. 301–310.
- [19] M. L. Yiu and N. Mamoulis, "Multi-dimensional top-k dominating queries," *VLDB J.*, vol. 18, no. 3, pp. 695–718, Jun. 2009.
- [20] M. L. Yiu and N. Mamoulis, "Efficient processing of top-k dominating queries on multi-dimensional data," in *Proc. 33rd Int. Conf. Very Large Data Bases (VLDB)*, 2007, pp. 483–494.
- [21] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous processing of preference queries in data streams," in *Proc. 36th Int. Conf. Current Trends Theory Pract. Comput. Sci.*, 2010, pp. 47–60.
- [22] M. Morse, J. M. Patel, and W. I. Grosky, "Efficient continuous skyline computation," *Inf. Sci.*, vol. 177, no. 17, pp. 3411–3437, Sep. 2007.
- [23] E. M. Khalefa, F. M. Mokbel, and J. J. Livandoski, "Skyline query processing for incomplete data," in *Proc. 24th Int. Conf. Data Eng. (ICDE)*, Apr. 2008, pp. 556–565.
- [24] M. Shamsul Arefin, "Skyline sets queries for incomplete data," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 5, pp. 67–80, Oct. 2012.
- [25] P. Godfrey, "Skyline cardinality for relational processing," *Found. Inf. Knowl. Syst.*, vol. 2942, pp. 78–97, 2005.
- [26] P. Wu, D. Agrawal, O. Egecioglu, and A. El Abbadi, "Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, Apr. 2007, pp. 486–495.
- [27] R. Bharuka and P. Sreenivasa KuMar, "Finding skylines for incomplete data," in *Proc. 24th Australas. Database Conf.*, vol. 137, 2013, pp. 109–117.
- [28] R. C.-W. Wong, A. W.-C. Fu, J. Pei, Y. S. Ho, T. Wong, and Y. Liu, "Efficient skyline querying with variable user preferences on nominal attributes," in *Proc. 34th Int. Conf. Very Large Data Bases (VLDB)*, 2008, pp. 1032–1043.
- [29] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. 17th Int. Conf. Data Eng. (ICDE)*, Apr. 2001, pp. 421–430.
- [30] S. Chaudhuri and L. Grava, "Evaluating top-k selection queries," in *Proc. 25th Int. Conf. Very Large Data Bases (VLDB)*, 1999, pp. 397–410.
- [31] X. Miao, Y. Gao, B. Zheng, G. Chen, and H. Cui, "Top-k dominating queries on incomplete data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 252–266, Jan. 2016.
- [32] X. Miao, Y. Gao, G. Chen, and T. Zhang, "k-dominant skyline queries on incomplete data," *Inf. Sci.*, vols. 367–368, pp. 990–1011, Nov. 2016.
- [33] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang, "Efficient computation of the skyline cube," in *Proc. 31st Int. Conf. Very Large Data Bases (VLDB)*, 2005, pp. 267–278.



- [34] Y. Gulzar, A. A. Alwan, N. Salleh, I. F. A. Shaikhli, and S. I. M. Alvi, "A framework for evaluating skyline queries over incomplete data," *Procedia Comput. Sci.*, vol. 94, pp. 191–198, 2016.
- [35] Y. Fang, and C.-Y. Chan, "Efficient skyline maintenance for streaming data with partially-ordered domains," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2010, pp. 322–336.



**GHAZALEH BABANEJAD DEHAKI** was born in Tehran, Iran, in March 1981. She received the bachelor's degree in the field of software engineering from the Faculty of Computer Science, Iran University of Science and Technology, in 2007, and the master's degree in knowledge management with multimedia from Multimedia University (MMU), Malaysia, specializing in semantic web ontology and recommender systems. She is currently pursuing the Ph.D. degree in the field of database systems with Universiti Putra Malaysia (UPM), specializing in preference query in dynamic and incomplete database systems.



**HAMIDAH IBRAHIM** received the Ph.D. degree in computer science from the University of Wales, Cardiff, U.K., in 1998. She is currently a Full Professor with the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM). Her current research interests include databases (distributed, parallel, mobile, biomedical, and XML) focusing on issues related to integrity maintenance/checking, ontology/schema/data integration, ontology/schema/data mapping, cache management, access control, data security, transaction processing, query optimization, query reformulation, preference evaluation–context-aware, information extraction, and concurrency control; and data management in mobile, grid, and cloud.



**FATIMAH SIDI** received the Ph.D. degree in management information system from Universiti Putra Malaysia (UPM), Malaysia, in 2008. She is currently working as an Associate Professor in the discipline of computer science with the Department of Computer Science, Faculty of Computer Science and Information Technology, UPM. Her current research interests include knowledge and information management systems, data and knowledge engineering, database, and data warehouse.



**NUR IZURA UDZIR** received the Bachelor of Computer Science and Master of Science degrees from Universiti Putra Malaysia (UPM), in 1996 and 1998, respectively, and the Ph.D. degree in computer science from the University of York, U.K., in 2006. She is currently an Associate Professor with the Faculty of Computer Science and Information Technology, UPM, since 1998. Her research interests include access control, secure operating systems, intrusion detection systems, coordination models and languages, and distributed systems. She is a member of the IEEE Computer Society.



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



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

...