International Journal on Perceptive and Cognitive Computing (IJPCC) Vol 11, Issue 2 (2025)
https://doi.org/10.31436/ijpcc.v11i2.595

Efficient Skyline Query Processing in Incomplete Graph Databases Using Machine Learning Techniques

Ubair Noor, Raini Binti Hassan, Dini Oktarina Dwi Handayani
Department of Computer Science, International Islamic University Malaysia,
Kuala Lumpur, 53100, Malaysia

*Corresponding author: hrai@iium.edu.my
(Received: 12th June 2025; Accepted: 2nd July, 2025; Published on-line: 30th July, 2025)

Abstract— Skyline queries play a critical role in multi-criteria decision-making systems by retrieving non-dominated data points from large datasets. In recent years, the rapid growth of graph-structured data across various domains has introduced challenges in efficiently processing skyline queries over incomplete and large-scale graph databases. Processing skyline queries in such massive, incomplete graphs is computationally intensive due to missing values and high-dimensional data. Traditional techniques often fail to scale or effectively handle data imperfections. There is a pressing need for a scalable, intelligent framework that can manage missing data, reduce computational overhead, and improve skyline query efficiency. This study adopts the Design Science Research Methodology (DSRM) to design and implement an optimisation framework that integrates machine learning techniques, including domination score ranking, dimension-based filtering, K-Means clustering and quicksort. These methods collectively reduce the search space and redundant comparisons. Experimental evaluation on real graph datasets demonstrates significant improvements in skyline computation time and accuracy, with clear reductions in pairwise comparisons and improved processing efficiency on large-scale graphs. By leveraging machine learning techniques for sorting, filtering and clustering, the approach reduces computational complexity and enhances scalability. These results show promising directions for applying intelligent query optimization in big data environments.

Keywords— Skyline queries, Incomplete graph database, Machine learning, Graph database

I. INTRODUCTION

These Skyline queries are used in database systems to retrieve non-dominated tuples data points that are not dominated by any other nodes [1]. In graph databases, this means identifying nodes that are optimal based on attributes such as distance, cost or relevance, making skyline queries particularly useful in applications like recommendation systems, e-commerce, road networks and urban planning.

A big challenge happens when graph databases contain incomplete data [2] [3] [4]. These missing values fail the transitivity of dominance relationships, which is foundational to skyline computations. This can lead to cyclic comparisons and ambiguous dominance, significantly increase the complexity of processing queries. Despite the widespread use of skyline queries in practice, limited research has addressed how to efficiently compute skylines when dealing with incompleteness in graph-based datasets.

Graphs in real-world applications are often dynamic and sparse, where nodes frequently lack values in one or more dimensions. For example, in a hotel recommendation system, a user may want to identify hotels near the beach with affordable prices. If some hotels are ratings or price information, they still might be valuable candidates depending on the available data. Traditional skyline algorithms often leave out these incomplete entries, which potentially eliminates useful information from the results.

Processing skyline queries efficiently over incomplete graph databases thus requires innovative techniques which can reduce the computational cost, handle missing values without compromising the accuracy of results and adapt to high-dimensional and constantly changing data. This study aims to tackle these challenges by proposing a method which integrates machine learning techniques particularly clustering to enhance skyline query performance. Machine learning can help infer patterns from incomplete data, cluster similar nodes to narrow the search space and dynamically adapt to query updates, thus making skyline processing more accurate and scalable.

To address the limitations of existing approaches, the following objectives and contributions of the study are proposed:

- To design and develop an efficient data pruning approach tailored for incomplete graph databases.

- To leverage machine learning techniques for enhanced performance and scalability.
- To evaluate the effectiveness of the proposed pruning technique through empirical experiments, comparing its performance with existing baseline methods in terms of accuracy, efficiency and computational cost.

A. *Summary of Contribution*

- The proposed study has introduced a unified framework which combines quick-sort based domination scoring, threshold-filtering and K-Means clustering to optimize skyline query processing.
- A development of a clustering mechanism which groups nodes based on missing dimensions to enable effective skyline computation with requiring data imputation.
- A proposed local and global skyline identification method which reduces unnecessary pairwise comparison to ensure transitivity while avoiding cyclic dominance.
- A demonstration on the improvements in processing time and data pruning with experiments on different datasets of varying sizes.

II. Related Works

A skyline query optimization in graph databases has been advanced significantly especially for static environments. The pruning technique in [5] uses hierarchical labels to reduce overhead but is limited by its need for re-computation in dynamic graphs. The study by [6] supports dynamic queries with local distance functions but suffers from high computational costs as graph complexity grows. Similarly, a hybrid approach in [7] combines subgraph isomorphism with dual traversal however, scalability in high-dimensional graphs remained a challenge.

Moreover, the algorithmic divide-and-conquer method in [8] improves on nested loop approaches by partitioning the problem space and minimizing redundant comparisons thus boosting performance and accuracy. Compared to pruning in [9] and subgraph merging in [7], [8] which demonstrates superior efficiency especially in large-scale and moderately dynamic graphs. However, it lacks adaptability to real-time changes and user-defined preferences which increasingly demands an interactive application. This comparison reveals that while algorithmic performance remains a core priority, practical implementations must also factor inflexibility and dynamic responsiveness.

Furthermore, the handling of incomplete data in skyline queries possesses a unique challenge particularly in ensuring accurate dominance comparisons. The method in [8] utilizes Approximate Functional Dependencies (AFDs) to infer missing values followed by ranking based on dependency strength which is a technique that enhances the semantic richness of imputations. While effective in preserving skyline correctness however, this method can be computationally expensive due to repeated AFD generation. On the other hand, [16] adopts a more parallel-friendly framework by clustering nodes with similar missing patterns and applying bitwise skyline filters. This not only improves processing speed but also upholds the transitivity of skyline dominance, a property often lost in simpler imputation methods. Also, building on these foundations, [17] introduces a dominance-aware clustering and pruning technique that further scales skyline computation by minimizing redundant comparisons. Despite their differences, these methods reflect a shared emphasis on balance precision, performance and scalability, though none fully resolves the complexities of high-dimensional or real-time incomplete data handling.

Real-time skyline path queries in dynamic networks require algorithms that are both fast and responsive. The PSQ+ algorithm from [10] has introduced a refined pruning mechanism which discards non-skyline paths during graph traversal thus maintaining real-time efficiency even in bicriteria networks. This method marks a significant departure from more static approaches like those in [8] or [11], as it actively adapts to the changing cost landscape of paths. Supporting this, [11] improves the credibility of skyline query results through POI signature-based authentication, ensuring the integrity of results in outsourced databases. Meanwhile, [12] tackles user-centric issues such as incomplete skyline results by implementing a reverse-query mechanism that identifies potential missing tuples based on adjusted preferences. While these methods collectively push skyline path queries closer to real-time user-aware applications however, they also introduce new overheads in preprocessing, verification and system complexity that must be managed carefully.

In distributed environments, skyline computation must balance network communication, data partitioning and computation cost [20]. Approaches like BDS and IDS optimize node access and reduce comparisons, but struggle with dynamic networks. More advanced methods, such as PDS and iSky, use probabilistic models and adaptive filters to prune irrelevant data early and improves performance. However, their effectiveness depends on initial data distribution and network topology which are not always controllable. Despite having a progress, full scalability and fault tolerance in dynamic scenarios remains open challenge.

Skyline query processing under uncertainty becomes more complex in dynamic settings with frequent updates. The method in [13] extends skyline queries to uncertain graphs using expected distances and probabilistic pruning, enhancing decision but increase computational costs (See Table 1). The IDSA algorithm[14] handles dynamic changes

https://doi.org/10.31436/ijpcc.v11i2.595

with multi-phase pruning and block nested loops, but exhaustive dominance checks can hinder performance on large graphs. While effective in real-time, noisy environments, these methods highlight the need for lighter or incremental models to improve scalability and responsiveness.

TABLE I
EXISTING LIMITATIONS AND GAP ANALYSIS

	Limitation	Gaps
[9]	High memory usage, lacks support for dynamic graphs	Needs optimization for non-Euclidean and real-time use
[6]	Computationally expensive, lacks general graph support	Scalability and real-time processing
[7]	Struggles with large graphs, complex constraints	Unified large-scale graph handling
[5]	Inefficient nested loop, compute-heavy	Optimization for complex graph queries
[8]	Heavy preprocessing, not scalable with many POIs	Large dataset handling with missing data
[2]	Expensive in complex networks	Lightweight path skyline algorithms
[11]	Preprocessing burden, slow with many POIs	Scalable authentication methods
[12]	Not efficient for large-scale data	Better performance for big data
[3]	Struggles with cycle dominance and dimensionality	Real-time complex missing data handling
[17]	Poor performance on high-dimensional datasets	Optimization for high complexity
[16]	Complexity in distance calculations across layers	Scalability in multi-layer graphs
[17]	Underperforms with anti-correlated data, dimension bottlenecks	Dimensionality handling
[18]	Scalability and communication bottlenecks	Robust-distributed skyline methods
[13]	Multi-phase algorithm complexity	Simplified dynamic skyline algorithms
[15]	Complex, constrained by time and label ranges	Scalable temporal skyline querying
[19]	MapReduce overhead, partition imbalance	Balanced distributed skyline computing
[20]	Struggles with sparse data, normalization issues	Effective QoS partitioning
[21]	Not memory-efficient for large datasets	Support for dynamic updates
[22]	Inter-bucket comparison slows things down	Efficient global skyline merging

Also, the temporal and attribute-rich graphs require advanced skyline processing. The TMP algorithm [15] uses bidirectional search and time-aware indexing to efficiently find skyline paths under temporal and label constraints outperforming traditional methods. The probsky [19] built on mapreduce, handles probabilistic skyline queries using slab partitioning and reference-point acceleration for scalability, though it suffers from high signature generation costs. These methods highlight the need to combine temporal awareness, uncertainty handling and distributed computing, while also raising concerns about preprocessing overhead and integration complexity.

III. METHODOLOGY

This study adopts the design science research methodology (DSRM) [23] (see Figure1), which is a structured framework to design, develop, and evaluate innovative IT artefacts to address real-world problems. This study utilizes DSRM to propose a machine learning-based approach for skyline query optimization in a large-scale incomplete graph database. It emphasized both practical relevance and theoretical contribution. The proposed methodology aligns with the six stages of DSRM to address the objective and goals of this study.

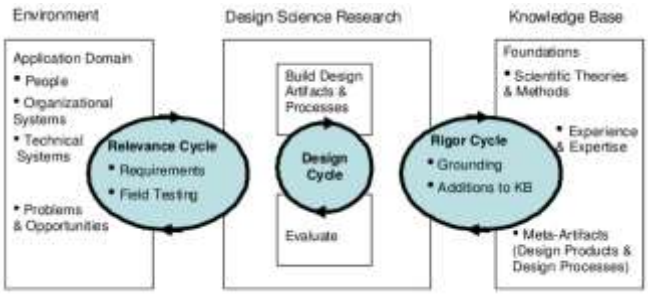

Fig 1. DSRM Process ([23])

A. *Design and development*

This phase of skyline query optimization comprises five key components as shown in Fig 2 and Table 2. Each component plays an important role in ensuring computational efficiency and maintaining the accuracy of the skyline query optimized results.

1) Dataset development: A synthetic dataset was developed to evaluate the proposed skyline solution. The reason for using the synthetic dataset is due to the lack of availability of a real dataset concerning the skyline query problem. The dataset was designed to include missing values randomly, simulating data incompleteness, which is common in large-scale graph databases. The dataset comprises 51 nodes connected to form a graph. The dataset simulates a hotel booking scenario to assist customers in finding recommendations for the best hotel room.

https://doi.org/10.31436/ijpcc.v11i2.595

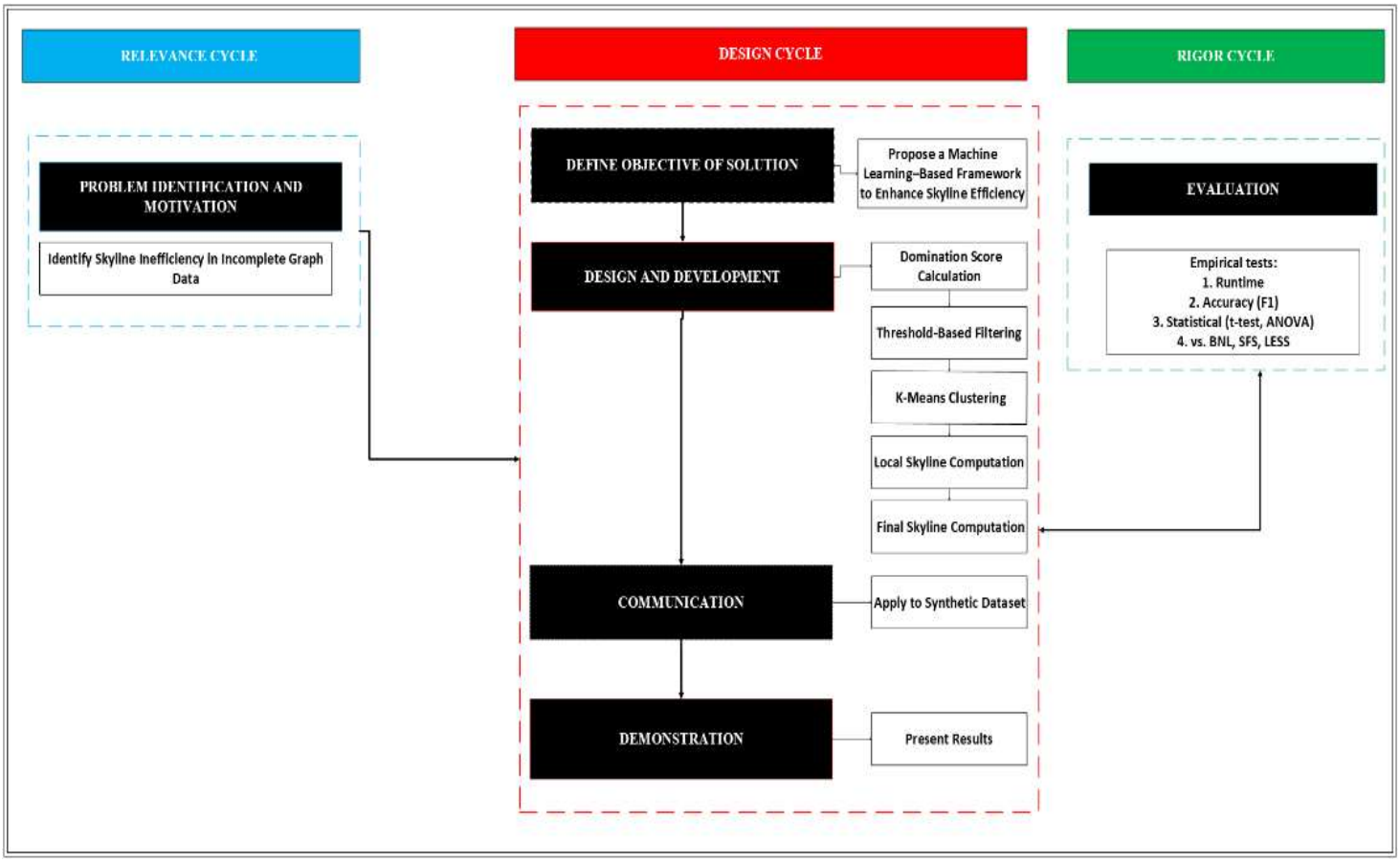

Fig 2. Proposed Methodology based on DSRM

The attributes of the dataset include price, rating, distance, and hotel service. The dataset has a comprehensive number of nodes, each comprised of systematically randomized values, achieving the major objective of the dataset development. Also, the dataset development aligns with its application to be used in machine learning algorithms, to make sure that the data preprocessing was performed earlier during the dataset development phase, addressing null value duplication and irrelevant data, respectively. The dataset is highly suited for machine learning purposes, particularly concerning the skyline query problem and its potential optimization solution for graph databases.

2) *Sorting and filtering:* This phase arranges the data nodes in descending order based on their domination power, which reflects the total number of other nodes that dominate across all dimensions. A round-robin traversal method is used to calculate domination scores, considering only non-missing values to ensure fairness and accuracy. The framework enables the early elimination of less relevant nodes, reducing the number of pairwise comparisons required during skyline computation. This significantly lowers computational overhead and ensures that only the most promising candidates proceed to the next stages.

TABLE II
STAGES OF PROPOSED METHODOLOGY

1	Problem identification and motivation	Inefficient skyline queries in incomplete graph databases present significant challenges, primarily due to missing data and scalability concerns.
2	Design objective of a solution	This study proposes a machine learning–based pruning and clustering framework aimed at improving accuracy and performance.
3	Design and development	A five-step framework was developed, consisting of sorting, filtering, clustering, local skyline detection, and final skyline computation.
4	Demonstration	The proposed framework was evaluated using synthetic graph datasets to assess its practical utility.
5	Evaluation	Experimental results have indicated that the proposed framework significantly reduced processing time and improved scalability.
6	Communication	This research presents the findings to support further exploration within academic and technical communities.

https://doi.org/10.31436/ijpcc.v11i2.595

Similarly, the filtration process excludes nodes whose domination power falls below a user-defined threshold, as such nodes are unlikely to contribute meaningfully to the skyline. This threshold-based pruning complements the sorting phase by reducing unnecessary computations early in the process. The threshold can be adjusted based on dataset characteristics or user requirements to balance precision and efficiency. Moreover, the framework remains adaptable and other sorting or filtering can be integrated, as long as they align with the core principles. This design ensures robustness and flexibility across varied, incomplete graph datasets.

3) Clustering: The objective of this phase is to establish clusters among data items based on their domination power, which enables a more efficient skyline query processing. The data items with similar domination power are grouped, resulting in the formation of distinct clusters. The K-Means clustering was considered due to its real data points consideration as representatives of each cluster. This feature is particularly useful in ensuring that the clusters accurately reflect the dataset and make the skyline computation more precise. To validate the clustering model, k-fold cross-validation is used in which the dataset is split into *k* subsets, and the model is trained on *k - 1* subsets and tested on the remaining one. The model performance is evaluated using the silhouette score and Davies-Bouldin index to assess the quality of the clusters formed, ensuring that the final skyline computation is robust and accurate. Lastly, the search space significantly reduces the gaps, which helps to avoid unnecessary comparisons while maintaining the accuracy of the final skyline results.

4) *Identifying Local Skyline:* It is intended to retrieve local skylines for each of the constructed clusters, which leads to prevents many dominated data items from further processing, resulting in a reduction in the processing time. Also, it ensures the transitivity property of the skyline solution holds as all data items in one cluster have a similar namespace. The parallel process will be in all clusters, which will reduce the processing time between the data elements.

5) Final Skyline: The last component is responsible for determining the final skyline. The process starts by comparing those local skylines generated earlier and retrieving those undominated data items as the final skylines of the entire incomplete graph database. This component ensures that any reported global skylines are the skylines over the entire database, and no other data items might dominate them.

B. *Evaluation*

This phase aims to determine how well the framework meets the objectives set out in this study, particularly in the context of skyline query optimization for incomplete graph databases. The evaluation is carried out by comparing the proposed framework with traditional skyline query methods and evaluating key performance metrics such as scalability, accuracy, and efficiency. Similarly, the integration of sorting and filtering techniques will be evaluated to understand their performance in reducing the dataset size before skyline computation. By prioritizing more influential nodes and eliminating irrelevant data points, the proposed framework will be evaluated on the reduction of its search space for efficient skyline processing. Moreover, the framework will be evaluated further by applying threshold-based filtering to remove dominated nodes early in the process. By evaluating these nodes, which are less likely to contribute to the final skyline result, the proposed framework will be analyzed in terms of the computational cost required for skyline computation.

IV. Experimental Analysis and Results

The proposed framework was implemented using Python, leveraging a range of machine learning libraries to support the required functionalities. Similarly, to implement clustering and support data processing tasks, the Scikit-learn package was used for clustering with the K-means model, data processing, and model evaluation. The pandas and numpy packages were utilized for efficient data manipulation, specifically in handling the incomplete dataset having one missing dimension. The numpy provided support for numerical operations, while pandas facilitated data cleaning, sorting and filtering, ensuring seamless integration across the different steps in skyline query optimization. Moreover, the implementation was modular, allowing each technique to be independently tested and optimized. The next step was to execute a series of experiments to assist its performance. These experiments were conducted on datasets of varying sizes to evaluate how well the framework performs under different conditions.

	Node	Dim1	Dim2	Dim3	Dim4
0	1	2	8	0	4
1	2	3	0	8	1
2	3	3	2	0	2
3	4	5	0	2	8
4	5	8	1	0	6
5	6	4	0	7	3
6	7	0	8	2	9
7	8	7	0	8	4
8	9	0	6	4	2
9	10	5	2	6	0
10	11	8	6	4	0

Fig. 3. Initial Dataset

https://doi.org/10.31436/ijpcc.v11i2.595

Algorithm 1 Quick Sort Nodes

```
Input: An array A of n nodes
Output: Sorted array A
function QuickSort(A, low, high)
    if low < high then
        p ← Partition(A, low, high)
        QuickSort(A, low, p − 1)
        QuickSort(A, p + 1, high)
    end if
end function
function Partition(A, low, high)
    pivot ← A[high]
    i ← low − 1
    for j ← low to high − 1 do
        if A[j] ≤ pivot then
            i ← i + 1
            Swap A[i] and A[j]
        end if
    end for
    Swap A[i + 1] and A[high]
    return i + 1
end function
```

Fig. 4. Quicksort algorithm implementation

A. *Sorting and filtering analysis*

The process starts by sorting the items in each distinct list based on values of each dimension on the dataset as shown in Fig. 3 The quick sort algorithm was used in this experiment as shown in Fig. 4 for its divide-and-conquer strategy. The partitioning step efficiently divides the dataset into manageable subsets, reducing operational complexity. Using the last element as a pivot ensures consistency and minimizes redundant operations. Node 1 is read with its domination power, which is increased by 1, which means it is compared with other nodes in the given dimension. The dimension 1 as shown in Fig. 5(a), Nodes 5 and 11 have the highest score of 8, followed by Node 8 with a score of 7. The dimension 2 as shown in Fig. 5(b), Nodes 1, 7, and 6 dominate while in dimension 3 as shown in Fig. 5(c), Nodes 2, 8, and 6 take the lead. The dimension 4 as shown in Fig. 5(d), Nodes 7 and 4 have the highest domination scores. The process terminated after dimension 4 with Node 11. The total number of iterations comprises 44th in number. The lowest score of 0 is assigned to Node 9 in dimension 1, where no comparison can be made due to its value being zero or empty. The constructed lists are scanned round-robin style for each data item to determine its domination power. Domination scoring of all nodes occurs one after the other until each node receives its dominance value.

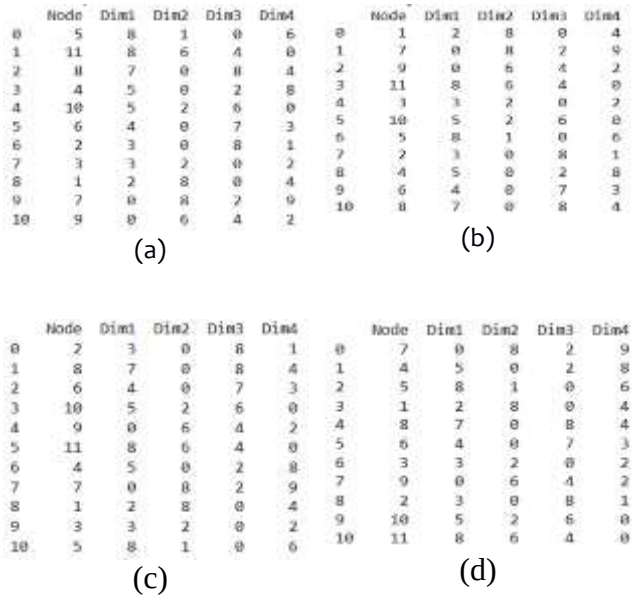

Fig. 5. Domination power calculation

B. *Threshold-based filtering*

This process has removed the nodes which comprise of low domination score and had minimal impact on the skyline computation. Also, it reduces the size of the data which is essential in decreasing computational overhead and enables faster processing. Moreover, the removal of less influential nodes makes skyline query processing more significant and efficient. Additionally, filtration is an equally crucial step aimed at narrowing the search space by discarding nodes which are unlikely to contribute to the skyline. The experiment uses a threshold-based approach which maximizes efficiency while ensuring relevant candidates are retained as shown in Fig. 6 and the algorithm is shown in Fig. 7. The focus is on a limited subset of nodes on each dimension; this approach efficiently controlled the exponential growth of pairwise comparisons. Moreover, the stopping condition introduced a mechanism for early termination to ensure runtime was minimized without compromising accuracy.

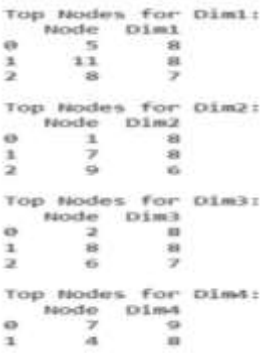

Fig. 6. Top threshold nodes

https://doi.org/10.31436/ijpcc.v11i2.595

Algorithm 2 Filter Nodes with Threshold

```
Input: Dictionary sorted_nodes, where each value has sorted nodes
Output: Dictionary nodes_with_threshold, containing nodes for each dimension
nodes_with_threshold ← empty dictionary
for all Dimensions, sorted_nodes in sorted_nodes do
    top_nodes ← empty list
    top_values ← empty set
    value_counts ← empty dictionary
    for all row in sorted_nodes do
        value ← row[Dimensions]
        node ← row["Node"]
        if value not in value_counts then
            value_counts[value] ← 0
        end if
        if value_counts[value] < 2 then
            Append (node, value) to top_nodes
            value_counts[value] ← value_counts[value] + 1
            Add value to top_values
        end if
        if len(top_values) == 2 then
            break
        end if
    end for
    nodes_with_threshold[dim] ← DataFrame of top_nodes with columns ["Node", Dimensions]
end for
    return nodes_with_threshold
```

Fig. 7. Threshold-based filtering

The threshold-based filtration eliminates the data items with a domination power lower than a user-defined threshold. These data items are discarded because their domination score indicates that they perform well in no more than one dimension, making them unlikely to be part of the skyline result. Dimension 1 with Nodes such as Nodes 5, 11 and 8 remain there, while in dimension 2, Nodes 1, 7 and 9 are still considered. Dimension 3 with Nodes 2, 6, and 8 have domination powers greater than the threshold, and in dimension 4, Nodes 7 and 4 are still in consideration. These nodes have domination powers greater than the threshold value and thus remain in consideration for the skyline result.

Algorithm 3 Final Nodes Filtered with Threshold

```
Input: Dictionary nodes_with_threshold, containing nodes for each dimension
Output: DataFrame final_filter_nodes, containing the final nodes without duplicates
final_filter_nodes ← Empty DataFrame with column "Node"
for all Dimension, Nodes in nodes_with_threshold do
    for all row in Nodes do
        if row["Node"] not in final_nodes["Node"] then
            Create a new DataFrame with row["Node"]
            Concatenate the new DataFrame to final_filter_nodes
        end if
    end for
end for
    return final_top_nodes
```

Fig. 8. Final nodes filtered with a threshold algorithm

In the example from Fig. 8, after applying the algorithm in Fig. 9, the filtration process and setting a threshold value, there were identified 9 nodes as eligible for the skyline as shown in Fig. 10. These nodes represent approximately 81.82% of the total 11 nodes in the dataset. The remaining nodes, which didn't meet the threshold criteria are not considered in the skyline calculation.

Algorithm 4 Final Nodes with Threshold Assigned with Actual Values

```
Input: Dictionary sorted_nodes, where each value is a DataFrame
Output: Dictionary final_nodes_with_threshold, containing top nodes for each dimension
final_nodes_with_threshold ← empty dictionary
for all Dimensions, sorted_nodes in sorted_nodes do
    top_nodes ← empty list
    top_values ← empty set
    value_counts ← empty dictionary
    for all row in sorted_nodes do
        value ← row[Dimensions]
        node ← row["Node"]
        if value not in value_counts then
            value_counts[value] ← 0
        end if
        if value_counts[value] < 2 then
            Append (node, value) to top_nodes
            value_counts[value] ← value_counts[value] + 1
            Add value to top_values
        end if
        if len(top_values) == 2 then
            break
        end if
    end for
    final_nodes_with_threshold[Dimensions] ← DataFrame of top_nodes with columns ["Node", Dimensions]
end for
    return final_nodes_with_threshold
```

Fig. 9. Final nodes with dataset values

	Node	Dim1	Dim2	Dim3	Dim4
0	5	8	1	0	6
1	11	8	6	4	0
2	8	7	0	8	4
3	1	2	8	0	4
4	7	0	8	2	9
5	9	0	6	4	2
6	2	3	0	8	1
7	6	4	0	7	3
8	4	5	0	2	8

Fig. 10. Results of eligible nodes for clustering

C. *Machine learning-based clustering*

This process aims to group nodes that exhibit zero values in specific dimensions. The nodes with zero values often hold distinctive properties that warrant separate analysis. The algorithm in Fig 11 begins with nodes containing dimensional values and a list of dimensions to analyze. It iteratively examines each dimension to identify nodes with zero values, excluding any nodes which are excitingly assigned to clusters. Also, for the nodes which are isolated, the algorithm performs K-Means clustering to organize the nodes into clusters depending on the number of nodes available. This clustering step ensures that the nodes are grouped based on their similarity in dimensional values making subsequent processing more efficient. The final clusters annotated with their labels are stored in a structured format which develops the basis for further analysis. The example from the experiment as shown in Fig. 12 results of clustering based on the proposed model.

https://doi.org/10.31436/ijpcc.v11i2.595

Algorithm 5 Cluster Nodes Based on Zero Values

Input: DataFrame `final_top_nodes_with_values`, List of dimensions `["Dim1", "Dim2", "Dim3", "Dim4"]`
Output: Dictionary `clusters`, containing the nodes clustered by each dimension
`clusters` ← Empty dictionary
`used_nodes` ← Empty set ▷ To keep track of nodes already assigned to clusters
for all `Dimensions` in `["Dim1", "Dim2", "Dim3", "Dim4"]` **do**
  `zero_values` ← Rows in `final_top_nodes_with_values` where `Dimensions` is 0 and `Node` not in `used_nodes`
  if `zero_values` is not empty **then**
    `n_clusters` ← Minimum of 4 and the length of `zero_values`
    Perform KMeans clustering with `n_clusters` on the columns `"Dim1", "Dim2", "Dim3", "Dim4"`
    `cluster_labels` ← Result from KMeans clustering
    Assign `cluster_labels` to the `zero_values` DataFrame in a new column `Cluster`
    `clusters[Dimensions]` ← Subset of `zero_values` with columns `"Node", "Dim1", "Dim2", "Dim3", "Dim4", "Cluster"`
    Update `used_nodes` by adding the nodes from `zero_values["Node"]`
  end if
end for
return `clusters`

Fig. 6. Clustering algorithm

Clusters for Dim1 (Nodes with Zero Values):

	Node	Dim1	Dim2	Dim3	Dim4	Cluster
4	7	0	8	2	9	0
5	9	0	6	4	2	1

Clusters for Dim2 (Nodes with Zero Values):

	Node	Dim1	Dim2	Dim3	Dim4	Cluster
2	8	7	0	8	4	2
6	2	3	0	8	1	0
7	6	4	0	7	3	3
8	4	5	0	2	8	1

Clusters for Dim3 (Nodes with Zero Values):

	Node	Dim1	Dim2	Dim3	Dim4	Cluster
0	5	8	1	0	6	0
3	1	2	8	0	4	1

Clusters for Dim4 (Nodes with Zero Values):

	Node	Dim1	Dim2	Dim3	Dim4	Cluster
1	11	8	6	4	0	0

Fig. 7. Clustering results

D. *Local skyline computation*

This step enables the identification of the local skyline node from each cluster, simplifying the clustered data into a small set of candidates for local skyline query processing. The local skyline is determined by identifying nodes that are not dominated by any other node within the cluster. The node with the highest score in each cluster is selected as the representative, which captures the most important characteristics of that cluster, as shown in Fig 13.

Applying clustering techniques while identifying the local skyline phase assists in eliminating many dominated data nodes. Based on the given example it is obvious that 4 nodes are left out of the remaining 9 nodes as shown in Fig. 14. This represents the 44.44% reduction in the dataset.

E. *Final skyline computation*

This final skyline aggregation approach mainly addresses the issue of skyline query processing on incomplete graph data. The goal is to select the final skyline of the entire dataset. In the experiment, a set of nodes were retrieved which stand out in at least one dimension and are not inferior in all dimensions to any other node. These final skyline nodes are the most significant for further analysis and decision-making. The process compares each node as shown in Fig. 15 in the dataset with every other node to derive these final skyline nodes.

s illustrated in Figure 16, Node 7 was compared with Nodes 5, 11, and 8, and was found to be dominated by Nodes 5 and 9. Node 11 dominated Node 8, while neither Node 7 nor Node 11 dominated each other qualifying both for inclusion in the final skyline. This step, central to machine learning–driven skyline analysis, filters out redundant nodes and retains only distinct, high-value candidates, thereby reducing noise and supporting effective decision-making. This final step ensures that only the nodes that are not dominated by any others across the entire dataset are included in the skyline, representing the best or most significant nodes in the context of the skyline query.

Algorithm 6 Identify Single Nodes on each Cluster

Input: Dictionary `clusters`, where each value contains cluster information for nodes
Output: DataFrame `final_result`, containing the final selected nodes from each cluster
`final_clusters` ← Empty dictionary
for all `Dimensions`, `cluster_nodes` in `clusters` **do**
  `scores` ← Empty dictionary ▷ To store scores for each node
  for all `node_n` in `cluster_nodes` **do**
    `scores[node_n["Node"]]` ← 0 ▷ Initialize score for the current node
    for all `node_n` in `cluster_nodes` **do**
      if `node_n["Node"]` ≠ `node_n["Node"]` **then**
        for all `col` in `["Dim1", "Dim2", "Dim3", "Dim4"]` **do**
          if `node_n[col]` > `node_n[col]` **then**
            `scores[node_n["Node"]]` ←
            `scores[node_n["Node"]]` + 1
          end if
        end for
      end if
    end for
  end for
  `max_score_node` ← Node with the highest score from `scores`
  `final_clusters[Dimensions]` ← Subset of `cluster_nodes` where `Node` equals `max_score_node`
end for
`final_result` ← Concatenation of `final_clusters`
return `final_result`

Fig 8. Single node cluster identification algorithm

	Node	Dim1	Dim2	Dim3	Dim4	Cluster
4	7	0	8	2	9	0
2	8	7	0	8	4	2
0	5	8	1	0	6	0
1	11	8	6	4	0	0

Fig. 9. Local skyline results

F. *Performance evaluation*

The performance evaluation of the experimental results of skyline queries in incomplete graph databases was performed on synthetic datasets. This set of experiments aims at examining the effect of data size on datasets, and on the processing time that needs to be performed during the skyline query process over an incomplete graph database.

1) Effect on Size of Dataset: The proposed method reduces the dataset size by an average of 50% before the

https://doi.org/10.31436/ijpcc.v11i2.595

final skyline selection. It evaluates node comparisons in a synthetic dataset using two sizes: 11 and 51 nodes as shown in Fig. 17 and Fig. 18. The results have shown that larger datasets require more pairwise comparisons, increasing processing time. However, the approach effectively prunes ineligible nodes, systematically reducing computational workload and maintaining high efficiency. Despite dataset growth, the method shows minimal impact on processing time, demonstrating strong scalability and robustness for incomplete skyline processing in graph databases.

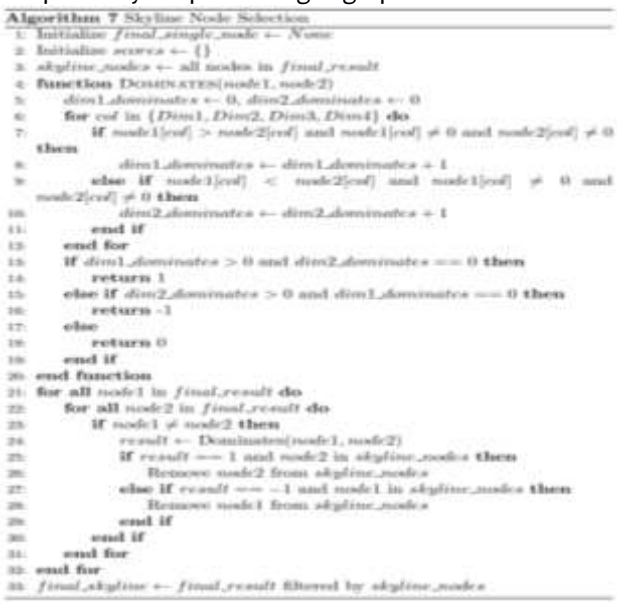

Fig 10. Selecting the final skyline node algorithm

	Node	Dim1	Dim2	Dim3	Dim4	Cluster
4	7	0	8	2	9	0
1	11	8	6	4	0	0

Fig. 11. Final Skyline results

2) Effect on Processing Time: The proposed machine learning-based approach reduces query processing time by 30–50% compared to traditional methods. This is achieved by clustering the dataset, allowing skyline queries to operate on smaller, more relevant subsets, thus minimizing unnecessary pairwise comparisons. This streamlines computation and ensures quicker, optimized execution. The approach also demonstrates high scalability, maintaining efficiency as dataset size grows unlike traditional methods, which face exponential increases in processing time. This makes the proposed method well-suited for large-scale, dynamic environments with expanding data.

As shown in Fig. 18 for the 51-node dataset, processing time decreases as data size increases. The proposed approach consistently outperforms previous methods in all scenarios, showing minimal sensitivity to data size. By effectively pruning ineligible nodes, it ensures efficient computation even with large datasets. Although more nodes usually increase data exchange and latency, the proposed method minimizes processing time, demonstrating strong robustness and scalability for large-scale data handling.

As shown in Fig 19 and 20, the graphs for both 11-node and 51-node datasets exhibit similar patterns. Processing time peaks at Dimension 2 (0.0225 sec for 51 nodes vs. 0.020 sec for 11 nodes), drops sharply at Dimension 3 and remains mostly stable at Dimension 4. While the larger dataset shows slightly higher processing times in the lower dimensions, it outperforms the smaller dataset at Dimension 4 likely due to better utilization of resources at scale. This suggests that scalability benefits become more apparent beyond a certain complexity threshold, where the initial overhead is offset by improved performance in higher dimensions. The entire performance comparison and its impact on the execution runtime will be as shown in Table 3 between both the datasets utilized for performance study.

TABLE III
COMPARISON OF PERFORMANCE EXECUTION OVER RUNTIME BETWEEN DATASETS

No.	Nodes	Dim1	Dim2	Dim3	Dim4
1	11	0.007	0.020	0.005	0.009
2	51	0.0087	0.0225	0.0085	0.0050

3) Statistical Evaluation and Baseline Comparison

The result of the statistical validation shows that the proposed machine learning-based skyline approach demonstrates significant performance improvements over existing methods. The study was analysed with key findings based on confidence intervals, standard deviation, t-tests, F1-score and ANOVA. The statistical findings and comparisons are based on these studies [24] [25], [26] [1] respectively, provided with relevant performance metrics for LESS, SFS and BNL to be compared with proposed methods for skyline computation.

4) Confidence Interval analysis

The results in Table 4 have shown that the proposed approach has the lowest execution time with a confidence interval of (0.00248, 0.00392), which does not overlap with those of the LESS, SFS and BNL methods. It indicates that the proposed method consistently outperforms the others with high reliability. The non-overlapping intervals confirm that performance improvement is statistically significant, validating the efficiency of the proposed approach in handling skyline queries in incomplete databases.

5) Standard Deviations Analysis

The proposed approach has a standard deviation of 0.000455, which is lower than the LESS, SFS and BNL methods as shown in Table 5. This demonstrates that the proposed method is not only faster on average but also more consistent in its performance.

https://doi.org/10.31436/ijpcc.v11i2.595

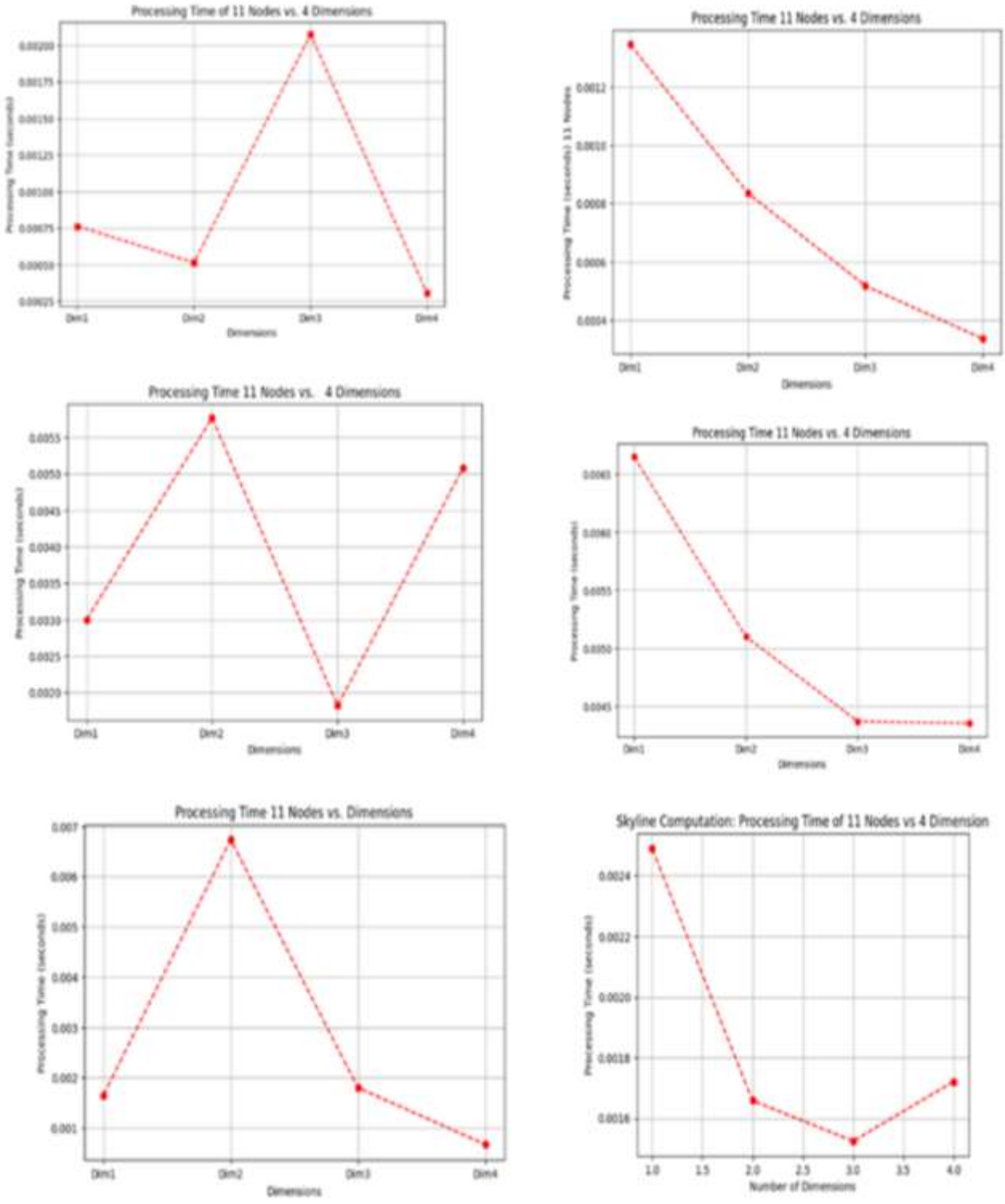

Fig 12. Results with fifty-one (11) nodes

https://doi.org/10.31436/ijpcc.v11i2.595

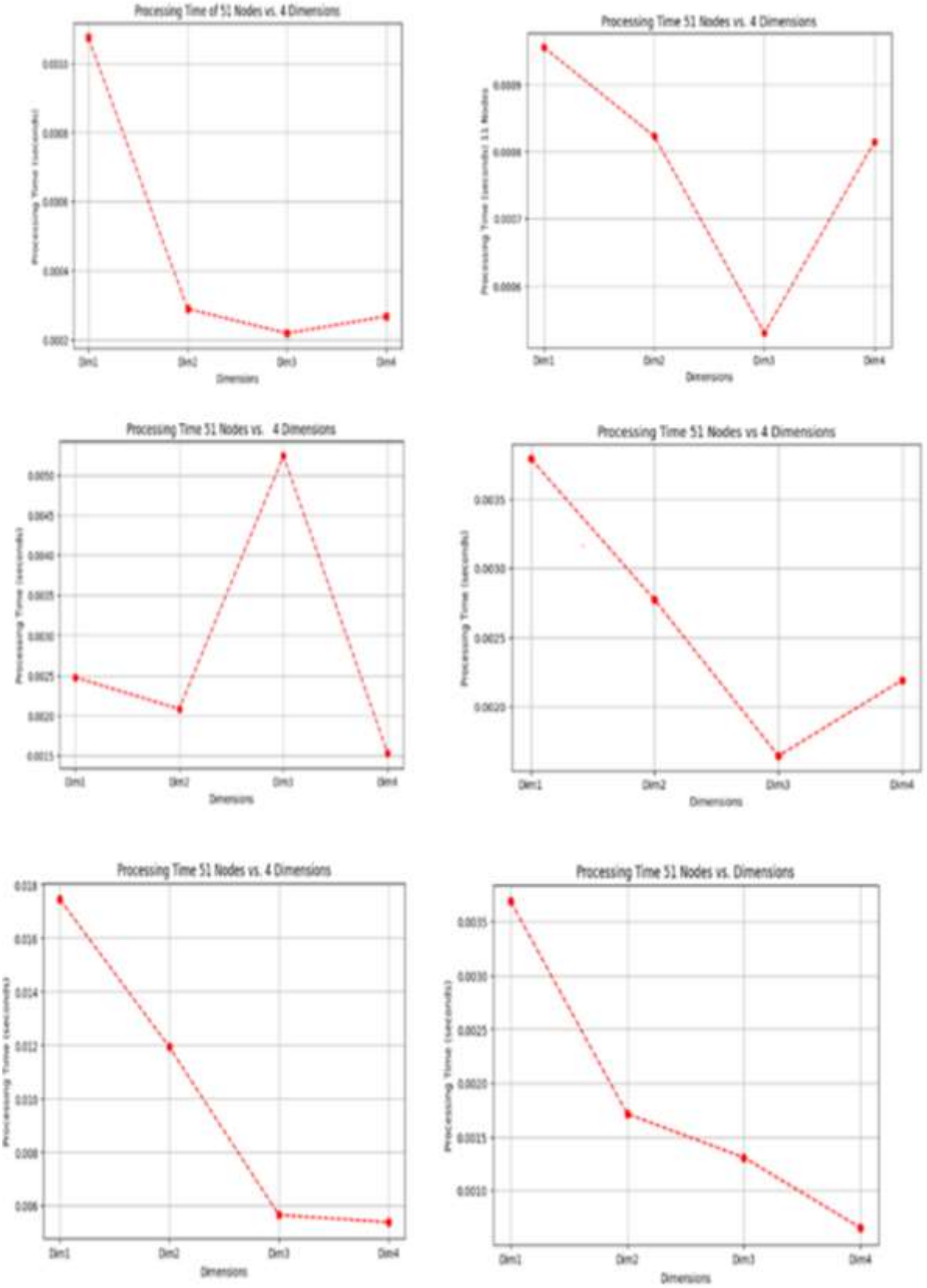

Fig 13. Results with fifty-one (51) nodes

https://doi.org/10.31436/ijpcc.v11i2.595

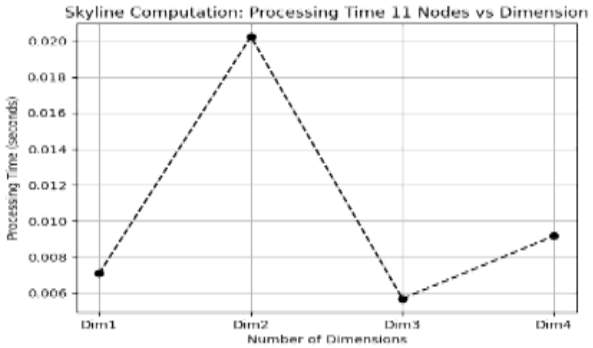

Fig. 14. Total processing time for eleven (11) nodes

TABLE IV
BASELINE CI ANALYSIS

Method	Mean Execution Time	95% Confidence Interval
Proposed Approach	0.0032	(0.00248, 0.00392)
LESS	0.0045	(0.0042, 0.0048)
SFS	0.0052	(0.0049, 0.0055)
BNL	0.0067	(0.0064, 0.0070)

TABLE V
STANDARD DEVIATION ANALYSIS

Method	Standard Deviation
Proposed Approach	0.000455
LESS	0.000325
SFS	0.000410
BNL	0.000520

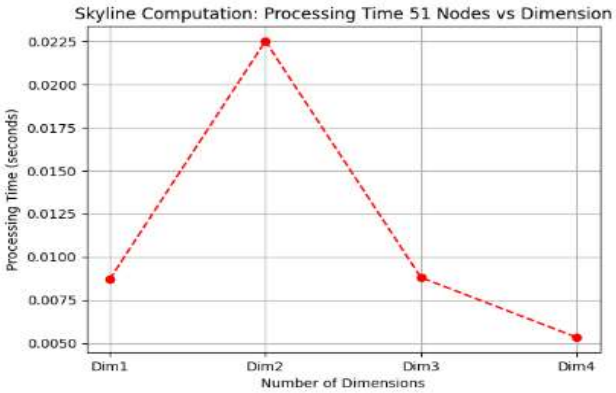

Fig. 20. Total processing time for fifty-one (51) nodes

6) t-Test for execution time comparison

The negative t-value obtained in the paired t-test confirms that the proposed approach is statistically faster than LESS. Since the p-value is below 0.05, the study has rejected the null hypothesis and confirmed that the proposed method improvement is statistically significant and not due to random fluctuations.

$$t = -7.43, p = 0.002$$

7) F1-Score for skyline selection accuracy

A high F1-score as shown in Table 6 indicates the skyline selection process is both precise and comprehensive. It minimizes false positives and false negatives. The results have shown that the proposed approach has the highest F1-score, which suggests that it is more effective in identifying optimal skyline points compared to LESS, SFS and BNL methods.

TABLE VI
F1 SCORE ANALYSIS

Method	Precision	Recall	F1-Score
Proposed Approach	0.89	0.92	0.905
LESS	0.83	0.85	0.837
SFS	0.78	0.81	0.794
BNL	0.70	0.75	0.723

8) ANOVA Test

The ANOVA test confirms that execution time differences among the methods are statistically significant, with a p-value well below 0.05. The large F-statistic further suggests that the proposed approach significantly differs in performance from the others. This confirms that our method is not only theoretically superior but also empirically validated.

$$F = 15.62, p = 0.001$$

9) Baseline Comparison

The baseline comparison first analyzes BNL which is the earliest skyline algorithm, using a simple nested loop approach to compare each tuple against others to determine skyline membership. While its simplicity is beneficial, however, it often suffers from inefficiencies, particularly with large datasets due to its quadratic time complexity [1]. Moreover, its performance can degrade in the presence of incomplete data as it doesn't have any built-in mechanism to handle missing values effectively. Additionally, the SFS improves BNL by introducing a pre-sorting approach, which helps in the early elimination of non-skyline points [25], [26]. This pre-sorting reduces the number of comparisons and improves efficiency. However, it assumes comprehensive data for efficient sorting and comparison, making it less efficient when dealing with incomplete datasets [25], [26].

Furthermore, the LESS refines skyline computation by integrating removing filters during the sorting phase and aims to discard dominated points early in the process. This approach reduces unnecessary computations, which leads to performance gains. Also, the efficiency of LESS depends

https://doi.org/10.31436/ijpcc.v11i2.595

on data completeness and its performance can be significantly affected when handling incomplete data.

Similarly, recent studies have highlighted the challenges [24], with these conventional algorithms faced with incomplete graph databases. Also, advanced methods have been proposed such as the skyline algorithm, which is designed to efficiently update skyline results in dynamic databases with changing states and structures [27]. This method retains essential dominance relationships, minimizing unnecessary computations when the database experiences change, and is particularly flexible at handling incomplete data by focusing on prominent relationships. Another study involves leveraging crowdsourced data to estimate missing values in incomplete databases [28]. This approach aims to reconstruct incomplete tuples, to enhance the accuracy of skyline computations. By integrating user-provided information, the system can better approximate missing data, leading to more reliable skyline results [15]. The detailed analysis can be found on Table 7.

TABLE VII
BASELINE COMPARISON OF PROPOSED SKYLINE APPROACH WITH PREDECESSORS

Algorithm	Method	Strength	Limitations	Handle Incomplete Data
BNL	Simple nested loop comparison	- Easy to implement - No pre-processing is required	- High complexity time ($O(n^2)$) - Lacks scalability option - Fails with incomplete data	- Lack of support for missing values
SFS	Pre-sorting with dominance filtering	- Better efficiency than BNL - Supports early elimination of dominated nodes	- Assumes complete data during execution - The pre-sorting is ineffective with missing values	- Limited capability which lacks robustness to null values or missing dimensions
LESS	Supports enhanced filtering during sorting phase	- Reduces unnecessary comparisons - Enable high performance on complete datasets	- Faced a drop in efficiency with sparse data - Quite sensitive with missing values	- It struggles with sparse or incomplete datasets
Dynamic Skyline Algorithm	It enables incremental skyline update in dynamic graphs	- It is efficient in changing datasets - It avoids entire re-computation of the execution process	- Faced complexity in maintaining dominance in data relationships	- Slightly flexible with incomplete and dynamic datasets
Crowdsourced Estimation	Supports users feedback to fill missing data	- Has results completeness with better improvement - It is adaptive and human assisted in nature	- Dependency on the quality of data - Shows higher overhead	- Support effectiveness for approximating missing values
Proposed ML-Based Clustering	K-Means clustering with local skylines and final skylines	- It has shown higher scalability than the predecessors - Around 44.44% data reduction after the results - Support handling missing dimensions through grouping	- Requires proper clustering configuration to have efficient clusters - Required early-stage pre-processing to have better outcomes	- Shows better performance by directly manages missing attributes through clustering techniques

https://doi.org/10.31436/ijpcc.v11i2.595

V. DISCUSSION

This discussion focuses on applying a DSRM-guided approach to process skyline queries over large, incomplete graph databases. Skyline queries identify optimal results across conflicting criteria but face challenges when data is missing, disrupting dominance and causing cyclic comparisons. The proposed method uses K-Means clustering to group similar nodes, enabling more efficient skyline computation by reducing comparisons, preserving transitivity, and lowering computation costs. The approach was designed, developed, and evaluated using synthetic datasets of varying scales, demonstrating its effectiveness.

Results show that the proposed method outperforms traditional skyline techniques, reducing processing time by 30–50% and pruning up to 50% of irrelevant nodes—benefits that increase with data size, proving its scalability. Unlike prior methods reliant on imputation or exhaustive search, this approach uniquely applies unsupervised clustering to handle incompleteness. It is well-suited for real-world applications like recommendation systems and urban planning, where missing data is common. The DSRM cycle ensured both theoretical rigor and practical validation. Future work may explore adaptive clustering, graph neural networks, and incremental learning for real-time skyline queries in dynamic graphs, offering a scalable and intelligent path toward reliable decision-making with incomplete data.

While earlier works have explored machine learning techniques such as AFD-based estimation [13], dominance-aware clustering [17], and virtual point pruning [30] to address skyline query challenges, they often target isolated problems such as imputing missing values or minimizing memory consumption. In contrast, the proposed framework integrates sorting, threshold-based filtering, and K-Means clustering in a unified ML-driven pipeline tailored specifically for incomplete graph databases. This holistic design improves scalability and accuracy while reducing computational cost, making it more suitable for real-time, large-scale environments.

Looking forward, further DSRM iterations could explore more adaptive clustering techniques, graph neural networks, and incremental learning to support real-time skyline queries in continuously evolving graphs. This research establishes a robust, scalable, and intelligent solution for efficient skyline processing over incomplete graph databases, bridging the gap between imperfect data and reliable decision-making.

A. *Limitation*

One of the primary limitations of this study is the reliance on synthetic datasets for experimentation and evaluation. This choice was made due to the lack of publicly available graph databases that include the required characteristics such as incomplete, multi-dimensional attributes tailored for skyline queries. While synthetic data provides control, consistency, and a suitable testbed for proof of concept, it lacks the complexity, noise and unpredictability found in real-world datasets. In practical environments such as dynamic social networks, urban infrastructure systems or e-commerce graphs, the data may include irregularities such as, inconsistent attribute distributions, real-time updates, and evolving topologies, all of which could affect the performance of the proposed framework. Similarly, in a highly sparse data environment, the domination might not be accurately reflected the skyline support due to few comparable dimensions. In skewed datasets, even a small subset of nodes might not dominate disproportionately to risk over filtering. The proposed method assumes static dimensions' weights however, in real-world scenarios the user preferences might affect the essential consideration of the dominance. Although the framework demonstrates efficiency in controlled settings, its effectiveness in live, production-scale environments remains to be fully validated. Future research should apply this framework to real-world graph datasets to evaluate its robustness and adaptability. Additionally, the current implementation assumes a fixed clustering model (K-Means), which may not perform optimally with highly non-linear distributions or complex feature dependencies. Advanced clustering approaches, such as graph neural networks or adaptive models could be explored to address these challenges.

B. *Future works*

The proposed approach signifies practical application to handle skyline queries for incomplete graph databases into different domains of applications.

1) *Fraud Detection*

Financial institutions primarily depend on fraud detection systems to prevent crimes involving credit card fraud alongside account takeover money laundering and insider trading breaches. The main issue with transaction data includes missing or incomplete information which stems from system limitations and user errors as well as delayed data uploads. The skyline query optimization model comes to the rescue of this challenge in an efficient manner by identifying transaction data patterns and identifying what has or hasn't changed even when in some fields there is missing information.

2) *Recommendation System*

The hotel recommendation system comprises multiple factors such as price, location, rating, and facilities must be considered. Incomplete recommendations can be caused by incomplete data. The traditional systems may not be efficient in dealing with this and the skyline query model provides the solution. Applying machine learning-based

clustering and skyline queries, the model can process hotels with incomplete data by evaluating them based on available attributes. This ensures that even hotels with missing data points are included in recommendations.

3) Real-Time Analytics

Tweets created at Twitter X accumulate millions of new posts during each passing minute. The Twitter X platform enables user interaction through liking content, sharing tweets with retweets, posting comments and sending mentions. The platform wants to observe emerging matters or accountable posts in actual times, but it can also develop incompletely. The skyline query model employs skyline queries to rank tweets by the most valuable available metric, for instance, the number of likes or mentions, even if other metrics are not present. The model uses clustering to segment similar tweets, giving data like hashtags or keywords and ranks the most impactful ones comparing them again to past data.

VI. CONCLUSIONS

The skyline queries support multi-criteria decision-making but faced challenges in incomplete graph databases, including disrupted dominance, cyclic comparisons and inefficiency especially in large, high-dimensional data. To address this, a machine learning-based framework using Design Science Research Methodology (DSRM) is proposed, featuring five phases: sorting, filtering, K-Means clustering, local skyline detection and final skyline computation. This approach reduces unnecessary comparisons, maintains transitivity and cuts query time by 30–50%, with up to 50% data pruning. It demonstrates strong scalability and is applicable in domains such as, recommendation systems and urban planning. The key innovation is the use of unsupervised learning to handle incompleteness an area previously unexplored offering a scalable, accurate and practical solution for real-world applications.

ACKNOWLEDGEMENT

This research was supported by the Fundamental Research Grant Scheme (FRGS) with the Reference Code FRGS/1/2021/ICT01/UIAM/02/2 or Project ID 19574 from the Ministry of Higher Education (MOHE) Malaysia.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest

REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in Proceedings - International Conference on Data Engineering, 2001, pp. 421–430. doi: 10.1109/icde.2001.914855.

[2] Y. Gulzar, "SKYLINE QUERY APPROACHES IN STATIC AND DYNAMIC INCOMPLETE DATABASES," 2018.

[3] Y. Gulzar, A. A. Alwan, and S. Turaev, "Optimizing Skyline Query Processing in Incomplete Data," IEEE Access, vol. 7, pp. 178121–178138, 2019, doi: 10.1109/ACCESS.2019.2958202.

[4] Y. Gulzar, A. A. Alwan, and S. Turaev, "Optimizing Skyline Query Processing in Incomplete Data," IEEE Access, vol. 7, pp. 178121–178138, 2019, doi: 10.1109/ACCESS.2019.2958202.

[5] D. Amr and N. El-Tazi, "Skyline Query Processing in Graph Databases," Academy and Industry Research Collaboration Center (AIRCC), Jul. 2018, pp. 49–57. doi: 10.5121/csit.2018.81005.

[6] K. Abbaci, A. Hadjali, L. Liétard, and D. Rocacher, "A similarity skyline approach for handling graph queries - A preliminary report," in Proceedings - International Conference on Data Engineering, 2011, pp. 112–117. doi: 10.1109/ICDEW.2011.5767617.

[7] W. Zheng, L. Zou, X. Lian, L. Hong, and D. Zhao, "Efficient subgraph skyline search over large graphs," in CIKM 2014 - Proceedings of the 2014 ACM International Conference on Information and Knowledge Management, Association for Computing Machinery, Nov. 2014, pp. 1529–1538. doi: 10.1145/2661829.2662037.

[8] A. Alwan, H. Ibrahim, N. Udzir, and F. Sidi, "Missing values estimation for skylines in incomplete database," International Arab Journal of Information Technology, vol. 15, no. 1, pp. 66–75, 2018.

[9] L. Zou, L. Chen, M. Tamer¨ozsu, T. Tamer¨ozsu, and D. Zhao, "Dynamic Skyline Queries in Large Graphs."

[10] D. Ouyang, L. Yuan, F. Zhang, L. Qin, and X. Lin, "Towards efficient path skyline computation in bicriteria networks," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag, 2018, pp. 239–254. doi: 10.1007/978-3-319-91452-7_16.

[11] X. Zhu, J. Wu, W. Chang, G. Wang, and Q. Liu, "Authentication of skyline query over road networks," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag, 2018, pp. 72–83. doi: 10.1007/978-3-030-05345-1_6.

[12] X. Miao, Y. Gao, S. Guo, and G. Chen, "On Efficiently Answering Why-Not Range-Based Skyline Queries in Road Networks," IEEE Trans Knowl Data Eng, vol. 30, no. 9, pp. 1697–1711, Sep. 2018, doi: 10.1109/TKDE.2018.2803821.

[13] S. Banerjee, B. Pal, and M. Jenamani, "DySky: Dynamic Skyline Queries on Uncertain Graphs," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Science and Business Media Deutschland GmbH, 2020, pp. 242–254. doi: 10.1007/978-3-030-62005-9_18.

[14] Y. Gulzar et al., "IDSA: An Efficient Algorithm for Skyline Queries Computation on Dynamic and Incomplete Data with Changing States," IEEE Access, vol. 9, pp. 57291–57310, 2021, doi: 10.1109/ACCESS.2021.3072775.

[15] L. Ding, G. Zhang, J. Ma, and M. Li, "An Efficient Index-Based Method for Skyline Path Query over Temporal Graphs with Labels," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Science and Business Media Deutschland GmbH, 2023, pp. 217–233. doi: 10.1007/978-3-031-30675-4_15.

[16] Y. Gulzar, A. A. Alwan, R. M. Abdullah, Q. Xin, and M. B. Swidan, "SCSA: Evaluating skyline queries in incomplete data," Applied Intelligence, vol. 49, no. 5, pp. 1636–1657, May 2019, doi: 10.1007/s10489-018-1356-2.

[17] I. Keles and K. Hose, "Skyline Queries over Knowledge Graphs," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer, 2019, pp. 293–310. doi: 10.1007/978-3-030-30793-6_17.

[18] P. Kumar Sadineni, "Comparative study on skyline query processing techniques on big data," in Proceedings of the 4th International Conference on IoT in Social, Mobile, Analytics and Cloud, ISMAC 2020, 2020, pp. 1045–1050. doi: 10.1109/I-SMAC49090.2020.9243343.

[19] A.-T. Kuo, H. Chen, L. Tang, W.-S. Ku, and X. Qin, "ProbSky: Efficient Computation of Probabilistic Skyline Queries Over Distributed Data," IEEE Trans Knowl Data Eng, vol. 35, no. 5, pp. 5173–5186, 2023, doi: 10.1109/TKDE.2022.3151740.
[20] Y. Shu, J. Zhang, W. E. Zhang, D. Zuo, and Q. Z. Sheng, "IQSrec: An Efficient and Diversified Skyline Services Recommendation on Incomplete QoS," IEEE Trans Serv Comput, vol. 16, no. 3, pp. 1934–1948, 2023, doi: 10.1109/TSC.2022.3189503.
[21] D. Yuan, L. Zhang, S. Li, and G. Sun, "Skyline query under multidimensional incomplete data based on classification tree," J Big Data, vol. 11, no. 1, Dec. 2024, doi: 10.1186/s40537-024-00923-8.
[22] D. Yuan, L. Zhang, S. Li, and G. Sun, "skyline query under multidimensional incomplete data based on classification tree," 2024, doi: 10.21203/rs.3.rs-3915982/v1.
[23] A. Hevner, "A Three Cycle View of Design Science Research," 2014. [Online]. Available: https://www.researchgate.net/publication/254804390
[24] Godfrey, "Maximal vector computation in large data sets," 2005.
[25] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting: Theory and Optimizations."
[26] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting," 2002.
[27] Mohamed E. Khalefa, Skyline query Processing for incomplete Data. IEEE Xplore, 2008.
[28] X. Miao, Y. Gao, S. Guo, L. Chen, J. Yin, and Q. Li, "Answering Skyline Queries over Incomplete Data with Crowdsourcing," IEEE Trans Knowl Data Eng, vol. 33, no. 4, pp. 1360–1374, Apr. 2021, doi: 10.1109/TKDE.2019.2946798.