Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Optimization of Dominance Testing in Skyline Queries Using Decision Trees

Jong-Hyeok Choi[1], Fei Hao[2], Yoo-Sung Kim[3] and Aziz Nasridinov[1,4]

[1]Bigdata Research Institute, Chungbuk National University, Cheongju 28644, South Korea
[2]Department of Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter EX4 4QF, UK
[3]Department of Information and Communication Engineering, Inha University, Incheon 22212, Korea
[4]Department of Computer Science, Chungbuk National University, Cheongju 28644, South Korea

Corresponding authors: Yoo-Sung Kim (email: yskim@inha.ac.kr) and Aziz Nasridinov (e-mail: aziz@chungbuk.ac.kr).

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00203, Development of 5G-based Predictive Visual Security Technology for Preemptive Threat Response). This work was also supported by the Industrial Strategic Technology Development Program (No. 200003991, Development of Korean Wave Convergence Service for AI-based Motion Evaluation and Learning Technology), funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea).

ABSTRACT Skyline queries identify skyline points, the minimal set of data points that dominate all other data points in a large dataset. The main challenge with skyline queries is executing the skyline query in the shortest possible time. To address and solve skyline query performance issues, we propose a decision tree-based method known as the *decision tree-based comparator* (*DC*). This method minimizes unnecessary dominance tests (i.e., pairwise comparisons) by constructing a decision tree based on the dominance testing. *DC* uses dominance relations that can be obtained from the decision rules of the decision tree to determine incomparability between data points. *DC* can also be easily applied to improve the performance of various existing skyline query methods. After describing the theoretical background of *DC* and applying it to existing skyline queries, we present the results of various experiments showing that *DC* can improve skyline query performance by up to 23.15 times.

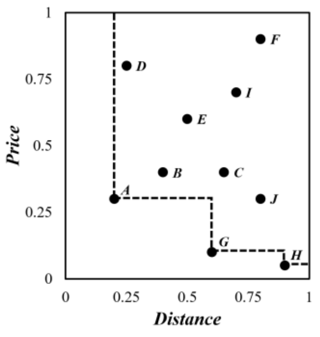
INDEX TERMS Database, decision tree, incomparability, query processing, skyline query.

I. INTRODUCTION

A skyline [1] refers to a minimal set of data points that dominate all other data points in a dataset. Dominance implies that the skyline points have the same values, or at least one better value, for all attributes than the remaining data points. Fig. 1 demonstrates an example of a skyline query in a database. Fig. 1 (a) lists a given dataset, and Fig. 1 (b) illustrates the skylines of the dataset (i.e., *A*, *G*, *H*).

Hotel	*Distance*	*Price*
A	0.2	0.3
B	0.4	0.4
C	0.65	0.4
D	0.25	0.8
E	0.5	0.6
F	0.8	0.9
G	0.6	0.1
H	0.9	0.05
I	0.7	0.7
J	0.8	0.3

(a) example dataset



(b) skyline query result

FIGURE 1. An example of skyline query.

There has been substantial research interest in developing efficient skyline query techniques to discover skylines, and various studies have been conducted to apply these methods in various fields, including retail [2], load networks [3], networks [4]–[6], web services [7], [8], and mobile edge computing [9]. In recent years, the skyline query technique has also been employed to compress a convolutional neural network (CNN) for deep learning [10].

The main challenge in such endeavors is to execute the skyline query in the shortest possible time. This is necessary to satisfy the time constraints imposed by the underlying user requests. Since a skyline query spends most of its time conducting dominance tests, performing pairwise data comparisons to determine dominance, and reducing the number of dominance tests, can result in a direct improvement in query performance [1]. Previous skyline query methods proposed for this purpose have ranged from sort-based methods [11]–[14] to those using indexing techniques [15]–[17], dominance relations [18]–[21], and parallel and distributed processing environments [22]–[28].

In this paper, using a concept called incomparability, we focus on dominance relations methods to minimize unnecessary operations between data points, and reduce dominance tests. Generally, incomparability occurs when two data points cannot dominate each other [16], [20]. For example, in Fig. 1, data points *A* and *G* are incomparable, and thus, cannot dominate each other. Existing dominance relation-based skyline query methods manage the skyline points based on a tree structure. However, these methods perform dominance tests using a tree structure, and this makes it challenging to remove multiple data points from a skyline query using a single dominance test procedure, as is the case in point-to-group or group-to-group comparisons [17] used in index-based methods. In addition, it is difficult to apply the dominance relation concept to other methods because of the structural dependencies of existing dominance relation-based methods.

In this study, we address the above issues by proposing a new decision tree-based method, known as the *decision tree-based comparator* (*DC*). The proposed method minimizes the unnecessary dominance tests by using dominance relations obtained from the decision rules of the proposed tree structure, to determine incomparability. By reducing the number of dominance tests in which dominance does not occur, we can significantly reduce skyline query time. Furthermore, unlike existing approaches, the proposed method can be easily applied to improve the performance of various existing skyline query methods, because of its unique tree structure. These are the specific contributions of this study.

- We propose a decision tree structure that can easily infer the dominance relation between data points. Specifically, the proposed decision tree structure contains decision rules that make it easy to classify the leaf nodes that have a dominance relation with the current data point by inferring the dominance relation between the leaf nodes.
- We propose a method known as *DC* that minimizes the number of dominance tests in the skyline query. This method eliminates the leaf nodes that exhibit incomparability with the current data point, thereby avoiding unnecessary dominance tests. In addition, we describe a method to further reduce the dominance tests by iterating the comparable leaf nodes using a concept called *population*, which is the number of skyline points belonging to a leaf node.
- We demonstrate how *DC* can be applied to existing skyline query methods. Applications of *DC* to Sort Filter Skyline (SFS) [11], [12], Sort and Limit Skyline Algorithm (SaLSa) [14], and Branch and Bound Skyline (BBS) [15], which are widely used skyline query methods, are explained at an algorithmic level.
- We evaluated the *DC* method experimentally to investigate its performance. We first measured and compared the number of dominance tests required to complete a skyline search, without *DC* and when *DC* was applied to an existing skyline query. We also measured and compared the time required to complete a skyline search with an existing skyline query with and without *DC*. The results of these experiments showed that applying *DC* reduced the skyline query time and dominance test calls of existing methods by up to 95.9% and 95.5%, respectively.

The remainder of this paper is organized as follows. Section 2 explains the problems of skyline queries and discusses state-of-the-art skyline query methods. Section 3 presents the proposed method. Section 4 demonstrates the application of *DC* to existing skyline query methods. The experimental results are outlined in Section 5. Section 6 summarizes the paper and highlights future work.

II. RELATED WORK

In this section, we describe related studies. Numerous methods have been proposed to process skyline queries efficiently. We classify these methods as sort-based, index-based, and dominance relation-based skyline query methods.

A. SORT-BASED SKYLINE QUERIES

Early studies related to skyline queries focused on searching skyline points using na ïve methods. For example, Borzsony *et al.* [1] proposed the Block-Nested Loop (BNL) and divide-and-conquer methods to perform skyline queries. These are na ïve methods that scan through a dataset and run a dominance test for each data point. However, they cannot search the skyline points monotonically, which results in many unnecessary dominance tests. Since then, several extension methods have been proposed that use sorting techniques to solve the BNL problem. Chomicki *et al.* [11], [12] introduced the SFS method. The SFS method first presorts the data points according to their entropy scores using a monotone scoring function, and then performs point-to-point comparisons, similarly to BNL. As the data points with lower entropy scores are more likely to become skyline points, the SFS prunes a considerable amount of data points during the early stage of pairwise data comparisons. Similar methods that use presorting with early data pruning strategies have been proposed in [13] and [14]. Linear Elimination Sort for Skyline (LESS) [13] contains an elimination-filter window to determine the set of data points that are most likely to dominate other data points during the sorting process. Subsequently, a dominance test is conducted with those data first to identify the data points to be dominated with small comparisons using the skyline-filter window.

However, as SFS and LESS still have to utilize all the data for the dominance test, SaLSa [14] provides a means of terminating the skyline query without using all of the data points. To this end, a new monotone limiting function known as minC was proposed in [14]. Based on these functions, the concept of a stop point was proposed, which can confirm that all unread data points will be dominated. However, it has not been widely used because of a problem with the efficiency of the monotone limiting function, which decreases as the dimensions increase. Also, recently, a method for sorting the incomplete data was proposed in [29] and has been used in many skyline queries in index-based or distributed environments [30].

These sort-based skyline query methods can effectively reduce dominance testing by enabling the skyline to exhibit monotonic properties. However, the problem of high cost computations, such as presorting or point-to-point comparisons, remains to be solved.

B. INDEX-BASED SKYLINE QUERIES

Index-based skyline query methods can remove multiple data points with a single dominance test by using an index structure. Among the early index-based methods the most representative approach is the BBS [15]. The BBS uses an R-tree-based indexing structure and performs point-to-group comparisons of the skyline points and a minimum bounding rectangle (MBR). This comparison uses the properties of the data space partitioning in the R-tree. If a specific skyline point dominates the lower-left corner point of a specific MBR, all of the data points belonging to the MBR are dominated by the same skyline point. Using these features, the BBS can reduce the number of dominant tests because multiple data points can be removed from the query process with a single comparison.

To address the problem where the computation required to construct an R-tree increases significantly as the dataset dimensions increase, Z-SKY was proposed by Lee *et al.* [16]. Z-SKY searches the skyline through a ZBtree by combining a B^+-tree and a Z-order curve. The ZBtree divides the Z-order curve into segments known as RZ-regions which can be managed according to certain criteria. Z-SKY enables group-to-group comparisons using a dominance test between these RZ-regions. However, with the specialization of integer datasets, the high computational cost of the ZBtree offset the benefits gained by reducing the number of dominance tests.

More recently, studies are being conducted to search the skyline by applying an index structure to incomplete data [30]-[32] or search the skyline in a Hadoop or GPU environment by applying an index [26]-[28].

The index-based skyline query methods are effective for skyline queries because they can remove multiple data points with a single dominance test. However, depending on the properties of the index technique used for the skyline query, it may be necessary to solve problems such as availability for only a specific data type, or the cost of the index structure will outweigh its advantages.

C. DOMINANCE RELATION-BASED SKYLINE QUERIES

Dominance relation-based skyline query methods reduce the number of dominance tests so that the skyline can be managed using lattice or tree structures, and avoid dominance tests, based on the expected incomparability through these structures. In the case of the Lattice Skyline [18], a lattice structure is created using low-cardinality attributes and the dominance relations between them, and this structure is used to identify the incomparability. However, a limitation exists, because the dominance relation can only be used when low-cardinality attributes exist.

For Object-based Space Partitioning Skyline [19], BSkyTree [20], and BJR-tree [21], the data points are partitioned into regions of the multi-dimensional data space by using a dominance relation that can be identified through point-to-point comparisons. Thereafter, the dominance relation between partitioned regions is constructed into a tree structure to easily determine the incomparability. When a new data point is input, these trees minimize the dominance testing by determining the region to which the input data point belongs, and comparable regions through the tree. The main advantage of these methods is that they can process the dominance tests using only the data points belonging to those regions.

These dominance relation-based skyline query methods succeed in reducing the number of dominance tests by effectively utilizing a dominance relation, which is obtained through the dominance test, and then avoiding dominance tests for incomparable cases. However, these skyline query methods can only be used for point-to-point comparisons. Moreover, it is difficult to apply the dominance relation concept to other methods, because of the structural dependencies of the existing dominance relation-based methods.

III. DECISION TREE-BASED COMPARATOR FOR SKYLINE QUERIES

The proposed *DC* is described in this section. The *DC* is a novel skyline query method that uses a decision tree structure to minimize the number of dominance tests. In addition, the *DC* can be easily applied to algorithms in conventional skyline query methods to improve their query performance. First in subsection A we describe the generation of the decision tree. We then outline the actual *DC* procedure in subsection B. The notations used in this paper are presented in Table I.

A. DECISION TREE FOR SKYLINE QUERIES

Recall from Section 1 that a data point that has a better value than the other data points in at least one dimension, while being equal to or better than the other data points in the remaining dimensions, is selected as a skyline point. The

TABLE I
SYMBOLS AND DEFINITIONS

Symbol	Definitions
N	Number of data points
d	Number of dimensions
TOP	Top-1 skyline point
TOP[i]	*i*-th dimension value of top-1 skyline point
CUR	Current input data point
CUR[i]	*i*-th dimension value of current input data point
NODE_IDX	Order of classified leaf node
BRANCH	Branch of decision tree
CUR_IDX	Order index of current leaf node
TGT_IDX	Order index of target leaf node
INCOM	Incomparability verification result
LAST	Maximum order of leaf nodes
SKYLF	Skyline windows conducted on leaf nodes
L_IDX	Leaf node order of input data point
population	Number of skyline points belonging to leaf node
Div	Total number of divisions in *population*

majority of state-of-the-art skyline query methods use monotonic functions, such as the entropy score [11]–[13] or mindist [15], [20] for effective skyline query processing. This is because, if a data point with numerous better values than the other data points is preferentially used in a skyline query through a monotonic function, the skyline can be searched with fewer dominance tests. That is, a data point with the lowest score according to the monotonic function has the highest probability of dominating the other data points, and all input data points must perform a dominance test with this data point.

Like the existing methods, we first sort the given data points according to a monotonic function. We then construct a decision tree for the sorted data points using a level-by-level approach, where each level in the tree corresponds to a dimension of the dataset. When new *d*-dimensional data is input to the decision tree, each data point is compared with a data point that has the lowest score according to the monotonic function (referred as the top-1 skyline point) in matching the level of the decision tree. If the input data point has a smaller value than, or an equal value to, the top-1 skyline point at the same level, it is classified as the left-side node; otherwise, it is classified as the right-side node.

Example 1. Suppose that a three-dimensional dataset related to a hotel reservation is provided, as per Table II. We first normalize the dimensions (i.e., distance, accommodation cost, and star rating) of the given dataset, and then, sort it based on the entropy score [11], [12]. A decision tree constructed based on this dataset is presented in Fig. 2, where each level in the tree corresponds to the dimensions of the dataset. Here, hotel *b* (distance: 0.6, cost: 0.5, rating: 0.25), which has the lowest entropy score in the dataset, is selected as the top-1 skyline point and is used to build a decision tree. Thus, when the rest of the data points are input to the decision tree, each data point is compared with hotel *b* when matching the level of the decision tree, and classified into the corresponding leaf node. For example, data points having a distance value less than or equal to 0.6 are classified into leaf nodes 0 to 3, while the remaining data points are classified into leaf nodes 4 to 7.

After classifying the input data points into leaf nodes, we need to identify skyline points using dominance tests. If a data point is not dominated by any other data point in the dataset, it is stored in the corresponding leaf node as a skyline point; otherwise, it is discarded. To minimize unnecessary dominance tests, we propose a set of classification rules, where the order of leaf nodes is expressed in bits. For example, if we express the order of leaf nodes in bits for the decision tree depicted in Figure 2, the front four nodes are represented by 000 (0), 001 (1), 010 (2), and 011 (3), while the following four nodes are represented by 100 (4), 101 (5), 110 (6), and 111 (7). At this point, regularity can be observed in the earliest bits: in the first dimension (marked in red), the data points classified to the left-side have bits starting with 0 (i.e., 000, 001, 010, and 011), while the other data points have bits starting with 1 (i.e., 100, 101, 110, and 111). This rule applies equally to the remaining dimensions. For example, in the second dimension (marked in green), the comparison is performed with 0.5, which is the second-dimension value of hotel *b*. The leaf nodes classified to the left-side have 0 as the second bit (i.e., 000, 001, 100, and 101), while the leaf nodes classified to the right-side have 1 as the second bit (i.e., 010, 011, 110, and 111). Similarly, in the third dimension (marked in blue), when the comparison is performed with 0.25, the leaf nodes classified to the left-side have 0 as the last bit (i.e., 000, 010, 100, and 110), while the leaf nodes classified to the right-side have 1 as the last bit (i.e., 001, 011, 101, and 111). According to the definition of dominance [1], it can be inferred that only the data points belonging to the leaf nodes with bits equal to or smaller than the current leaf node in all dimensions can dominate the data points of the current leaf node. Thus, by using the order of the leaf nodes represented as bits, we can verify incomparability in advance. The procedure of the proposed decision tree, *D-Classifier*, is described in Algorithm 1.

TABLE II
HOTEL DATASET SORTED BY ENTROPY SCORE [11], [12]

Hotel	Distance	Cost	Rating	Entropy Score
b	0.6	0.5	0.25	1.0986
e	0.45	0.6	0.4	1.1780
a	0.9	0.45	0.2	1.1957
i	0.55	0.9	0.15	1.2199
f	0.4	0.95	0.25	1.2274
k	0.7	0.7	0.2	1.2436
c	0.5	0.4	0.8	1.3297
g	0.95	0.95	0.05	1.3844
d	0.75	0.65	0.4	1.3969
h	0.95	0.45	0.85	1.6546

Example 2. Let us continue Example 1 and consider hotel *g* as an example. Hotel *g* is classified into leaf node 6 by

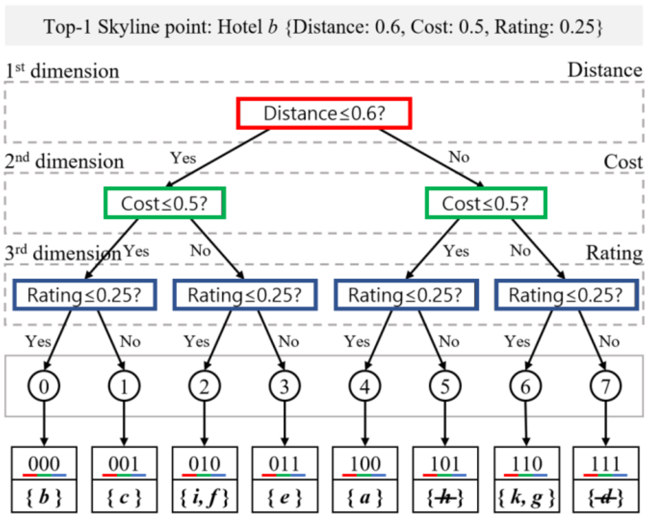


FIGURE 2. Classification of data points in Table II using a decision tree.

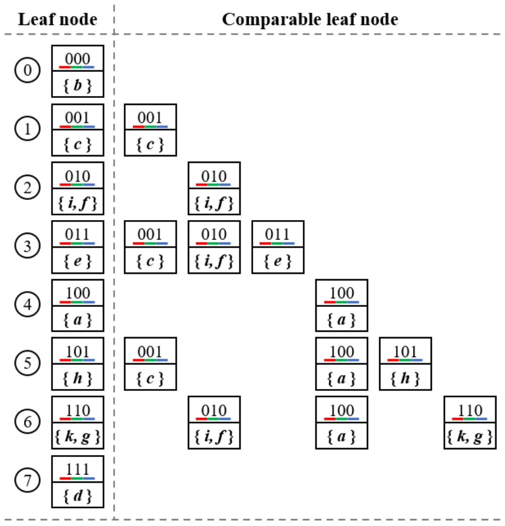


FIGURE 3. List of comparable leaf nodes for each leaf node of Fig. 2.

comparison with hotel *b*, which is the top-1 skyline point. Afterward, it is necessary to check whether it can become a skyline point through the dominance test. We can see from the bit value, 110, of leaf node 6 that the hotels of the corresponding leaf node always have a value less than or equal to 0.25 in the third dimension. With the bit value, it can be known in advance that hotels belonging to leaf nodes 1 (001), 3 (011), and 5 (101) with values greater than 0.25 in the third dimension cannot dominate hotel *g*. Consequently, hotels *c* and *e*, which cannot dominate hotel *g*, can be excluded from the dominance test sooner.

After the dominance test with data points selected as the skyline points in the previous step (i.e., hotels *i*, *f*, *a*, and *k*), we can see that hotel *g* is not dominated by these skyline points and thus, will be stored in leaf node 6 as a skyline point. On the other hand, hotel *d*, which is a subsequent input data point, is classified as leaf node 7. However, considering that hotel *d* has the worst values in all dimensions compared with the top-1 skyline point, it is discarded. Hotel *h*, which is the last input data point, is classified into leaf node 5. After a dominance test with the data points in comparable leaf nodes 0, 1 and 4, we can see that hotel *h* is dominated by hotel *c* in leaf node 1, and thus, is discarded.

Fig. 3 shows the leaf nodes that require dominance testing. Originally, all of the leaf nodes needed to be compared with leaf node 0. However, since the comparison was already completed by the classifying process with the decision tree, further comparison is not required. Similarly, the last leaf node 7 must be compared with all other leaf nodes. However, considering that leaf node 7 was already dominated by the top-1 skyline point belonging to leaf node 0, further comparison with the other leaf nodes is unnecessary. In addition, since it is possible to know by the bit value that all of the leaf nodes located behind the current leaf node have a larger value in at least one dimension, those unnecessary dominance tests can also be excluded.

As we saw in Examples 1 and 2, the data points of a leaf node with 1 in the first bit and 0 in all remaining bits cannot dominate the data points of all leaf nodes with 0 in the first bit. This is because all of the data points in the corresponding leaf node have a larger value in the first dimension than the data points of all leaf nodes with bits starting with 0. In contrast, the corresponding leaf node has the potential to dominate the data of all leaf nodes with the first bit of 1 because all the remaining bits are 0. For this reason, incomparability only occurs when a bit has a larger bit than the current leaf node in at least one dimension of the leaf nodes.

This incomparability can be easily verified by the bitwise OR operation. To this end, we propose *Incmp*, an incomparability verification method, which is described in Algorithm 2. As shown in line 2, performing a bitwise OR operation with a leaf node that has a larger bit in one or more dimensions returns a result that is greater than the current leaf node order, which enables us easily verify the incomparability.

From the *D-Classifier* and *Incmp* algorithms, we can observe that the bit value obtained from the order of the leaf nodes allows us to check incomparability in advance. Using the proposed algorithms, the total of the 2^{2d} dominance

Algorithm 1 *D-Classifier*

```
Input:    TOP: Top-1 skyline point
          CUR: Current input data point
          d: Number of dimensions
Output:   NODE_IDX: Index of classified leaf node
Begin
 1:  NODE_IDX = 0
 2:  for dim = 1 to d do
 3:      if CUR[dim] <= TOP[dim] then
 4:          BRANCH = 0
 5:      else
 6:          BRANCH = 1
         //Bitwise left shift to apply dimensional order
 7:      NODE_IDX = (NODE_IDX << 1) + BRANCH
 8:  return NODE_IDX
End
```

Algorithm 2 *Incmp*

```
Input:   CUR_IDX: Order index of current leaf node
         TGT_IDX: Order index of target leaf node
Output:  INCOM: Incomparability verification result
Begin
      //Bitwise OR between inputs
  1:  if (CUR_IDX | TGT_IDX) > CUR_IDX then
  2:      INCOM = True
  3:  else
  4:      INCOM = False
  5:  return INCOM
End
```

relation between leaf nodes, which was necessary when incomparability was not known, can be reduced to 3^d, meaning that only $(3/4)^d$ of the dominance relation is needed to check incomparability. Furthermore, considering that leaf node 0 and the last leaf node in the *DC* do not require dominance testing, as explained in Fig. 3, the dominance relation between the leaf nodes is further reduced from $(2^d - 2)^2$ to $3^d - 2^{d+1} + 1$. This indicates that the unnecessary dominance tests, where dominance does not occur, are significantly reduced.

B. DECISION TREE-BASED COMPARATOR

Recall from subsection A that the decision tree is a classification method that can be used to minimize unnecessary dominance tests by identifying incomparability when no dominance occurs between data points. In the *DC*, the leaf node to which the current input data point belongs is classified using the proposed *D-Classifier* algorithm. Subsequently, using the proposed *Incmp* algorithm, the dominance test is performed by limiting the data points of the leaf node where no incomparability occurs. If an input data point is dominated by another data point during this process, the dominance test for the corresponding input is immediately terminated. Conversely, if an input data point is not dominated by any other data points, it is stored in the leaf node obtained from the *D-Classifier*. In other words, the data points stored in the leaf nodes consist of skyline points that are not dominated by other data points.

However, in this process, moving to the next leaf node after a comparison with all skyline points belonging to the comparable leaf node creates the following problems. Firstly, when a monotonic function is used, there is a higher probability that a data point which is determined early to be a skyline point will dominate the other data points, compared with a skyline point that is determined later. This is because data points with superior values are preferentially used for the calculations in monotonic functions. However, when the dominance test is performed on a leaf node basis, the skyline points with high dominance probability cannot be preferentially used. Therefore, cases exist in which data points that could be dominated earlier are dominated later. Secondly, to solve such a problem, when using a skyline window composed of a single list as in conventional methods, it is necessary to perform *Incmp* for all skyline points until the input data point is dominated, which causes unnecessary

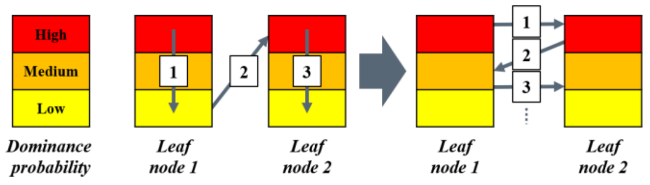


FIGURE 4. Example of the *DC*'s dominance test using a divide-and-conquer manner.

computation even in incomparable cases. This subsequently reduces query performance.

To solve these two contradictory problems in the *DC*, the dominance test is conducted in a divide-and-conquer manner based on the concept of *population*, which is the number of skyline points belonging to a leaf node. Fig. 4 demonstrates the dominance test procedure when it is performed using the proposed divide-and-conquer manner.

Let us assume that skyline points belonging to each leaf node are divided into 10 groups according to dominance probability, which can be determined using the entropy score. Afterwards, the input data point is first compared with the top 10% skyline points of all comparable leaf nodes in sequential order. Here, if the input data point is dominated by one of the top 10% skyline points in the leaf nodes, then it is immediately discarded. If the input data point is not dominated by any of the top 10% skyline points in any of the comparable leaf nodes, then it is compared with the skyline points corresponding to the next top 10% skyline points (i.e., 10%-20% of the skyline points) in all comparable leaf nodes. If the input data point is not dominated by any of the skyline point in the leaf nodes through this divide-and-conquer strategy, then it becomes a skyline point and is stored in the corresponding leaf node. When the skyline query is processed in this manner, those skyline points with a high dominance probability can be used preferentially in each comparable leaf node. This enables us to increase the probability of the input data points being dominated early.

The proposed divide-and-conquer strategy can minimize the required dominance tests in *DC*. However, note that when a fixed division value (*Div*) is used, regardless of the *population*, an unnecessary overhead may occur for the following reasons. First, as the skyline points increase, the number of skyline points that must be compared by dominance testing before moving to the next comparable leaf node may also increase. Conversely, if there are too few skyline points stored in each leaf node, the overhead caused by traversing through leaf nodes may increase too. Therefore, to avoid unnecessary overhead and keep the dominance test running efficiently even as the skyline increases, *Div* is dynamically increased according to the average number of skyline points belonging to leaf nodes.

The *DC* algorithm and its optimization variants are presented in Algorithms 3 to 5. The efficiency of these algorithms is demonstrated in Section 5.

Algorithm 3 presents *DC_Init*, which is a leaf node initialization function that is required for the *DC*. In this

Algorithm 3 *DC_Init*

```
Input:   d: Number of dimensions
         LAST: Maximum order of leaf nodes
Output:  SKYLF: Skyline windows conducted for leaf nodes
Begin
        //Leaf nodes and incomparability initialization
  1:    for CUR_IDX = 1 to LAST do
            //Until idx is the same as CUR_IDX
  2:        for idx = 1 to CUR_IDX do
  3:            if not Incmp(CUR_IDX, IDX) then
  4:                SKYLF[CUR_IDX].comparable(IDX)
  5:    return SKYLF
End
```

function, *Incmp* is used to identify comparable leaf nodes and to store this information. The essential parts of *DC_Init* are lines 1 and 2. The zeroth leaf node can only store the top-1 skyline point, and a comparison with the top-1 skyline point is initially conducted through the *D-Classifier*. For this reason, when searching for a comparable leaf node, it begins from the first, and not the zeroth, leaf node. Moreover, as the skyline points corresponding to the same leaf node are also comparable, the process is repeated until *idx* is equal to *CUR_IDX* to add itself as a comparable leaf node.

Algorithm 4 presents the actual *DC* algorithm. The *DC* behaves as an extension of the dominance test for easy application to other skyline query algorithms. The *p*-ratio in line 2 represents the ratio required to preferentially use the skyline points with a high dominance probability from each leaf node in the dominance test. Lines 3 to 4 are used to verify that *CUR* is dominated by the top-1 skyline point using the leaf node order obtained from the *D-Classifier*. If this is not the case, the comparison is repeated with the skyline points of the comparable leaf nodes in lines 6 to 17. In line 8, the order of leaf nodes is returned so that the comparable leaf nodes previously searched with *DC_Init* can be accessed sequentially. Lines 9 to 10 define the start and end locations of the skyline points to be compared in each comparable leaf node through *p_ratio* and *p_cur*. Thereafter, the actual dominance test is performed, as per line 12. If *CUR* is dominated by a specific skyline point, the value is returned as immediately dominated, according to lines 13 and 14. At line 15, since all the comparable leaf nodes have been cycled, *p_cur* is increased to access the next sequence of skyline points. Lines 16 to 17 check that all leaf node *populations* to be compared have been identified, and if so, the comparison ends. Subsequently, in line 18, the True or False stored in *Dominated* is finally returned.

Algorithm 4 *DC*

```
Input:   CUR: Current input data point
         L_IDX: Leaf node order of input data point
         LAST: Maximum order of leaf nodes
         Div: Total number of divisions in population
         SKYLF: Skyline windows conducted for leaf nodes
Output:  Dominated: Dominance result of input data point
Begin
  1:    Dominated = False
  2:    p_ratio = 1 / Div  //Population traversal ratio
  3:    if L_IDX > LAST then
  4:        Dominated = True  //CUR dominated by TOP
  5:    else
  6:        p_cur = 0
  7:        while not Dominated do
  8:            foreach IDX $\in$ SKYLF[L_IDX].comparable do
  9:                start = SKYLF[IDX].size * p_ratio * p_cur
 10:                end = SKYLF[IDX].size * p_ratio * (p_cur + 1)
 11:                for ptr = start to end do
 12:                    if SKYLF[IDX][ptr] dominate CUR then
 13:                        Dominated = True
 14:                        return Dominated
 15:            p_cur = p_cur + 1
 16:            if p_ratio * p_cur > 1 then
 17:                break
 18:    return Dominated
End
```

Algorithm 5 presents the basic structure of the overall skyline query that is required to search the skyline with *DC_Init* and *DC*, which we call *DC-basic*. Line 3 determines the order of the final leaf node. At this time, the actual final leaf node is the (2^d - 1) leaf node, but since this leaf node is dominated by the zeroth to which the top-1 skyline point belongs, the leaf node before the actual final leaf node is our final leaf node. Therefore, -2 is used here and not -1. The skyline is then searched in lines 5 to 19 using the data points that have been presorted with the monotonic function. Lines 7 to 12 use the first input data point to create skyline windows known as *SKYLF*, corresponding to the leaf nodes of the *DC*, and set the variables to store the data point as a top-1 skyline point. From the second data point, as per lines 14 to 15, the *D-Classifier* determines which leaf node *CUR* belongs to, and verifies whether it is dominant using the *DC*. At line 17, if *CUR* has not been dominated, *CUR* is stored as a skyline point on the corresponding leaf node via the *L_IDX* obtained in line 14. After that, the necessity of updating *Div* is checked, as in line 18, and *Div* is increased when the average number of skyline points in *SKYLF* exceeds a certain standard. Once the search for all data points has been completed, all of the skyline points stored in the leaf node *SKYLF* are confirmed as the skyline, and these are merged and returned, as indicated in lines 20 to 22.

Algorithm 5 *DC-basic*

```
Input:    DATA: Ordered dataset by monotonic function
          d: Number of dimensions
          N: Number of data points
Output:   SKYLINE: Set of skyline points by DC
Begin
 1:  SKYLINE = {}
 2:  SKYLF = NULL
 3:  LAST = $2^d – 2$  //Order of last possible leaf node
 4:  ptr = 0
 5:  while ptr < N do
 6:     CUR = DATA[ptr]
 7:     if SKYLF equal to NULL then
 8:        SKYLF = DC_Init(d, LAST)
 9:        Dominated = False
10:        TOP = CUR
11:        L_IDX = 0
12:        Div = 1
13:     else
14:        L_IDX = D-Classifier(TOP, CUR, d)
15:        Dominated = DC(CUR, L_IDX, LAST, Div, SKYLF)
16:     if not Dominated do
17:        SKYLF[L_IDX].add(CUR)
18:        Div = DivUpdateCheck(SKYLF)
19:     ptr = ptr + 1
20:  for IDX = 0 to LAST do
21:     SKYLINE = SKYLINE ∪ SKYLF[IDX]
22:  return SKYLINE
End
```

Algorithm 6 *SFS-DC*

```
Input:    DATA: Sorted dataset by entropy score at Heap
          d: Number of dimensions
Output:   S: Set of skyline points of DATA
Begin
 1:  S = {},  SKYLF = NULL, LAST = $2^d – 2$
 2:  unfinished = True
 3:  while (unfinished) do
 4:     T = open_cursor(DATA)
 5:     unfinished = False
 6:     while next_data(T, t) do
 7:        if SKYLF equal to NULL then
 8:           SKYLF = DC_Init(d, LAST)
 9:           Dominated = False
10:           TOP = t, L_IDX = 0, Div = 1
11:        else
12:           L_IDX = D-Classifier(TOP, t, d)
13:           Dominated = DC(t, L_IDX, LAST, Div, SKYLF)
14:        if not Dominated then
15:           if "SKYLF is full" then
16:              unfinished = True
17:              break
18:           else
19:              SKYLF[L_IDX].add(t)
20:              Div = DivUpdateCheck(SKYLF)
21:     if (unfinished) then
22:        S = open_new_file(SecondPass)
23:        write(S, t)
24:        while next_data(T, t) do
25:           L_IDX = D-Classifier(TOP, t, d)
26:           Dominated = DC(t, L_IDX, LAST, Div, SKYLF)
27:           if not Dominated then
28:              write(S, t)
29:  free(DATA)
30:  close(S)
31:  Heap = SecondPass
32:  for IDX = 0 to LAST do
33:     S = S ∪SKYLF[IDX]
34:  free(SKYLF)
35:  return S
End
```

IV. APPLICATIONS OF THE DECISION TREE-BASED COMPARATOR

In this section, we discuss how the proposed *DC*-related algorithms can be applied to the existing state-of-the-art skyline query methods, which do not use the incomparability concept. To this end, we demonstrate an application of the *DC* algorithms to the sort-based SFS [11], [12], SaLSa [14], and index-based BBS [15], which are representative skyline query methods that do not use the incomparability concept and their own skyline windows.

A. SFS-DC

The SFS [11], [12] uses monotonic functions and sorting to ensure that the skyline points are not dominated by the following sequence of input data points.

The SFS includes a procedure that stores skyline candidates in a separate file when the skyline window is full. However, in addition to this feature, the dataset can be processed in a manner very similar to that of *DC-basic*, and *SFS-DC* is achieved by applying *DC* to the existing SFS, as demonstrated in Algorithm 6. This algorithm shows that the incomparability concept can be easily applied because there is no structural change, other than changing the existing dominance test to be performed through the *D-Classifier* and *DC*.

B. SaLSa-DC

The SaLSa is a method that uses the concepts of a monotonic function and stop point together, thereby eliminating the need to access all data points by terminating the query early if a skyline point can no longer occur through the stop point. To achieve this, SaLSa performs checks relating to the stop point, but the skyline is determined by a dominance test between the skyline points and the current input data point. Therefore, in SaLSa, by replacing the logic related to dominance tests with *DC*-related algorithms, it is possible to use incomparability and easily improve query performance.

The specific SaLSa algorithm with *DC* is presented as Algorithm 7. In this algorithm, a processing procedure is required, corresponding to lines 8 to 10, which sets the first input data point as *TOP*, such as *SFS-DC*. But in the subsequent logic, most of the processing proceeds in the same manner as the existing SaLSa algorithm.

Algorithm 7 *SaLSa-DC*

```
Input:   DATA: Sorted dataset by minC
         d: Number of dimensions
Output:  S: Set of skyline points of DATA
Begin
  1:  S = {}, stop = False, p_stop = undefined, u = DATA
  2:  SKYLINE = {},  SKYLF = NULL, LAST = $2^d - 2$
  3:  while not stop and $u \neq \{\}$ do
  4:      p = u.next_data, u = u.remove(p)
  5:      if p_stop_plus $\leq$ minC(p) and p_stop $\neq$ p then
  6:          stop = True
  7:      else
  8:          if SKYLF equal to NULL then
  9:              SKYLF = DC_Init(d, LAST)
 10:              TOP = t, L_IDX = 0, Div = 1
 11:          else
 12:              L_IDX = D-Classifier(TOP, t, d)
 13:          if not DC(p, L_IDX, LAST, Div, SKYLF) then
 14:              SKYLF[L_IDX].add(p)
 15:              Div = DivUpdateCheck(SKYLF)
 16:              if p_plus < p_stop_plus then
 17:                  p_stop = p
 18:  for IDX = 0 to LAST do
 19:      S = S $\cup$ SKYLF[IDX]
 20:  return S
End
```

Algorithm 8 *BBS-DC*

```
Input:   R: R-tree of dataset
         d: Number of dimensions
Output:  S: Set of skyline points
Begin
  1:  S = {},  SKYLF = NULL, LAST = $2^d - 2$
  2:  insert all entries of root R into heap H
  3:  while not H.empty
  4:      e = H.pop()  //read and remove top entry of H
  5:      if SKYLF equal to NULL then
  6:          L_IDX = 0, Div = 1
  7:      else
  8:          L_IDX = D-Classifier(TOP, e, d)
  9:      if DC(e, L_IDX, LAST, Div, SKYLF) then
 10:          discard e
 11:      else  //e is not dominated
 12:          if e is an intermediate entry then
 13:              foreach child i of e do
 14:                  L_IDX = D-Classifier(TOP, i, d)
 15:                  if not DC(i, L_IDX, LAST, Div, SKYLF) then
 16:                      H.push(i)
 17:          else  //e is a data point
 18:              if SKYLF equal to NULL then
 19:                  SKYLF = DC_Init(d, LAST)
 20:                  TOP = e
 21:              SKYLF[L_IDX].add(e)
 22:              Div = DivUpdateCheck(SKYLF)
 23:  for IDX = 0 to LAST do
 24:      S = S $\cup$SKYLF[IDX]
 25:  return S
End
```

C. BBS-DC

BBS is representative of index-based skyline queries, and performs point-to-group comparisons using the MBR of the R-tree to remove multiple data points with a single dominance test. To accomplish this, the BBS performs a comparison by assuming the lower-left corner as the point for performing the dominance test with the data point and the MBR, which is a group of data points. Therefore, even with an MBR (i.e., an intermediate entry), a comparison with a data point is made possible in the dominance test by assuming the value of the lower-left corner to be a point. This concept has been used in many index-based methods, such as Z-Sky [16]. In particular, in line 9 of Algorithm 8, comparisons occur frequently between the skyline point and MBR, and if the lower-left corner of the MBR is dominated by a specific skyline point, all of the data in the corresponding MBR are dominated by the corresponding skyline point. Therefore, in this case, as indicated in line 10, the corresponding MBR and its children are removed from the query. Therefore, even in the case of the MBR, the use of incomparability makes it possible to reduce unnecessary dominance tests, where dominance cannot occur.

The BBS algorithm with the *DC* applied is presented in Algorithm 8. Although the *DC_Init* call is different from the previous case, there was no change in utilizing the incomparability when the dominance test was changed to the *D-Classifier* and *DC*.

V. PERFORMANCE EVALUATION

In this section, we perform a performance evaluation of the proposed *DC*. First, we describe the experimental environment used to perform the evaluations. Then, we present the experimental results, where the superiority of the proposed method is confirmed by comparing the performance when *DC* was applied to existing algorithms and when it was not. We also provide an in-depth analysis of the experiment results, presenting what led to the performance improvements.

A. EXPERIMENTAL ENVIRONMENT

In skyline queries performance highly depends on the characteristics of the dataset, such as the number of dimensions and the distribution of data points. Therefore, to experimentally evaluate the skyline queries on various scenarios, we generated and used synthetic datasets with various distributions and various dimensions, using the generator proposed by Borzsony *et al.* [1]. The generated datasets had anti-correlated (correlation: −0.5), independent, and correlated (correlation: 0.5) distributions, and were organized into 4, 8, 12, and 16 dimensions for each distribution. Also, the data points belonging to each dataset consisted of real numbers with ten decimal places in the range [0, 1] for each dimension. In these synthetic datasets, the number of skyline points increases as the dimension or cardinality increases, and the number of skyline points increases in the order of correlated, independent, and anti-correlated, even when they have the same dimensions and cardinality.

Furthermore, to evaluate scalability in relation to dataset cardinality, the datasets were generated with 10K (ten thousand), 100K (one hundred thousand), 1M (one million), and 10M (ten million) data points, respectively. Also, to evaluate *DC* using real-world datasets, we evaluated three types of real-world datasets called Household [20], Gas [33], Weather [34]. The Household dataset consists of 128K data

TABLE III
SPECIFICATIONS OF DATASETS

Category	Dataset	Dimensionality	Cardinality
Synthetic	Correlated	{4, 8, 12, 16}	{10K, 100K, 1M, 10M}
Synthetic	Independent	{4, 8, 12, 16}	{10K, 100K, 1M, 10M}
Synthetic	Anti-correlated	{4, 8, 12, 16}	{10K, 100K, 1M, 10M}
Real	Household	6	127,931
Real	Gas	10	928,991
Real	Weather	15	566,268

points in 6 dimensions and consists of US census data on expenses such as electricity and mortgage. The Gas dataset consists of 929K data points in 10 dimensions and contains a record of a gas sensor array composed of eight metal oxide gas sensors, and temperature and humidity sensors for monitoring home activity. Lastly, the Weather dataset had 566K data points in 15 dimensions and consisted of average monthly precipitation totals and elevation at over half a million sensor locations obtained from the University of East Anglia climatic research unit. In Table III, the previously mentioned datasets are summarized once again.

To evaluate the *DC*, the first experiment shows the difference in the performance of the *DC-basic* skyline query when *Div* was used and when it was not. The second experiment shows the results of the comparison experiments with and without *DC* applied to existing skyline queries. For this comparison, we conducted experiments using the *DC* algorithm applications for SFS, SaLSa, and BBS proposed in Section 4. And in the last experiment, a comparative experiment was conducted using three real-world datasets.

Lastly, all the skyline query methods were implemented using C++ 14, and the experiments were carried out on an Intel Core i7-6700 3.4 GHz processor with 64-bit Windows 10 Pro and 16 GB of main memory.

B. EXPERIMENTAL RESULTS

In this subsection, we present the results of experiments, using *Div*, skyline computation time, and dominance test calls to evaluate *DC* from various aspects.

The first experiment shows the skyline query performance improvement based on *Div* usage. To do this, we experimentally show the difference in the performance of the *DC-basic* skyline query when *Div* was fixed to 1, to check all of the skyline points in the leaf node without division, and when *Div* for division was increased based on the number of skyline points. In this experiment, the *Div* was increased by one whenever the average number of skyline points in the leaf nodes increased by 64. In addition, to evaluate the difference in performance due to *Div* from various aspects, the experiment was configured to vary the dimensions for the 1M dataset, using various distributions, or to vary the number of data in 8-dimensions with the various distributions.

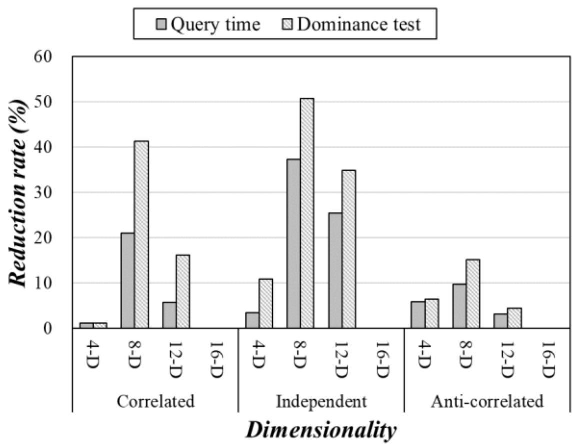


FIGURE 5. **The reduction rates in various dimensions when using *Div*.**

Fig. 5 shows the query time and dominance test reduction rates according to dimensions in various data distributions when *Div* was used. In this experiment, since the number of average skyline points per leaf node did not satisfy the *Div* increase criterion in 16-dimensions, there was no difference with the use of *Div*. However, in the other dimensions, the query time decreased from 1.2% to 37.3%, and the dominance test call decreased from 1.2% to 50.6%. The largest difference was shown in the 8-dimensions, where the average number of skyline points was the largest. This shows that even when the number of unnecessary dominance tests is minimized through incomparability, it is important to first use the skyline points with the high dominance probability for dominance tests, to eliminate data that are not selected as skylines early.

In addition, in this experiment, there was greater performance improvement with the independent dataset than with the correlated or anti-correlated datasets. This is because in the correlated dataset, the number of skyline points is small, so the value of *Div* is not frequently used. And, in the anti-correlated dataset, the probability of dominance between the data is very low, so that new data must be compared with most of the skyline points selected early. Accordingly, there was no significant difference in performance according to *Div*. However, a larger *Div* was used for the dataset with independent distribution, because the number of skylines was greater than that of the correlated dataset, and data could be removed early because the dominance probability between the data was higher than that of the anti-correlated dataset. Therefore, the performance improvement when using *Div* was most noticeable in the dataset with independent distribution.

Fig. 6 shows the query time and dominance test reduction rates according to cardinality when using *Div*. The experimental results show that as the cardinality of the dataset increased, the reduction in query time and dominant test calls with *Div* also increased. This occurs because an increase in cardinality leads to an increase in skyline points, and an increase in skyline points leads to an increase in *Div*.

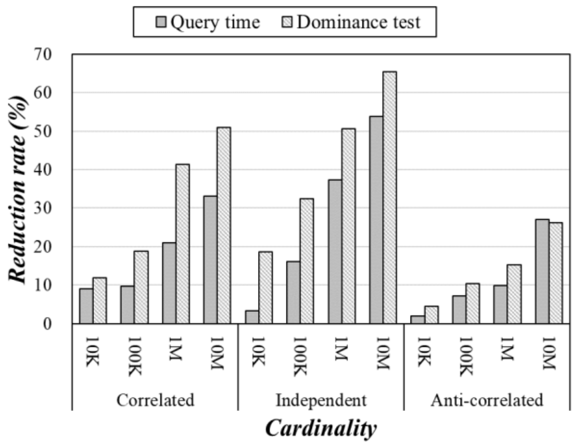


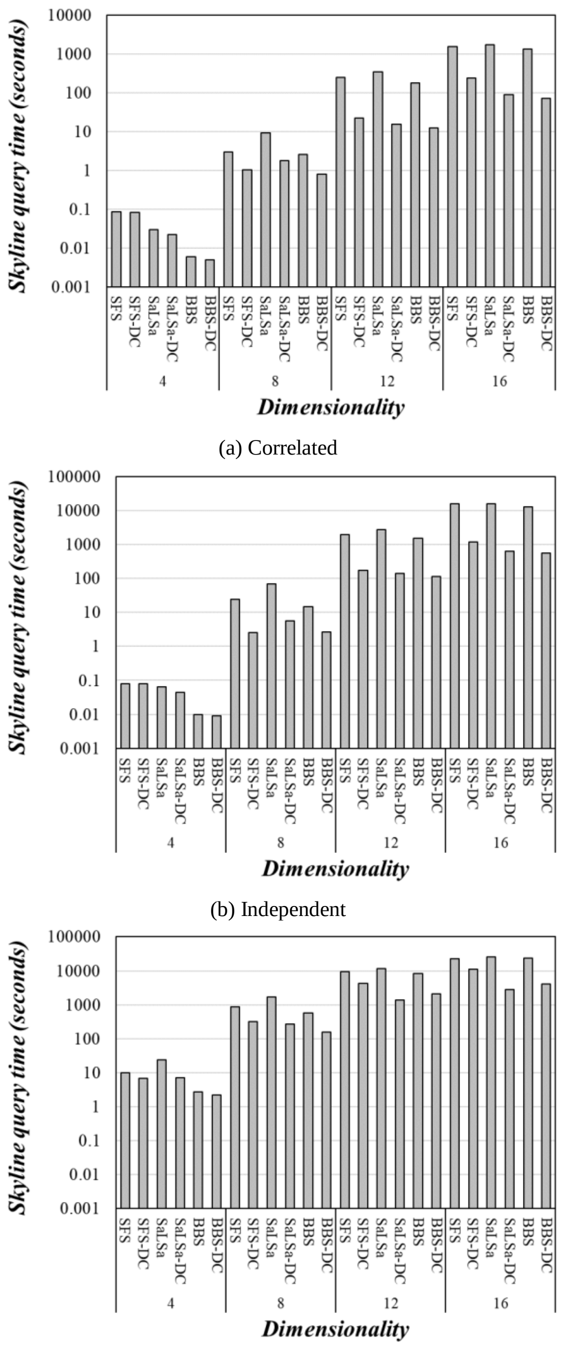
FIGURE 6. **The reduction rate in various cardinalities when using *Div*.**

This increased *Div* makes it possible to quickly eliminate data that cannot be skylines by allowing the skyline points with high dominance probability to be more preferentially used for dominance tests. Therefore, as the cardinality increases, the performance improvement due to *Div* also increases.

In the second experiment, to evaluate performance improvements when *DC* was applied to existing skyline query methods, the difference in the performances of the existing methods without the *DC* algorithm, and when *DC* was applied to them, was compared using various aspects.

Fig. 7 shows the skyline query time results for each dimension for SFS, SaLSa, and BBS, which are current skyline query methods, and *SFS-DC*, *SaLSa-DC*, and *BBS-DC*, when *DC* was applied to them, using a log scale. This experiment showed that the skyline query time was reduced in most cases for the methods that applied the *DC* algorithm. Specifically, the *DC*-applied methods significantly reduced the skyline query time from at least 50.5% to a maximum of 95.9% in 8-dimensions or more, compared with the existing methods. Also, in most cases, there was a more prominent reduction in the skyline query time of SaLSa and BBS than that of SFS. This is because, in the case of SaLSa, the time required to reach the stop point was significantly reduced because the unnecessary dominance test between data sorted through minC could be reduced through *DC*. And in the case of BBS, using *DC*, the point-to-group comparison, which is conducted at the beginning of the query, can be performed with a smaller number of skyline points. As a result, the large number of data belonging to the MBR can be eliminated more quickly, so that the skyline query time is significantly reduced.

Fig. 8 shows the number of dominant test calls that occurred when the same experiment shown in Fig. 7 was conducted. In this experiment, the number of dominant test calls decreased in all cases, and the reduction rate ranged



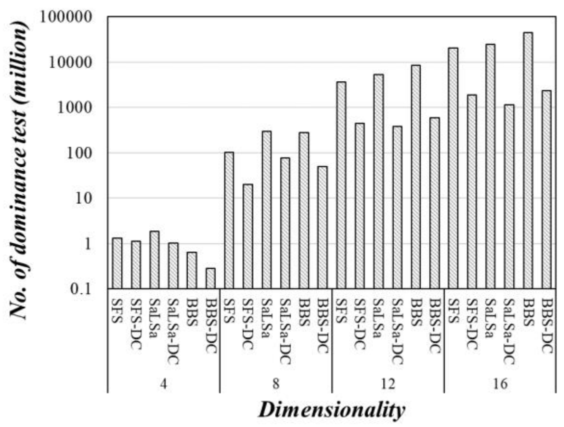
(a) Correlated

(b) Independent

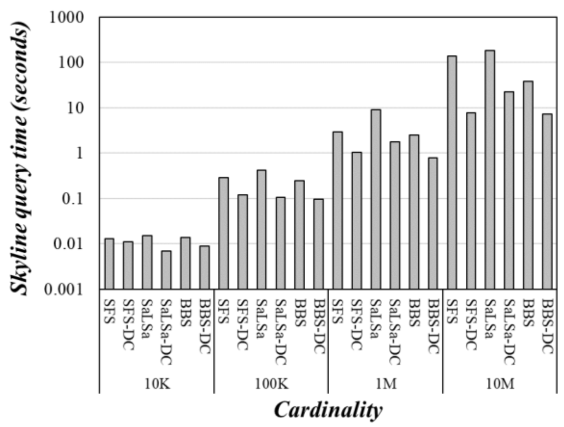
(c) Anti-correlated

FIGURE 7. **Skyline query time according to dimensionality.**

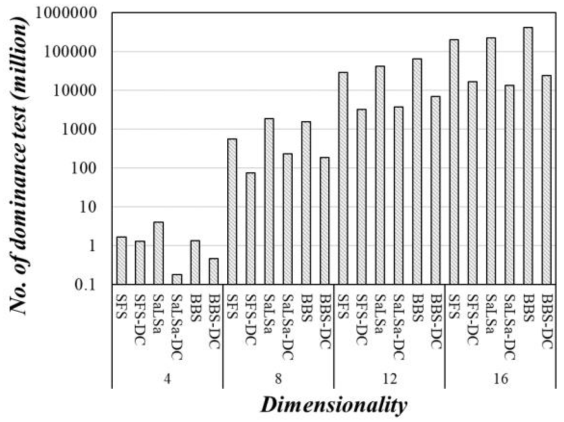
from a minimum of 11.8% to a maximum of 95.5%. Notably, the correlated and independent datasets showed at least a 73.6% reduction in dominant test calls over 8-dimensions. However, in the anti-correlated dataset, only 24.5% to 83% reduction in dominant test calls occurred, because even with *DC*, as the number of skylines increased, the dominance tests needed to confirm skyline points also accumulated.



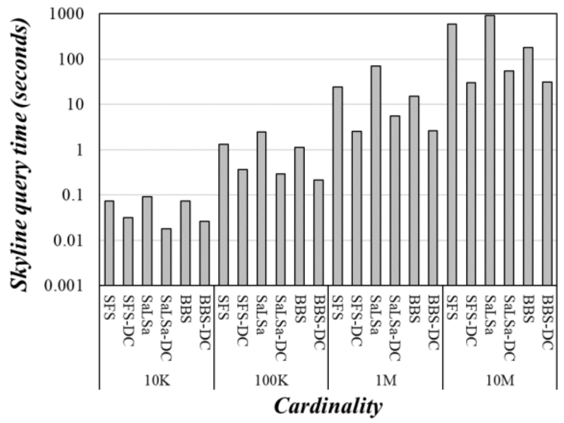
(a) Correlated



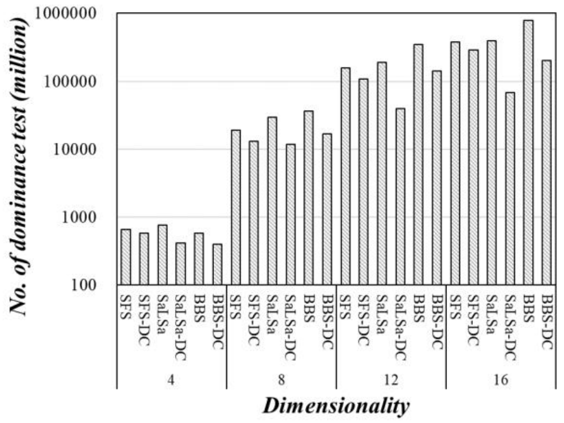
(a) Correlated



(b) Independent

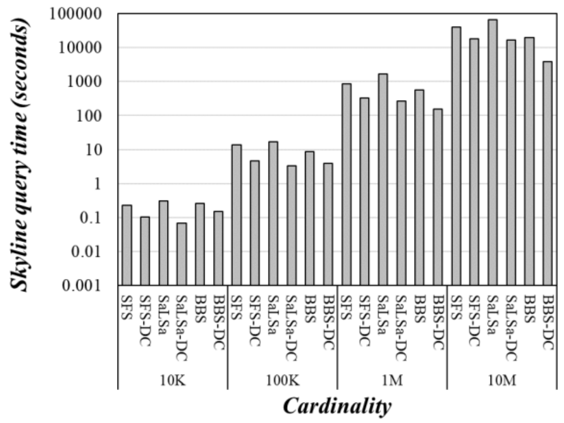


(b) Independent



(c) Anti-correlated

FIGURE 8. **Number of dominance test calls according to dimensionality.**

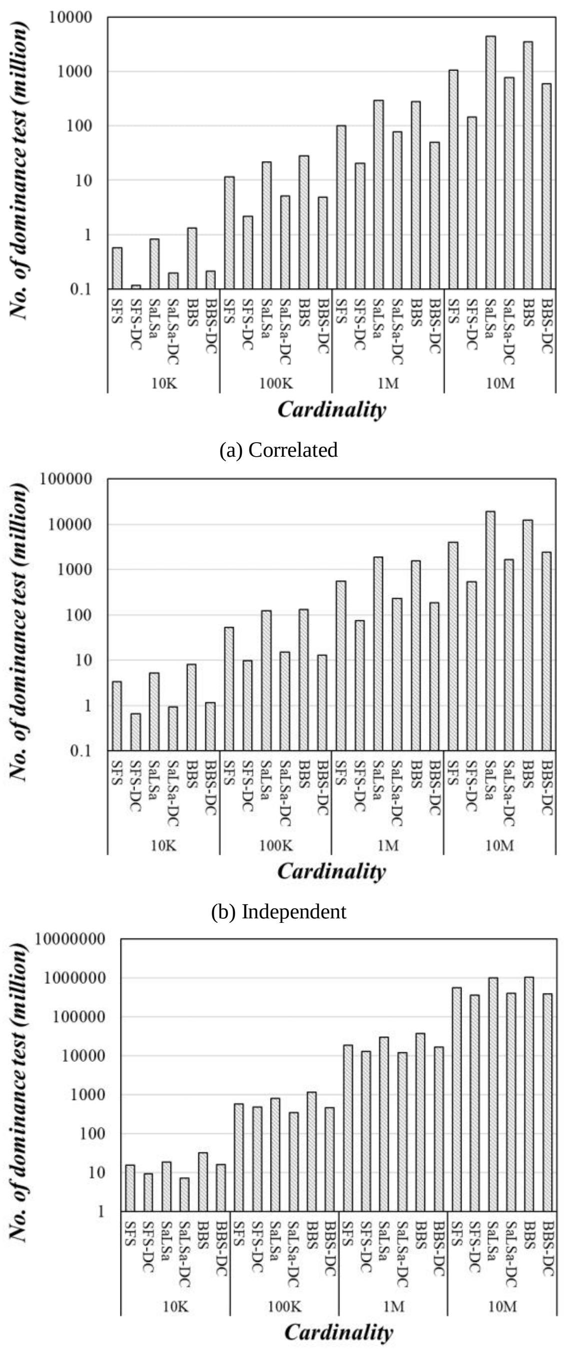


(c) Anti-correlated

FIGURE 9. **Skyline query time according to cardinality.**

Fig. 9 shows the difference in skyline query time for the existing methods and *DC* applied methods according to cardinality. In this experiment, except for the anti-correlated datasets, as the cardinality increased, the degree of decrease in skyline query time declined due to *DC*. In addition, in the anti-correlated dataset, there were cases where the degree of decrease in query time declined compared to previous performance, for certain cardinalities depending on the method, but most of them increased. And, for advanced cardinality of 100K or more with increased *Div*, the skyline query time decreased from a minimum of 55.2% to a maximum of 94.9%.

Fig. 10 shows the number of dominant test calls obtained with the same experiment as the one in Fig. 9. The rate of reduction in dominance test calls seen in this experiment was generally similar to the reduction rate for skyline query time. This is because most of the time consumed in the skyline query occurs in the dominance test. This confirms that the

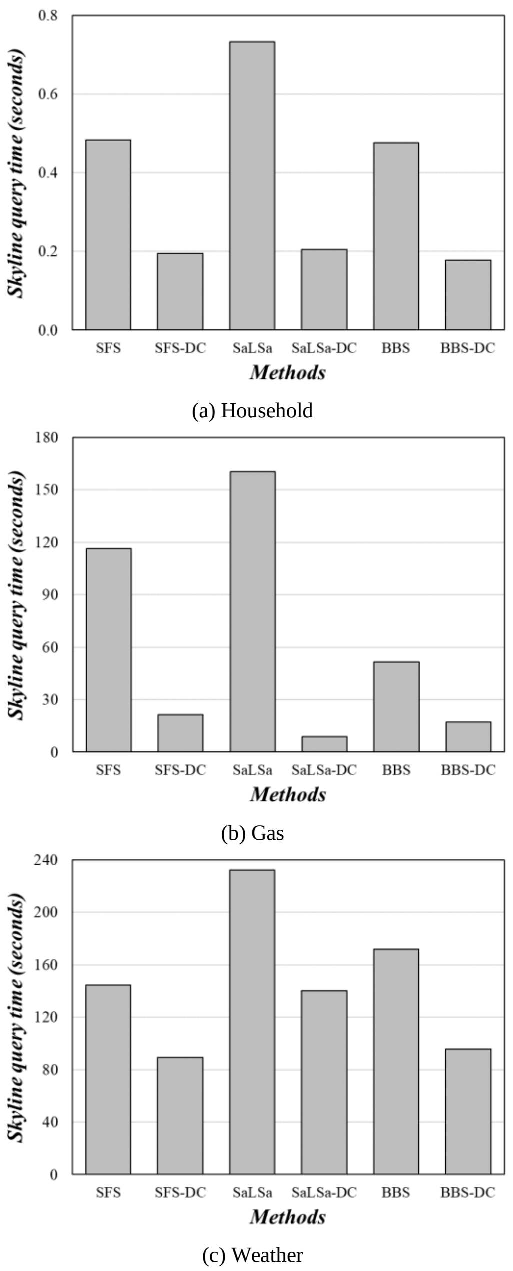


(a) Correlated

(b) Independent

(c) Anti-correlated

FIGURE 10. Number of dominance test calls according to cardinality.



(a) Household

(b) Gas

(c) Weather

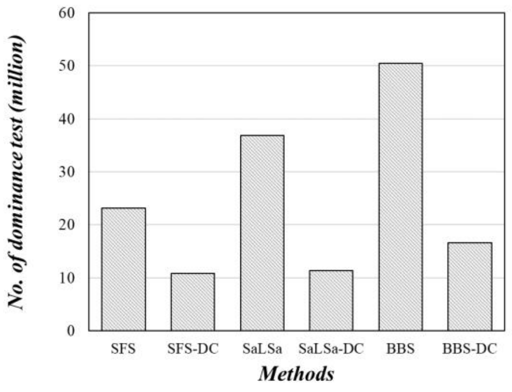
FIGURE 11. Skyline query time according to real-world datasets.

reduction in skyline query time with *DC* can be largely attributed to the reduction in dominance test calls.

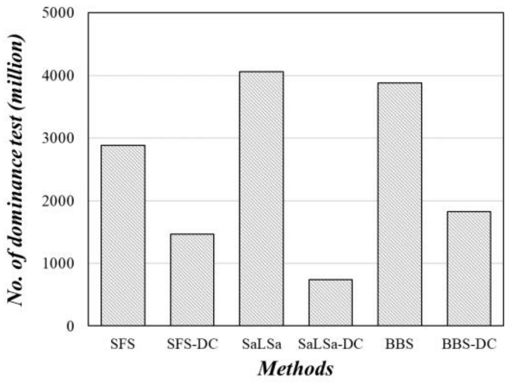
In the final experiment, a comparative experiment was conducted using three real-world datasets called Household, Gas, and Weather.

Fig. 11 shows the skyline query time results using the three types of real-world datasets. In this experiment, all the *DC*-applied methods showed a reduction in skyline query time compared with the existing methods, and the skyline search was accomplished with a maximum of 94.4% less time. Fig. 12 shows the number of dominant test calls obtained in the same experiment using real-world datasets, as shown in Fig. 11. This experiment showed that when *DC* was applied, it was possible to search the skyline with fewer than 16.5% to 81.8% of the dominance test calls compared with the existing method. The results of these experiments show that by using *DC* it is possible to reduce the skyline query time and dominance test calls that occur in the existing skyline query methods. These results are consistent with the experimental results using the synthetic datasets.

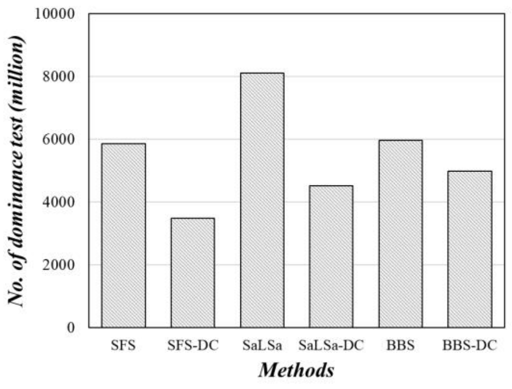
The various experimental results indicate that when *DC* is used, the number of unnecessary dominance tests performed in existing methods can be effectively reduced, using incomparability obtained from the decision tree. As a result,



(a) Household



(b) Gas



(c) Weather

FIGURE 12. Number of dominance test calls according to real-world datasets.

with the proposed method, we can minimize dominance tests, leading to a reduction in skyline query time. This result can be particularly helpful to solve the known problems with skyline queries, which have limitations at high dimensions and with high cardinality data.

VI. CONCLUSION

In this paper, we have proposed a *decision tree-based comparator* (*DC*) to optimize dominance tests for skyline queries. There were three key findings. First, the proposed *DC* allowed us to eliminate leaf nodes and their data points when they exhibited incomparability with the current data point, thereby avoiding unnecessary dominance tests. Second, the proposed *DC* method was easily applied to improve the performance of various existing skyline query methods because of its unique tree structure. Third, using various experiments, we demonstrated that *DC* can reduce skyline query time and dominance test calls in existing methods by up to 95.9% and 95.5%, respectively.

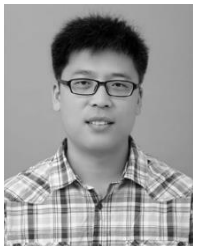
It is important to note that further considerations are required when applying the proposed *DC* to distributed and parallel processing environments, or incomplete data as mentioned in Section 2. Thus, in future work, we plan to conduct research to demonstrate the effectiveness of the proposed *DC* for reducing dominance testing that occurs while searching for local and global skylines in a distributed and parallel processing environment. We are also planning to conduct a study that utilizes the concept of incomparability with the proposed *DC* method, even with incomplete data.

REFERENCES

[1] S. Borzsony, D. Kossmann, and K. Stocker, “The Skyline operator,“ in *Proc. 17th ICDE*, Heidelberg, Germany, 2001, pp. 421–430, DOI: https://doi.org/10.1109/ICDE.2001.914855.
[2] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou, “Parallel distributed processing of constrained skyline queries by filtering,“ in *Proc. 24th ICDE*, Cancun, Mexico, 2008, pp. 546–555, DOI: https://doi.org/10.1109/ICDE.2008.4497463.
[3] Y. Chen and C. Lee, “Skyline Path Queries With Aggregate Attributes,“ *IEEE Access*, vol. 4, pp. 4690–4706, Aug. 2016, DOI: https://doi.org/10.1109/ACCESS.2016.2602702.
[4] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi, “Skyline queries against mobile lightweight devices in MANETs,“ in *Proc. 2nd ICDE*, Atlanta, GA, USA, 2006, p. 66, DOI: https://doi.org/10.1109/ICDE.2006.142.
[5] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu, “Efficient skyline query processing on peer-to-peer networks.“ in *Proc. 23rd ICDE*, Istanbul, Turkey, 2007, pp. 1126–1135, DOI: https://doi.org/10.1109/ICDE.2007.368971.
[6] A. Nasridinov, S.-Y. Ihm, Y.-H. Park, “Skyline-based aggregator node selection in wireless sensor networks,” *Int. J. Distrib. Sens. Netw.*, vol. 9, Sept. 2013, Art no. 356194, DOI: https://doi.org/10.1155/2013/356194.
[7] W. T. Balke, U. Güntzer, and J. X. Zheng, “Efficient distributed skylining for web information systems,” in *Advances in Database Technology – EDBT 2004*, E. Bernito et al., Eds. Springer, Berlin, Heidelberg, 2004, pp. 256–273, DOI: https://doi.org/10.1007/978-3-540-24741-8_16.
[8] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis, “Ranking and clustering web services using multicriteria dominance relations,” *IEEE Trans. Serv. Comput.*, vol. 3, no. 3, pp. 163–177, July-Sept. 2010, DOI: https://doi.org/10.1109/TSC.2010.14.
[9] Y. Zhang, J. Li, Z. Zhou, and X. Liu, “Efficient dynamic service maintenance for edge services,” *IEEE Access*, vol. 6, pp. 8829–8840, Feb. 2018, DOI: https://doi.org/10.1109/ACCESS.2018.2806391.
[10] C. Yu, X. He, H. Ma, X. Qi, J. Lu, and Y. Zhao, “S-DenseNet: A DenseNet compression model based on convolution grouping strategy using skyline method,” *IEEE Access*, vol. 7, pp. 183604–183613, Dec. 2019, DOI: https://doi.org/10.1109/ACCESS.2019.2960315.
[11] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, “Skyline with presorting,” in *Proc. 19th ICDE*, Bangalore, India, 2003, pp. 717–719, DOI: https://doi.org/10.1109/ICDE.2003.1260846.
[12] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, “Skyline with presorting: Theory and optimizations,” in *Intelligent Information Processing and Web Mining*, M. A. Kłopotek, S. T. Wierzchoń, and K. Trojanowski, Eds. Springer, Berlin,

Heidelberg, 2005, pp. 595–604, DOI: https://doi.org/10.1007/3-540-32392-9_72.
[13] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in *Proc. 31st VLDB*, Trondheim, Norway, 2005, pp. 229–240.
[14] I. Bartolini, P. Ciaccia, M. Patella, SaLSa: computing the skyline without scanning the whole sky, in *Proc. 15th CIKM*, New Yor, NY, USA, 2006, pp. 405–414, DOI: https://doi.org/10.1145/1183614.1183674.
[15] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proc. SIGMOD/PODS03*, San Diego, CA, USA, 2003, pp. 467–478, DOI: https://doi.org/10.1145/872757.872814.
[16] K. C. K. Lee, W.-C Lee, B. Zheng, H. Li, and Y. Tian, "Z-SKY: an efficient skyline query processing framework based on Z-order," *VLDB J.*, vol. 19, no. 3, pp. 333–362, Jun. 2010, DOI: https://doi.org/10.1007/s00778-009-0166-x.
[17] J.-H. Choi, F. Hao, and A. Nasridinov, "HI-Sky: Hash index-based skyline query processing," *Appl. Sci*., vol. 10, no. 5, Mar. 2020, Art no. 1708, DOI: https://doi.org/10.3390/app10051708.
[18] M. Morse, J. M. Patel, H. V. Jagadish, "Efficient skyline computation over low-cardinality domains," in *Proc. 33rd VLDB*, Vienna, Austria, 2007, pp. 267–278.
[19] S. Zhang, N. Mamoulis, and D. W. Cheung, "Scalable skyline computation using object-based space partitioning," in *Proc. SIGMOD*, Providence, RI, USA, 2009, pp. 483–494, DOI: https://doi.org/10.1145/1559845.1559897.
[20] J. Lee and S. Hwang, "Scalable skyline computation using a balanced pivot selection technique," *Inf. Syst*., vol. 39, pp. 1–21, Jan. 2014, DOI: https://doi.org/10.1016/j.is.2013.05.005.
[21] K. Koizumi, P. Eades, K. Hiraki, and M. Inaba, "BJR-tree: fast skyline computation algorithm using dominance relation-based tree structure," *Int. J. Data Sci. Anal*., vol. 7, pp. 17–34, Feb. 2019, DOI: https://doi.org/10.1007/s41060-018-0098-x.
[22] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi, "Parallelizing skyline queries for scalable distribution," in *Advances in Database Technology – EDBT 2006*, Y. Ionnidis et al., Eds. Springer, Berlin, Heidelberg, 2006, pp. 112–130, DOI: https://doi.org/10.1007/11687238_10.
[23] A. Vlachou, C. Doulkeridis, and Y. Kotidis, "Angle-based space partitioning for efficient parallel skyline computation," in *Proc. SIGMOD*, Vancouver, Canada, 2008, pp. 227–238, DOI: https://doi.org/10.1145/1376616.1376642.
[24] G. Valkanas and A. N. Papadopoulos, "Efficient and adaptive distributed skyline computation," in *SSDBM 2010*, M. Gertz and B. Ludäscher, Eds. Springer, Berlin, Heidelberg, 2010, pp. 24–41, DOI: https://doi.org/10.1007/978-3-642-13818-8_4.
[25] A. Nasridinov, J.-H. Choi, and Y.-H. Park, "A two-phase data space partitioning for efficient skyline computation," *Cluster Comput.*, vol. 20, no. 4, pp. 3617–3628, Dec. 2017, DOI: https://doi.org/10.1007/s10586-017-1070-6.
[26] C. Kalyvas and M. Maragoudakis, "Skyline and reverse skyline query processing in SpatialHadoop," *Data Knowl. Eng.*, vol. 122, pp. 55–80, Jul. 2019, DOI: https://doi.org/10.1016/j.datak.2019.04.004.
[27] W. Wang, "A scalable spatial skyline evaluation system utilizing parallel independent region groups," *VLDB J.*, vol. 28, no. 1, pp. 73–98, Feb. 2019, DOI: https://doi.org/10.1007/s00778-018-0519-4.
[28] M. Tang, Y. Yu, W. G. Aref, Q. M. Malluhi and M. Ouzzani, "Efficient Parallel Skyline Query Processing for High-Dimensional Data," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 10, pp. 1838-1851, Feb. 2018, DOI: https://doi.org/10.1109/TKDE.2018.2809598.
[29] K. Zhang, H. Gao, X. Han, Z. Cai and J. Li, "Modeling and Computing Probabilistic Skyline on Incomplete Data," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 07, pp. 1405-1418, Mar. 2020, DOI: https://doi.org/10.1109/TKDE.2019.2904967.
[30] Y. Gulzar, A. A. Alwan and S. Turaev, "Optimizing Skyline Query Processing in Incomplete Data," *IEEE Access*, vol. 7, pp. 178121-178138, Dec. 2019, DOI: https://doi.org/10.1109/ACCESS.2019.2958202.
[31] Y. Gulzar, A. A. Alwan, S. Turaev, S. Wani, A. B. Soomo and Y. Hamid, "IDSA: An Efficient Algorithm for Skyline Queries Computation on Dynamic and Incomplete Data With Changing States," *IEEE Access*, vol. 9, pp. 57291-57310, Apr. 2021, DOI: https://doi.org/10.1109/ACCESS.2021.3072775.
[32] L. Ding, X. Zhang, H. Zhang, L. Liu and B. Song, "CrowdSJ: Skyline-Join Query Processing of Incomplete Datasets With Crowdsourcing," *IEEE Access*, vol. 9, pp. 73216-73229, May. 2021, DOI: https://doi.org/10.1109/ACCESS.2021.3079324.
[33] R. Huerta, T. Mosqueiro, J. Fonollosa, N.F. Rulkov, and I. Rodriguez-Lujan, "Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring," *Chemom. Intell. Lab. Syst.*, vol. 157, no. 15, pp. 169–176, Oct. 2016, DOI: https://doi.org/10.1016/j.chemolab.2016.07.004.
[34] S. Chester, D. Šidlauskas, I. Assent, and K. S. Bøgh, "Scalable parallelization of skyline computation for multi-core processors," in *Proc. 31st ICDE*, Seoul, South Korea, 2015, pp. 1083-1094, DOI: https://doi.org/10.1109/ICDE.2015.7113358.

JONG-HYEOK CHOI received the B.Sc. degree in computer education from Chungbuk National University, South Korea, in 2015, and the M.Sc. and Ph.D. degrees in computer science from Chungbuk National University 2017 and 2021, respectively. He is currently a member of the Data Analysis Laboratory led by Professor Aziz Nasridinov and a postdoctoral researcher at the Bigdata Research Institute of Chungbuk National University. His research interests include traditional databases, big data analysis, artificial intelligence, computer vision, and e-learning solution for K-12 students and teachers.



FEI HAO received the B.Sc. degree in Information and Computing Science and the M.Sc. degree in Computer Software and Theory from Xihua University, China, in 2005 and 2008, respectively, and the Ph.D. degree in Computer Science and Engineering from Soonchunhyang University, South Korea, in 2016. Since 2016, he has been with Shaanxi Normal University, Xi'an, China. He is currently taking a Marie Sklodowska-Curie Individual Fellowship at the

University of Exeter, Exeter, United Kingdom. His research interests include social computing, ubiquitous computing, big data analysis and processing and mobile cloud computing.

YOO-SUNG KIM is a professor of Department of Artificial Intelligence Engineering at Inha University. He received BS in computer sciences in 1986 from Inha University, MS in computer sciences in 1988 and PhD in computer sciences in 1992 from KAIST (Korea Advanced Institute of Science and Technology), respectively. He worked at Samsung Electronic as a researcher from 1990 to 1992 and at Purdue University in 1996 as a visiting scholar, and at Indiana University in 2005 as a visiting research professor. His research areas are multimedia database, data mining, and intelligent video surveillance systems.

AZIZ NASRIDINOV received the B.Sc. degree from the Tashkent University of Information Technologies, Uzbekistan, in 2006, and the M.Sc. and Ph.D. degrees in computer engineering from Dongguk University in 2009 and 2012, respectively. He is currently an Associate Professor in Data Analytics Laboratory, Department of Computer Science, Chungbuk National University. He has also served as a program committee member and co-organizer for numerous top-tier conferences, including ACM SAC, IEEE Big Data, IEEE Globecom, AAAI and CHI, and also served on the editorial board of several international journals. His research interests include traditional databases, big data analytics with machine learning, and computer vision.