A Rule-based Skyline Computation over a Dynamic Database

Ghazaleh Babanejad Dehaki, Hamidah Ibrahim†, Fatimah Sidi, Nur Izura Udzir
Department of Computer Science
Universiti Putra Malaysia
Serdang, Selangor, Malaysia
ghazaleh.babanejd@upm.student.edu.my,
{hamidah.ibrahim, izura, fatimah}@upm.edu.my

Ali A. Alwan
Department of Computer Science
International Islamic University Malaysia, Kuala Lumpur, Malaysia
aliamer@iium.edu.my

ABSTRACT

Skyline query which relies on the notion of Pareto dominance filters the data items from a database by ensuring only those data items that are not worse than any others are selected as skylines. However, the dynamic nature of databases in which their states and/or structures change throughout their lifetime to incorporate the current and latest information of database applications, requires a new set of skylines to be derived. Blindly computing skylines on the new state/structure of a database is inefficient, as not all the data items are affected by the changes. Hence, this paper proposes a rule-based approach in tackling the above issue with the main aim at avoiding unnecessary skyline computations. Based on the type of operation that changes the state/structure of a database, i.e. insert/delete/update a data item(s) or add/remove a dimension(s), a set of rules are defined. Besides, the prominent dominance relationships when pairwise comparisons are performed are retained; which are then utilised in the process of computing a new set of skylines. Several analyses have been conducted to evaluate the performance and prove the efficiency of our proposed solution.

CCS CONCEPTS

• Information systems • Database management system

KEYWORDS

Skyline queries, Incomplete database, Dynamic database, Pairwise comparisons.

1 Introduction

Skyline queries also known as Pareto optimum [26] work by finding the best, most preferred data items from a collection of data items in a database. The following example, i.e. the hotel booking scenario, is commonly used to explain the skyline query. Consider a user who wanted to find a hotel that is nearest to the city centre (minimum distance) and with the cheapest price (minimum price). Generally, hotels that are near to a city centre are more expensive than those which are far away from the city centre. Consequently, finding a hotel that meets both preferences (dimensions) returns an empty result. Skyline query which relies on the powerful skyline operator introduced by [26] attempts to filter the data items that are not dominated by any other data items in the database. Hence, based on the above example, the following filtered hotels are suggested to the user: (i) hotel that is nearest to the city centre (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 city centre, and (iii) hotel(s) with price cheaper than hotel (i) and distance nearer than (ii). As a result, the more dimensions that need to be considered, the more choices are given to the user.

Databases are dynamic in nature in which their states/structures change throughout their lifetime. These changes are necessary to reflect the changing requirements, new functionalities, compliance to new regulations, integration with other systems, and new security or privacy measures. The changes on the database could be achieved through a data manipulation operation(s) (insert, delete, update) or a data definition operation(s) (alter table, etc). The skylines derived before changes are made towards the database are no longer valid in the new state/structure of the database. However, computing the skylines over the entire database after changes are made is inefficient as not all the data items are affected by the changes.

This paper proposes a rule-based approach that attempts to avoid unnecessary computation of skylines, i.e. unnecessary pairwise comparisons between data items, when a new set of skylines needs to be identified due to changes made towards a database. To make our solution more useful, we consider both complete and incomplete database. Here, we dealt with two types of operations, namely: (i) operations that change the state of a database which include inserting a new data item(s) into the database or deleting/updating an existing data item(s) from the database and (ii) operations that do not only change the state but also the structure of a database which include adding a new dimension(s) to the database or removing an existing dimension(s) from the database. Besides, the domination relationships between data items that are identified when pairwise comparisons are performed are retained to be utilised during the process of identifying a new set of skylines. This paper enhances our previous work [7] with regard to the following: (i) based on the type of operation that changes the state and/or the structure of a database, it presents the theoretical foundation that clearly shows the derivation of the set of rules in identifying a new set of

iiWAS '20, November 30-December 2, 2020, Chiang Mai, Thailand

ACM ISBN 978-1-4503-8922-8/20/11$15.00
https://doi.org/10.1145/3428757.3429117

skylines, (ii) the rule-based approach can be applied to any level of domination relationships, i.e. skyline, local skyline, bucket skyline, etc., as commonly derived by most skyline algorithms especially those that group/cluster/categorise the data items based on certain criteria like the bitmap representations of the data items [1], [16], [21], [24], [26]; in other words the rules can be easily applied to these skyline algorithms which enhance their capability to handling dynamic databases, and (iii) the rule-based solution is applicable for operations that change the state as well as the structure of a database, while [7] is limited to the operations that change the state of a database.

The remaining parts of the paper are structured as follows. In Section 2, the previous works related to the area of processing skyline queries are presented. The necessary definitions and notations are described in Section 3. Section 4 elaborates our proposed rule-based approach in processing skyline queries over a dynamic database. A running database example is also given to clarify the proposed approach. The experimental result is demonstrated in Section 5. Conclusion is presented in the final section, Section 6.

2 Related Work

Since the introduction of skyline operator by [26], many variants of skyline algorithms have evolved; among the notable algorithms include *Divide-and-Conquer* (*D&C*), *Block Nested Loop* (*BNL*) [26], *Bitmap and Index* [16], *Sort Filter Skyline* (*SFS*) [11], [12], *Nearest Neighbor* (*NN*) [5], *Linear Elimination Sort Skyline* (*LESS*) [23], *Branch and Bound Skyline* (*BBS*) [4], *Lattice Skyline* (*LS*) [19], *Implicit Preference Order tree* [25], and *Sort and Limit Skyline algorithm* (*SaLSa*) [10]. These algorithms are designed mainly to resolve the optimisation problem that relates to the reduction of the number of pairwise comparisons that needs to be performed during the skyline computations. However, these algorithms are based on a rigid assumption that the database is complete, in which all values of the dimensions are present.

Recently, several approaches have been established in resolving issues related to the incompleteness of data in a database [1], [2], [3], [6], [13], [14], [15], [18], [20], [21], [22], [24], [27], [28], [29], [30], [31]. However, only the works by [1], [21], [24] emphasise on skyline queries, with similar aim that is to resolve the optimisation problem while the databases are with missing values. These works are further elaborated below.

In [21] two skyline algorithms are proposed, namely: *Bucket* and *Iskyline*. The *Bucket* algorithm relies on the bitmap representation to group the data items into buckets. Local skylines are then identified by utilising the conventional skyline algorithm. Finally, the local skylines of each bucket are compared to each other to derive the final skylines. Meanwhile, the *Iskyline* algorithm works in a similar fashion as *Bucket* algorithm, with additional of two optimisation techniques to reduce the number of local skylines. Later, an approach is proposed named *SIDS* by [24], which works based on the following rule: if a point has not been pruned 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. Similar to *Iskyline* algorithm, *Incoskyline* [1] works by clustering the data items into related buckets, in which local skylines are then identified. *Incoskyline* which is based on a heuristic approach, derives a set of virtual skylines named k-dom from the local skylines to eliminate the local skylines of a cluster.

Although there are works that emphasise on dynamic database like [17] but these works support solution for top-k and top-k dominating. To the best of our knowledge, there is no work that has been done in computing skylines over a dynamic database in which the changing state/structure of the database is due to inserting a new data item(s) into the database, deleting or removing an existing data item(s) from the database, adding a new dimension(s) to the database, and removing an existing dimension(s) from the database.

3 Preliminaries

In this section, we first present the necessary definitions and introduce the notations that are used throughout this paper. Our explanation is based on the following: Assume an initial database, D, with m dimensions, $d = \{d_1, d_2, ..., d_m\}$, also denoted as D^m, with h data items $D = \{p_1, p_2, \ldots, p_h\}$. Note that the m dimensions are the dimensions considered during skyline computations.

Definition 1 *Dominance Relationship*: A data item $p_i \in D$ is said to dominate another data item $p_j \in D$ where $i \neq j$ 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.

Definition 2 *Skylines*: $p_i \in D$ is a skyline of D if there are no other data items $p_j \in D$ where $i \neq j$ that dominates p_i. We use the symbol S to denote the set of skylines of a database D.

Definition 3 *Changing Database State* [7], [8], [9]: Given a database, D, its state is changed to a new state, D_{new}, due to the following operations:

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

Definition 4 *Changing Database Structure*: The database, D, with m dimensions denoted as D^m, is said to be in a new structure denoted as D^n with n dimensions, where $n > m$ or $n < m$ due to either (i) a set of new dimensions, $d_{<add>} = \{d_a, d_{a+1}, \ldots, d_{a+b}\}$, is added to D^m or (ii) a set of existing dimensions, $d_{<remove>} = \{d_r, d_{r+1}, \ldots, d_{r+s}\}$, is removed from D^m.

With regard to *Definition* 3 and *Definition* 4, the database D[1] is said to be *dynamic* as its state/structure changes due to the operations stated in the definitions.

[1] Without loss of generality, the term database covers both complete and incomplete database.

Figure 1(a) shows a toy example, which is used throughout this paper to clarify our proposed solution. There are ten data items with each data item having three dimensions labelled as d_1, d_2, and d_3 that are used in the computation of the skylines. Nevertheless, the proposed solution can handle missing values (incomplete database) as clearly shown by Figure 1(b) and the analyses that we have conducted. Figure 1(b) is an example of data items that are grouped based on their missing values. The symbol '–' is used to denote missing value.

Data Item	d_1	d_2	d_3
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

(a)

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	–

(b)

Figure 1: An example of (a) complete database and (b) incomplete database

4 The Proposed Approach

In this section, we elaborate on our proposed rule-based approach which aims at avoiding unnecessary computation of skylines, i.e. unnecessary pairwise comparisons between data items, when a new set of skylines needs to be identified due to changes made towards a database. Section 4.1 presents the lists that we have designed in order to identify and retain the prominent dominance relationships while Section 4.2 and Section 4.3 explain the rules that we have derived based on the type of operations considered in this paper.

4.1 Dominance Relationships

In our work, the domination relationships between data items that are identified when pairwise comparisons are performed are retained in a list called the *Domination History* (*DH*). The *DH* has the following structure: *DH* = <*Dominating*, *Dominated*> where *Dominating* is the data item that dominates other data items while *Dominated* is the list of data items that is dominated by the data item. For instance, if $p_i \succ p_j$, then p_i is the *dominating* data item while p_j is the *dominated* data item of p_i while if $p_r \succ p_t$, $p_r \succ p_u$, and $p_r \succ p_v$ then p_r is the *dominating* data item while $\{p_t, p_u, p_v\}$ is the list of *dominated* data items of p_r. Table 1 presents the pairwise comparisons that are performed based on the data items given in Figure 1(a) while the shaded rows represent the final entries of *DH*. The *DH* list is updated whenever a new domination relationship is identified. Note that the same results are obtained for Figure 1(b).

Table 1: Sample of entries in the *DH*

Data Items		Pairwise Comparison	*DH*	
			Dominating	*Dominated*
w_1	w_2	$w_1 \succ w_2$	w_1	$\{w_2\}$
w_1	w_3	$w_1 \succ w_3$	w_1	$\{w_2, w_3\}$
w_1	w_4	$w_4 \succ w_1$	w_1	$\{w_2, w_3\}$
			w_4	$\{w_1\}$
w_4	w_5	$w_4 \succ w_5$	w_1	$\{w_2, w_3\}$
			w_4	$\{w_1, w_5\}$
w_4	w_6	$w_4 \succ w_6$	w_1	$\{w_2, w_3\}$
			w_4	$\{w_1, w_5, w_6\}$
w_4	w_7	$w_4 \succ w_7$	w_1	$\{w_2, w_3\}$
			w_4	$\{w_1, w_5, w_6, w_7\}$
w_4	w_8	$w_4 \succ w_8$	w_1	$\{w_2, w_3\}$
			w_4	$\{w_1, w_5, w_6, w_7, w_8\}$
w_4	w_9	$w_4 \succ w_9$	w_1	$\{w_2, w_3\}$
			w_4	$\{w_1, w_5, w_6, w_7, w_8, w_9\}$
w_4	w_{10}	$w_4 \succ w_{10}$	w_1	$\{w_2, w_3\}$
			w_4	$\{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}\}$

We have designed another two lists, namely: (i) *Dominating* (*DG*) that keep tracks of the data items that dominate other data items as well as those data items that are not dominated by other data items and (ii) *Dominated* (*DD*) that keep tracks of the data items that are dominated by other data items. By having the *DG* and *DD* lists, the skylines/local skylines/bucket skylines can be easily identified. The terms local skyline and bucket skyline are used to represent the data item that is not dominated by other data items within a group/bucket while the term skyline is used to represent the data item that is not dominated by other data items in the whole database. In this section, we will use the term candidate skyline/potential skyline to refer to both skyline and local/bucket skyline as not to limit our solution to a certain level of domination relationships. Thus, the candidate skylines are the data items that appeared in the *DG* but not in the *DD*, i.e. $\{S \mid S \in DG \wedge S \notin DD\}$. This is because the data item that appears in both lists, i.e. *DG* and *DD*, 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 other data items and thus should not be considered as one of the candidate skylines. Hence, a data item $p_i \in D$ can be one of the following: (i) $p_i \in DG$ and $p_i \notin DD$, (ii) $p_i \in DG$ and $p_i \in DD$, and (iii) $p_i \notin DG$ and $p_i \in DD$. Figure 2 presents the *DG* and *DD* based on the data items given in Figure 1. Based on this example, w_4 is a candidate skyline, i.e. it is a final skyline if the data items presented in Figure 1 is the whole database or local skyline otherwise.

DG
w_1
w_4

DD
w_1
w_2
w_3
w_5
w_6
w_7
w_8

Figure 2: Example of *DG* and *DD* lists

4.2 Insert/Delete/Update a Data Item(s)

In this section, we present our proposed rules in deciding whether an existing candidate skyline, $p_i \in D$, which is identified based on the condition $p_i \in DG \wedge p_i \notin DD$ still has a potential to be a candidate skyline due to operations that change the state of a database which include inserting a new data item(s) into the database or deleting/updating an existing data item(s) from the database. Here, we assume the following: an initial database, D with m dimensions $d = \{d_1, d_2, \ldots, d_m\}$ and a set of skylines, S, derived based on D.

4.2.1 Insert Operation. Given a set of data items to be inserted $D_{<insert>}$ into the D, the new state of the database, $D_{new} = D \cup D_{<insert>}$. 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, S', given the $D_{<insert>}$, pairwise comparisons are performed only between the data item $p_i \in DG$ and $p_i \notin DD$ and the data item $p_j \in D_{<insert>}$. This is because p_i is considered as the best, most preferred data items within its collection (database, group, cluster, bucket, etc.), hence if $p_i \succ p_j$ then p_i is still considered as the best, most preferred data items. Otherwise, if $p_j \succ p_i$, then p_j is better than p_i and better than other data items that are dominated by p_i. Here, the DG and DD are utilised. Based on the above, the following rules are defined:

Rule 1*i*: If $p_j \succ p_i$ where $p_i \in DG$, $p_i \notin DD$, and $p_j \in D_{<insert>}$, then p_j has a potential to be a candidate skyline while p_i is no longer a candidate skyline.

Rule 2*i*: If $p_i \succ p_j$ where $p_i \in DG$, $p_i \notin DD$, and $p_j \in D_{<insert>}$, then p_i is still a candidate skyline while p_j has no potential to be a candidate skyline.

Rule 3*i*: If p_i and p_j do not dominate each other where $p_i \in DG$, $p_i \notin DD$, and $p_j \in D_{<insert>}$, then both p_i and p_j have potentials to be candidate skylines.

Assuming the $D_{<insert>}$ given in Figure 3, $w_4 \in DG$, and $w_4 \notin DD$ as shown in Figure 2. Comparing w_4 with each data item of $D_{<insert>}$ will produce the following results:

(a) $w_4 \succ w_{11}$, where *Rule* 2*i* is applied. In this case, w_{11} is not a candidate skyline.
(b) $w_{12} \succ w_4$, in which w_{12} has a potential to be a candidate skyline as indicated by *Rule* 1*i*, while w_4 is no longer a candidate skyline.
(c) Both w_4 and w_{13} do not dominate each other and based on *Rule* 3*i* both have potentials to be candidate skylines.

Based on this example, w_4 is no longer a candidate skyline while w_{12} is a candidate skyline.

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

Figure 3: Example of $D_{<insert>}$

4.2.2 Delete Operation. Given a set of data items to be deleted $D_{<delete>}$ from the D, the new state of the database, $D_{new} = D - D_{<delete>}$. 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 $D_{<delete>}$, it is important to identify the data items that are dominated by $p_i \in D_{<delete>}$ or the data items that dominate the data item $p_i \in D_{<delete>}$, since these data items will have potentials to be candidate skylines. Here, the DH, DG, and DD are utilised. Based on the above, the following rules are defined:

Rule 1*d*: If $p_i \in DG$ and $p_i \notin DD$, i.e. p_i is a candidate skyline, and p_i dominates a data item $p_j \in D$ while $p_j \notin D_{<delete>}$, then p_j has a potential to be a candidate skyline.

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

Assuming the $D_{<delete>}$ given in Figure 4. Utilising the above rules will produce the following results:

(a) $w_4 \in DG$ and $w_4 \notin DD$ as shown in Figure 2, based on *Rule* 1*d*, $\{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}\} \notin D_{<delete>}$ have potentials to be candidate skylines. The list $\{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}\}$ is obtained from the DH. w_1, for instance is a candidate skyline once the data item w_4 is deleted from the database.
(b) $w_2 \notin DG$ and $w_2 \in DD$ as shown in Figure 2, based on *Rule* 2*d* and the DH, $w_1 \succ w_2$ while $w_1 \notin D_{<delete>}$ and thus it has a potential to be a candidate skyline.

Data Item	d_1	d_2	d_3
w_2	1	3	1
w_4	6	6	6

Figure 4: Example of $D_{<delete>}$

4.2.3 Update Operation. An update operation in our work is achieved by performing a delete operation followed by an insert operation. Thus, the above rules in subsections 4.2.1 and 4.2.2 are applied accordingly.

4.3 Add/Remove a Dimension(s)

In this section, we present our proposed rules in deciding whether a data item $p_i \in D^m$ which dominates another data item, say $p_j \in D^m$ where $i \neq j$, still maintain its domination power due to operations that change the state as well as the structure of a database which include adding a new dimension(s) to the database and removing an existing dimension(s) from the database. Here, we assume the following: an initial database, D^m with m dimensions $d = \{d_1, d_2, \ldots, d_m\}$ and a set of skylines, S^m, derived based on D^m.

4.3.1 Add a Dimension(s). Given a set of dimensions to be added $d_{<add>}$ to the D^m, the new structure of the database denoted as D^n contains $d \cup d_{<add>}$ dimensions with $n = |d| \cup |d_{<add>}|$. It is unwise to perform pairwise comparisons between all the data items of D^n since $D^m \subseteq D^n$. With this regard, only the D^{n-m} should be analysed. In our work, to derive a set of skylines, S^n, given the $d_{<add>}$, pairwise comparisons are performed only between the data item $p_i \in DG$ and the data item that it dominates $p_j \in D^m$. This is because p_i is considered better than p_j based on d, hence if $p_i \succ p_j$ based on $d_{<add>}$ then p_i is still considered better than p_j. Otherwise, if $p_j \succ p_i$ based on $d_{<add>}$, then p_j has a potential to be a candidate skyline. Here, the DH, DG, and DD are utilised. Based on the above, the following rules are defined:

Rule 1*a*: If $\forall d_k \in d_{<add>}$, $p_i.d_k \geq p_j.d_k$ where $p_i \in DG$, then p_i still has a potential to be a candidate skyline of D^n.

Rule 2*a*: If $\forall d_k \in d_{<add>}$, $p_j.d_k \geq p_i.d_k \wedge \exists d_l \in d_{<add>}$, $p_j.d_l > p_i.d_l$, where $p_i \in DG$ and $p_i \notin DD$, then p_j which was initially not a candidate skyline now has a potential to be a candidate skyline of

D^n. Note p_i is still a candidate skyline since it is still better than p_j based on d.

Rule 3*a*: If p_i and p_j do not dominate each other based on $d_{<add>}$, then both p_i and p_j have potentials to be candidate skylines of D^n.

Assuming the $d_{<add>}$ given in Figure 5, $w_1 \in DG$, and $w_4 \in DG$ as shown in Figure 2. Utilising the above rules will produce the following results:

(a) Based on *DH* given in Table 1, $w_1 \succ \{w_2, w_3\}$ which implies w_1 is better than both w_2 and w_3 with regard to d. However, having the two new dimensions, either w_1 still dominates w_2 and/or w_3 or they do not dominate each other which make them the candidate skylines. $w_1 \succ w_2$ based on d and $d_{<add>}$ in which we can conclude that w_1 has a potential to be a candidate skyline of D^n as defined by Rule 1*a*. Similar case can be seen between w_1 and w_3.

(b) Based on *DH* given in Table 1, $w_4 \succ \{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}\}$ which implies w_4 is better than the data items listed above with regard to d. However, comparing w_4 against w_1 based on d_4 and d_5 shows that they do not dominate each other and are considered as the candidate skylines as defined by Rule 3*a*. However, comparing w_4 and w_5 based on $d_{<add>}$, we noticed that $w_5 \succ w_4$, hence w_5 now has a potential to be a candidate skyline as defined by Rule 2*a* while w_4 is still a candidate skyline as it is better than w_5 with regard to d.

Data Item	d_4	d_5
w_1	6	4
w_2	4	3
w_3	4	2
w_4	3	7
w_5	4	8
w_6	2	1
w_7	6	5
w_8	5	3
w_9	3	2
w_{10}	1	1

Figure 5: Example of $d_{<add>}$

4.3.2 Remove a Dimension(s). Given a set of dimensions to be removed $d_{<remove>}$ from the D^m, the new structure of the database denoted as D^n contains $d - d_{<remove>}$ dimensions with $n = |d| - |d_{<remove>}|$. It is unwise to perform pairwise comparisons between all the data items of D^n since $D^n \subseteq D^m$. With this regard, only the D^{m-n} should be analysed. In our work, to derive a set of skylines, S^n, given the $d_{<remove>}$, pairwise comparisons are performed only between the data item $p_i \in DG$ and the data item that it dominates $p_j \in D^m$. This is because p_i is considered better than p_j based on d, hence if $p_i \succ p_j$ based on $d - d_{<remove>}$ then p_i is still considered better than p_j. Otherwise, if $p_j \succ p_i$ based on $d - d_{<remove>}$, then p_j has a potential to be a candidate skyline. Here, the *DH*, *DG*, and *DD* are utilised. Based on the above, the following rules are defined:

Rule 1*r*: If $\forall d_k \in d_{<remove>}$, $p_i.d_k = p_j.d_k$, then p_i is still a candidate skyline of D^n.

Rule 2*r*: If $\exists d_l \in d_{<remove>}$, $p_i.d_l > p_j.d_l$, there are two further possible cases, either (i) $\forall d_k \in d$, $p_i.d_k \geq p_j.d_k \wedge \exists d_l \in d$, $p_i.d_l > p_j.d_l$ or (ii) $\forall d_k \in d$, $p_i.d_k = p_j.d_k$ hold. For both cases, p_i is still a candidate skyline while for case (ii) both p_i and p_j have potentials to be candidate skylines of D^n. However, it is not easy to differentiate between cases (i) and (ii) without analysing the D^n.

Assuming the $d_{<remove>}$ given in Figure 6, $w_1 \in DG$, and $w_4 \in DG$ as shown in Figure 2. Utilising the above rules will produce the following results:

(a) Based on *DH* given in Table 1, $w_1 \succ \{w_2, w_3\}$ which implies w_1 is better than both w_2 and w_3 with regard to d. However, when the dimension d_3 is removed, either w_1 still dominates w_2 and/or w_3 or they do not dominate each other which makes them the candidate skylines. $w_1 \succ w_2$ based on $d_{<remove>}$ in which we cannot conclude whether $w_1 \succ w_2$ based on $d - d_{<remove>}$ without further analysing the values of these dimensions as defined by *Rule* 2*r*. Similar case can be seen between w_1 and w_3.

(b) Based on *DH* given in Table 1, $w_4 \succ \{w_1, w_5, w_6, w_7, w_8, w_9, w_{10}\}$ which implies w_4 is better than the data items listed above with regard to d. However, comparing w_4 and w_5 based on $d_{<remove>}$, we noticed that $w_4 = w_5$, hence $w_4 \succ w_5$ and still has a potential to be a candidate skyline as defined by *Rule* 1*r*.

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

Figure 6: Example of $d_{<remove>}$

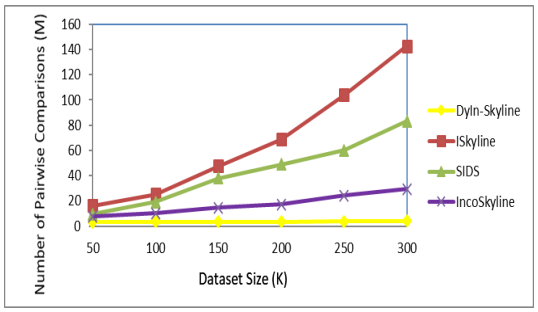
5 Results and Discussion

To fairly evaluate the performance and prove the efficiency of our proposed rule-based approach in processing skyline queries over a dynamic database, we have conducted several preliminary analyses. These analyses are conducted on Intel Core i7 3.6GHz processor with 32GB of RAM and Windows 8 professional. The rule-based solution is embedded into *DyIn-Skyline* [7] and compared to *ISkyline* [21], *SIDS* [24], and *Incoskyline* [1].

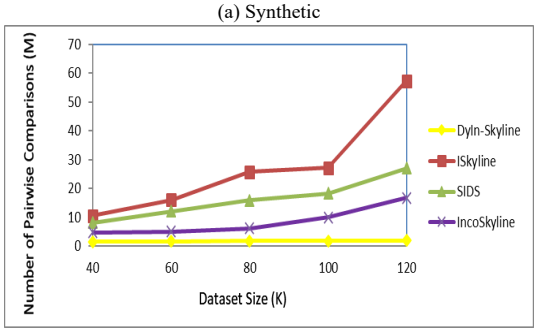
In conducting the analysis, we follow [7], in which we first run each algorithm, namely: *ISkyline*, *SIDS*, and *Incoskyline* over the initial database and derive a set of skylines, S_p. When the database changes its states, we run again each of the above algorithms over the new state of the database and derive a new set of skylines, S_p'. While for our proposed solution, we first run the conventional skyline algorithm over the initial database and derive a set of skylines, *S*. The lists *DH*, *DG*, and *DD* are also derived at this stage. When the database changes its states, depending on the type of modification made, we apply the appropriate rule 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 data sets are used in the analyses, namely: synthetic with 4 dimensions and 50K of initial size; and *NBA* with 13 dimensions and 120K of initial size; while 20% of the data is incomplete. Each analysis is run 10 times and we report the average value of number of pairwise comparisons of these runs.

Figures 7(a) and 7(b) present the number of pairwise comparisons achieved by the rule-based approach embedded into the *DyIn-Skyline* [7], *ISkyline* [21], *SIDS* [24], and *Incoskyline* [1], based on the *synthetic* and *NBA* data sets, respectively. In this analysis, the size of the data set is increased from 50K to 300K for the *synthetic* data set and from 40K to 120K for the *NBA* data set. The size of the data sets is increased by increasing the number of data items in the data sets which reflects changes in the state of the data sets.



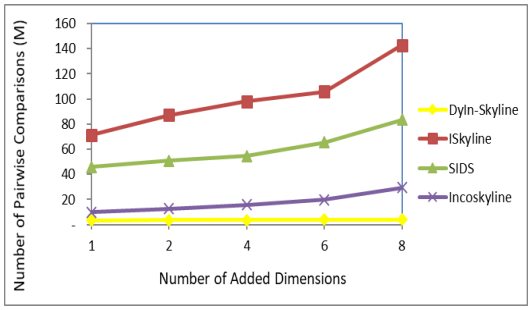
(a) Synthetic



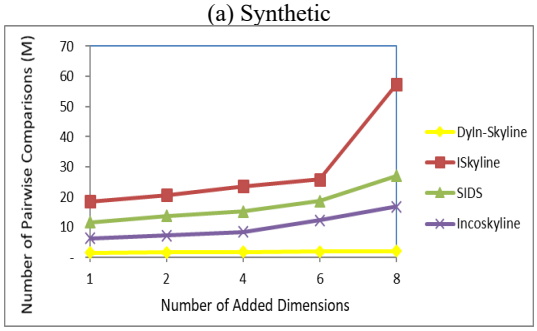
(b) NBA

Figure 7: The results of number of pairwise comparisons with varying data set size

Figures 8(a) and 8(b) present the number of pairwise comparisons achieved by the *DyIn-Skyline*, *ISkyline* [21], *SIDS* [24], and *Incoskyline* [1], based on the *synthetic* and *NBA* data sets, respectively as a number of new dimensions is added to the existing structure of the data sets. Here, the numbers of new dimensions added to the *synthetic* and *NBA* data sets are set to 1, 2, 4, 6, and 8. Meanwhile, figures 9(a) and 9(b) present the number of pairwise comparisons achieved by the *DyIn-Skyline*, *ISkyline* [21], *SIDS* [24], and *Incoskyline* [1], based on the *synthetic* and *NBA* data sets, respectively as a number of dimensions is removed from the existing structure of the data sets. Here, the numbers of dimensions removed from the *synthetic* and *NBA* data sets are set to 1, 2, 4, 6, and 8.

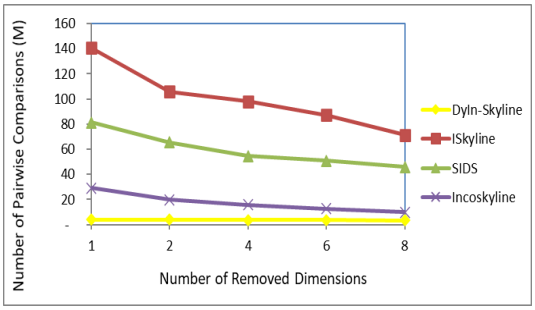


(a) Synthetic

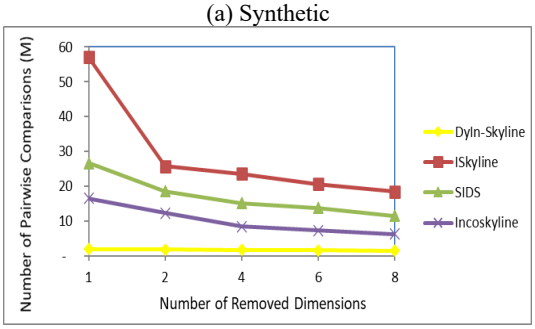


(b) NBA

Figure 8: The results of number of pairwise comparisons with varying number of new added dimensions



(a) Synthetic



(b) NBA

Figure 9: The results of number of pairwise comparisons with varying number of removed dimensions

From the figures 7 – 9, it is obvious that varying the size of the data set, the number of new added dimensions, and the number of removed dimensions does not have a significant impact on the performance of *DyIn-Skyline* which utilised the rule-based approach. It can be seen from the figures that in most cases, *DyIn-Skyline* shows a steady performance, 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 the data items are unaffected by the changes. Meanwhile, the rules that we have derived identify those data items that are affected by the changes, based on the type of operation. For instance, if the changes are based on a new dimension(s) being added, then the pairwise comparisons are performed only based on the values of the added dimensions. This has significantly avoided unnecessary pairwise comparisons and consequently reduces the number of pairwise comparisons performed.

6 Conclusion

In this paper, we proposed a rule-based approach that is capable of deriving skylines over a dynamic database. The rules are designed based on the type of operation that changes the state and/or the structure of a database. These operations include inserting a data item(s) into a database, deleting/updating an existing data item(s) from a database, in which these operations change the state of the database. The proposed rule-based approach can also handle operations that not only change the state but also the structure of the database, namely: adding a new dimension(s) to a database and removing an existing dimension(s) from a database. The proposed rule-based approach is general and can be applied to any skyline algorithms.

ACKNOWLEDGMENTS

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. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Ali A. Alwan, H. Ibrahim, N.I. Udzir, and F. Sidi. 2016. An Efficient Approach for Processing Skyline Queries in Incomplete Multidimensional Database, *Proceedings of the Arabian Journal for Science and Engineering*, Vol. 41, pp. 2927–2943.

[2] Ali A. Alwan, Hamidah Ibrahim, Nur Izura Udzir. 2018. Missing Values Estimation for Skylines in Incomplete Database. *The International Arab Journal of Information Technology*, IAJIT, 15(1): 66–75.

[3] Alwan, A. A., Ibrahim, H., Udzir, N. I., Sidi, F. 2017, Processing Skyline Queries in Incomplete Distributed Databases, *Journal of Intelligent Information Systems*, 48(2): 399–420.

[4] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive Skyline Computation in Database Systems. *ACM Transactions on Database Systems*, 30(1): 41–82.

[5] Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries, *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 28)*, pp. 275–286.

[6] Garrett Wolf, Aravind Kalavagattu, Hemal Khatri, Raju Balakrishnan, Bhaumik Chokshi, Jianchun Fan, Yi Chen, and Subbarao Kambhampati. 2009. Query Processing Over Incomplete Autonomous Databases: Query Rewriting Using Learned Data Dependencies. *The International Journal on Very Large Data Bases*, 18(5): 1167–1190.

[7] Ghazaleh Babanejad Dehaki, Hamidah Ibrahim, Fatimah Sidi, Nur Izura Udzir, Ali A. Alwan, and Yonis Gulzar. 2020. Efficient Computation of Skyline Queries over a Dynamic and Incomplete Database, *Journal of IEEE Access*, 8: 141523–141546.

[8] Ghazaleh Babanejad Dehaki, Hamidah Ibrahim, Nur Izura Udzir, Fatimah Sidi and Ali A. Alwan. 2018. Efficient Skyline Processing Algorithm over Dynamic and Incomplete Database. *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services (iiWAS2018)*, pp. 190–199.

[9] Ghazaleh Babanejad Dehaki, Ibrahim H., Udzir N. I., Sidi F., Alwan A. A., 2017. Deriving Skyline Points over Dynamic and Incomplete Databases. *Proceedings of the 6th International Conference on Computing and Informatics (ICOCI2017)*, pp. 77–83.

[10] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. 2006. SaLSa: Computing the Skyline without Scanning the Whole Sky, *Proceedings of the 15th International Conference on Information and Knowledge Management (ICIKM06)*, pp. 405–414.

[11] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2005. Skyline with Presorting: Theory and Optimizations. *Intelligent Information Systems Journal*, 31: 595–604.

[12] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with Presorting, *Proceedings of the 19th International Conference on Data Engineering (ICDE03)*, pp. 717–816.

[13] Jongwuk Lee, Hyeonseung Im, and Gae-won You. 2016. Optimizing Skyline Queries over Incomplete Data, *Information Sciences*, 361: 14–28.

[14] Kaiqi Zhang, Hong Gao, Xixian Han, Zhipeng Cai, and Jianzhong Li. 2017. Probabilistic Skyline on Incomplete Data, *Proceedings of the 2017 ACM Conference on Information and Knowledge Management*, pp. 427–436.

[15] Kaiqi Zhang, Hong Gao, Xixian Han, Zhipeng Cai, and Jianzhong Li. 2016. ISSA: Efficient Skyline Computation for Incomplete Data. *Proceedings of the* International *Conference on Database Systems for Advanced Applications*, pp. 321–328.

[16] Kian-Lee Tan, E. Pin-Kwang, and C.O. Beng. 2001. Efficient Progressive Skyline Computation, *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB27)*, pp. 301–310.

[17] Maria Kontaki, Apostolos N. Papadopoulos, and Yannis Manolopoulos. 2010. Continuous Processing of Preference Queries in Data Streams, *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 47–60.

[18] Marwa B. Swidan, Alwan A. A., Turaev S., Ibrahim H., Abualkishik A. Z., Gulzar Y. 2020. Skyline Queries Computation on Crowdsourced-Enabled Incomplete Database. *Journal of IEEE Access*, 8: 106660–106689.

[19] Michael D. Morse, Jignesh M. Patel, and William I. Grosky. 2007. Efficient Continuous Skyline Computation, *Information Science Journal*, 177(17): 3411–3437.

[20] Mohamed A. Soliman, Ihab F. Ilyas., Shalev, and Ben-David. 2010. Supporting Ranking Queries on Uncertain and Incomplete data. *The Very Large Database Journal, VLDB*, 19(4): 477–501.

[21] Mohamed E. Khalefa, Mohamed F. Mokbel, and Justin J. Livandoski. 2008. Skyline Query Processing for Incomplete Data, *Proceedings of the 24th International Conference on Data Engineering (ICDE 2008)*, pp. 556–565.

[22] Mohammad Shamsul Arefin and Y. Morimoto. (2012). Skyline Sets Queries for Incomplete Data. *International Journal of Computer Science & Information Technology*, 4(5): 67.

[23] Parke Godfrey. 2005. Skyline Cardinality for Relational Processing. *Foundations of Information and Knowledge Systems*. 2942: 78–97.

[24] Rahul Bharuka and P. Sreenivasa Kumar. 2013. Finding Skylines for Incomplete Data. *Proceedings of the Twenty-Fourth Australasian Database Conference*, Vol. 137, pp. 109–117.

[25] Raymond Chie-Wing Wong, Ada Wai-Chee Fu, Jian Pei, Yip Sing Ho, Tai Wong, and Yubao Liu. 2008. Efficient Skyline Querying with Variable User Preferences on Nominal Attributes, *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB 34)*, pp. 1032–1043.

[26] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator, *Proceedings of the 17th International Conference on Data Engineering (ICDE01)*, pp. 421–430.

[27] Xiaoye Miao, Yunjun Gao, Gang Chen, and Huiyong Cui (2015). Top-*k* Dominating Queries on Incomplete Data. *IEEE Transactions on Knowledge and Data Engineering*, 28(1): 252–266.

[28] Xiaoye Miao, Yunjun Gao, Gang Chen, and Tianyi Zhang. 2016. *k*-dominant Skyline Queries on Incomplete Data. *Information Sciences*, 367: 990–1011.

[29] Yonis Gulzar, Ali A. Alwan, Norsaremah Salleh, Imad Fakhri Al Shaikhli, and Syed Idrees Mairaj Alvi. 2016. A Framework for Evaluating Skyline Queries over Incomplete Data. *Procedia Computer Science*, 94: 191–198.

[30] Yonis Gulzar, Alwan, A. A., Ibrahim, H., and Xin, Q., 2018. D-SKY: A Framework for Processing Skyline Queries in a Dynamic and Incomplete Database. *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services (iiWAS2018)*, pp. 164–172.

[31] Yonis Gulzar, Alwan A. A., and Sherzod Turaev. 2019. Optimizing Skyline Query Processing in Incomplete Data. *Journal of IEEE Access*, 7: 178121–178138.