Hindawi
Wireless Communications and Mobile Computing
Volume 2017, Article ID 9309181, 13 pages
https://doi.org/10.1155/2017/9309181

WILEY | Hindawi

Research Article

An Indexing Method to Construct Unbalanced Layers for High-Dimensional Data in Mobile Environments

Sun-Young Ihm,[1] Jae-Hee Hur,[2] and Young-Ho Park[2]

[1]*Big Data Using Research Center, Sookmyung Women's University, Cheongpa-ro 47-gil 100, Yongsan-Ku, Seoul 04310, Republic of Korea*
[2]*Department of IT Engineering, Sookmyung Women's University, Cheongpa-ro 47-gil 100, Yongsan-Ku, Seoul 04310, Republic of Korea*

Correspondence should be addressed to Young-Ho Park; yhpark@sm.ac.kr

Received 28 July 2017; Accepted 27 September 2017; Published 20 November 2017

Academic Editor: B. B. Gupta

A top-k query processing is widely used in many applications and mobile environments. An index is used for efficient query processing and layer-based indexing methods are representative to perform the top-k query processing efficiently. However, the existing methods have a problem of high index building time for multidimensional and large data; thus, it is difficult to use them. In this paper, we proposed a new concept of constructing layer-based index, which is called unbalanced layer (UB-Layer). The existing methods construct a layer as a balanced layer with outermost data and wrap the rest of the input data. However, UB-Layer constructs a layer as an unbalanced layer that does not wrap the rest of the data. To construct UB-Layer, we fist divide the dimension of the input data into divided-dimensional data and compute the convex hull in each divided-dimensional data. And then, we combine divided-convex hull to build UB-Layer. We also propose UB-SelectAttribute algorithm for dividing the dimension with major attributes. We demonstrate the superiority of the proposed methods by the performance experiments.

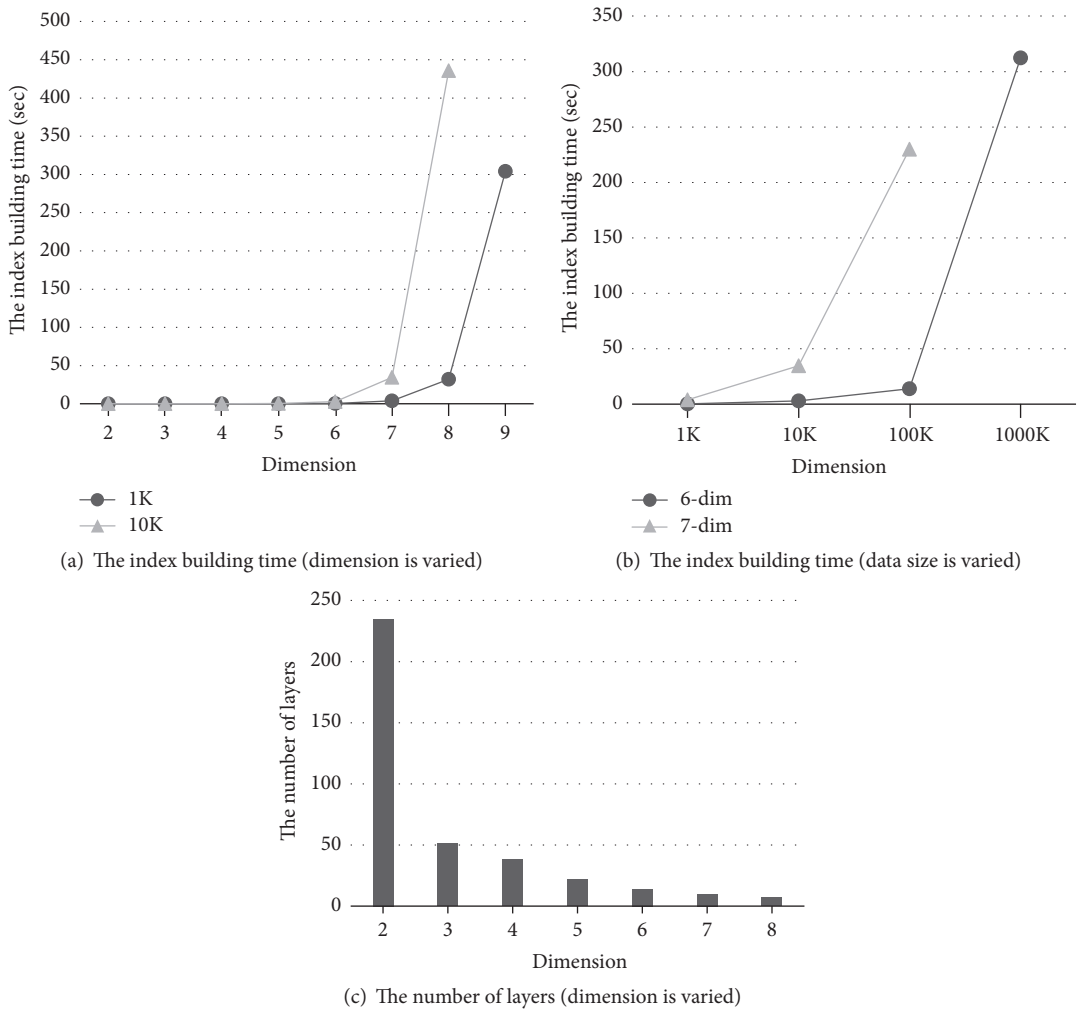
1. Introduction

Recently, Searching user interested data among a large amount of data has become very important as the size of databases grows. In particular, searching the data that users want over high-dimensional and large data is more difficult in mobile environments, because it is the only part of the massive amount of data that users are interested in. However, especially in the mobile environment, efficient retrieval from vast amounts of data is limited.

Furthermore, as the attributes of the data are varied, problems arise in index construction and query processing for the high-dimensional and large database. For example, when users are searching for a used car, they should consider the various attributes such as manufacturer, model, rating, year, price, region, mileage, fuel, transmission, and color, and it will take a long time.

Top-k query processing retrieves k data that is most interested in from a given data considering various attributes. Top-k query processing is handled in various applications such as searching hotels or cameras on the websites [1–3], healthcare [4], and cloud environments [5–7]. Top-k query processing retrieves k results that users want by considering various attributes in a large amount of data, and it is very inefficient to search all data at each query. Therefore, it is important to build an index in order to perform top-k query processing efficiently. There have been a number of studies that construct an index by making layers over the entire database. These methods find top-k answers by accessing just the first few layers.

The convex hull [8] is a representative method of index construction method for top-k query processing. The method reads total data and constructs indices into the layer and the convex hull is the outer layer surrounding all data layer lists geometrically. Since the convex hull algorithm constructs layer lists based on the control points from all direction, the method has the advantage of processing omnidirectional queries. Even though the method is advantageous in query processing, convex hull includes a small amount of data in one layer so that the construction time is inefficient.



(a) The index building time (dimension is varied)

(b) The index building time (data size is varied)

(c) The number of layers (dimension is varied)

Figure 1: The experimental results of constructing the convex hull.

Figure 1 shows the experimental result of constructing the convex hull as dimension and size of data are varied. As shown in Figure 1(a), as the dimension increases, the computing time increases sharply, and if data has more than 9 attributes, in the case of 10K data, the experiment is impossible. Figure 1(b) shows the computing time as data size varied, and when the size of data increases, the construction time of convex hull increases sharply. And if the data size is more than 1000K, the experiment is also impossible. Figure 1(c) shows the number of layers in the convex hull as the dimension is varied. If the dimension increases, the number of layers decreases sharply. That is, the number of data contained in a one layer increases as the number of layers increases.

As the number of attributes and the size of the data increase, the time for constructing the convex hull increases sharply and the number of layers decreases. Therefore, it is almost impossible to build the convex hull from a high-dimensional and large database.

In this paper, we propose a UB-Layer (unbalanced layer) which is a new concept of unbalanced layer considering all direction of query processing, and we also propose a method of building UB-Layer. The proposed method is a study to construct an index to efficiently perform top-k query processing, and query processing is not covered in this paper. The UB-Layer is an unbalanced layer because it is constructed around the outermost points but also contains some inner points. In order to build the UB-Layer, we first find out the main attributes of input data, and then we divide the input data with the main attributes. Next, we construct the divided-convex hull based on the divided data, and, finally, we combine the divided-convex hull to build a final layer. UB-Layer is a more efficient method for top-k query processing because it builds layers much faster and has more layers than convex hull.

More precisely, we make the following contributions in this paper:

(1) We propose a new concept of unbalanced layer method for the high-dimensional and large database, called the UB-Layer (unbalanced layer). UB-Layer first makes a divided dataset by dividing the dimension of input data and constructs a divided-convex hull in each divided dataset. The

proposed method could build a list of layers in the high-dimensional and large dataset and reduces the index building time compared to the convex hull.

(2) We propose a method, which is called UB-SelectAttribute, to divide a dimension by selecting major attributes for improving the precision of the UB-Layer. UB-SelectAttributes divide the dimension using the major attributes of input data.

(3) We show the performance advantages of UB-Layer through various experiments. We compare the index building time, the number of layers, and precision of UB-Layer with the existing method convex hull.

The rest of this paper is organized as follows. Section 2 describes the background and Section 3 describes the existing work related to this paper. Section 4 explains the proposed method, UB-Layer. Section 5 presents the results of performance evaluation. Section 6 summarizes and concludes the paper.

2. Background

In this section, we describe the background related to this paper. We first describe a top-k query processing in Section 2.1 and describe a convex hull method for mobile computing.

2.1. Top-k Query Processing. Top-k query processing retrieves k results from input data. The following example shows an example of top-k query processing to search for a hotel.

Example 1. Suppose a customer *Tom* who wants to book a hotel for a summer vacation searches for a hotel on a website. A hotel has various attributes such as rates, hotel ratings, customer ratings, distance from subway stations, distance from major attractions, and type of accommodation. The customer *Tom* considers rates and distance from the subway station for his reservation, and he wants to search a hotel with small attribute values. In this example, we will consider only two attributes for easy explanation. He also wants to set the weights for each attribute as (0.6: rates, 0.4: distance) as search conditions. The scoring function f for this is $f(t) = 0.6 * \text{rates} + 0.4 * \text{distance}$.

```
SELECT      *
FROM        Hotels
ORDER BY    f(t)
STOP AT     3
```

Table 1 shows the search result according to the query of the customer Tom. According to the scoring function, the top 1 result is hotel C, the top 2 result is hotel E, and the top 3 result is hotel B among the five hotels.

2.2. Convex Hull Method for Mobile Computing. Convex hull is a method to construct a list of layers with the set of outermost points. There are several studies construct convex hull for mobile computing and wireless network environments. Mobile computing requires wireless network access [9], and it is much more difficult than wired communication because mobile computing should consider the signal, path, noise, delays, errors, power, and so on. One of the basic issues of mobile computing and wireless network is the energy constraint [10]. Thus, the high efficiency service is needed for mobile computing. Convex hull is used to retrieve the user interested data and retrieve sensors to connect in mobile computing.

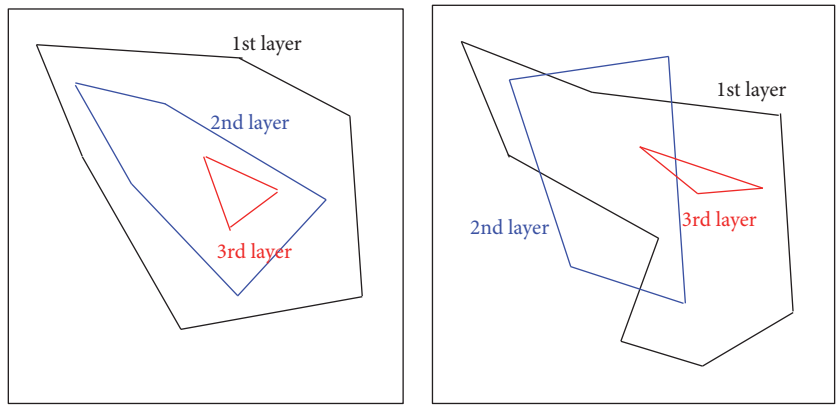
Table 1: An example of top-k query processing.

Hotels	Rates (10$)	Distance (km)	$f(t)$
A	5	15	9
B	12	2	8
C	7	5	6.2
D	20	3	13.2
E	8	5	6.8

3. Related Work

In this section, we explain convex hull method and discuss the existing work. The representative methods using the convex hull for constructing a list of layers are ONION [11], HL-Index [12, 13], and aCH-Index [14]. ONION constructs an index by making layers with the vertices of the convex hull over the objects in the multidimensional space. That is, it creates the first layer with the convex hull vertices over all objects and then creates the second layer with the convex hull vertices over the remaining objects, and so on. Finally, the set of layers becomes the layer list. HL-index constructs a layer list with the convex hull as Onion does and then constructs sorted lists in the ascending or descending order based on each attribute value of the objects in each layer. HL-index keeps those lists in order to fasten query processing. The aCH-Index first finds the skyline points over entire objects and then partitions the skyline layer into multiple subregions. Then, aCH-Index combines the convex hull by computing over in each subregion. Convex hull method can perform top-k query processing from every direction. However, the method has the problem of the high time complexity in index generation.

Besides, there are several methods such as the method that uses R-tree to construct convex hull [15], CONHEM (Convex-hull Edge Method) which is the method that sets minimum region to calculate the allowance [16], and VAICH (Visual Attention Imitation Convex Hull) which is based on the human prospective of constructing convex hull [17]. VAICH constructs the convex hull by eliminating data from the centroid to the limit. Also, other research using GPU to fasten the calculation for constructing convex hull. Tang et al. [18] proposed the method that uses both GPU and CPU simultaneously in order to construct convex hull. When it comes to get the input, the method eliminates initial point by GPU based filter and constructs the convex hull by using CPU based algorithm. Reference [19] proposed the parallel algorithm called CudaHull. CudaHull constructs 3-dimensional convex hull using GPU from CUDA programming. gHull [20] maximizes the parallelization based on the relationship between the 3-dimensional Voronoi diagram



(a) A balanced layer (b) An unbalanced layer

Figure 2: The concept of the UB-Layer.

and 3-dimensional convex hull and uses GPU to constructs 3-dimensional convex hull. Singh et al. [21] proposed the convex hull approach in conjunction with Gaussian mixture model in order to improve the detection accuracy without capitalizing much of computation time. CHVS [22] is a fast convex hull vertices selection algorithm for online classification and the proposed method converts convex hull into a linear equation problem with a low computational complexity.

There are also several methods to construct convex hull for mobile environments. Jiang et al. [23] proposed a depth-adjustment deployment algorithm based on two-dimensional convex hull for network energy efficiently. Kundu et al. [24] proposed an on-demand power saving routing algorithm for mobile ad hoc networks, which identifies node using k-means and convex hull algorithm. Xu et al. [25] proposed a path selection algorithm using convex hull to reduce the data delivery latency on the mobile elements in wireless sensor networks. However, existing methods did not study how to construct convex hull efficiently and have used convex hull to select nodes or data. In addition, they have not considered a high-dimensional data. Thus, in this paper, we present the efficient algorithm to construct convex hull for high-dimensional data.

4. UB-Layer (Unbalanced Layer)

In this section, we propose a new layer-based index building method for high-dimensional and large database, which is called UB-Layer (unbalanced layer). As we mentioned in Introduction, the constructing time of convex hull increases exponentially when the dimension of the data increases. And also the number of layers decreases; thus the convex hull is actually impossible to use for the Top-k processing. The UB-Layer improves the convex hull by dividing the dimension of input data and it reduces the computing time and increases the number of layers. Figure 2 shows the concept of UB-Layer. First, Figure 2(a) is the result of index constructed by the existing layer-based indexing method. It consists of balanced layers which the outside layer covers with the other layers. Figure 2(b) shows unbalanced layer which this research suggests; it consists of layers which the outside layer does not cover with the other layers. We define this type of layers with unbalanced layer in this research.

UB-Layer is made by 3 steps: (1) dimension division step; (2) constructing divided-convex hull step; (3) combining step. First, we divide the dimension of input data to create data that contains divided dimension in divided dimension level. Second, we create divided-convex hull based on the data with divided dimension. Finally, we create final UB-Layer to merge each divided-convex hull in merging level.

4.1. Dimension Division Step. In this section, we explain the first step, dimension division step to construct UB-Layer. Divided dimension level divides the dimension of input data.

4.1.1. UB-Basic Algorithm. In this section, we propose UB-Basic algorithm for dividing the dimension of the input data. UB-Basic algorithm is a naïve method which divides the dimension of the input data into two divided-dimensional datasets. Here, let $D_{1:n}$ be a given dataset with n attributes. UB-Basic algorithm divides $D_{1:n}$ into two divided-dimensional datasets $D_{1:k}$ and $D_{k+1:n}$ $(k = n/2)$. For example, in the case of 6-dimensional data, it is divided into two 3-dimensional datasets and 5-dimensional data is divided into 2-dimensional and 3-dimensional datasets. Table 2 shows the process of dividing the 4-dimensional input data into two 2-dimensional datasets div_1 and div_2 using UB-Basic algorithm, and Figure 3 shows the result of div_1 and div_2. In Table 2, the original dataset has four attributes X_1 to X_4, and UB-Basic algorithm divides the input data into two divided-dimensional datasets where one has two attributes X_1 and X_2 and other has X_3 and X_4.

Algorithm 1 shows the UB-Basic algorithm for dimension division. The input of the algorithm is S, which is a set of d-dimensional data objects, and d, which means the number of dimensions, that is, the number of attributes. And the result of the UB-Basic algorithm is div, which is a set of data

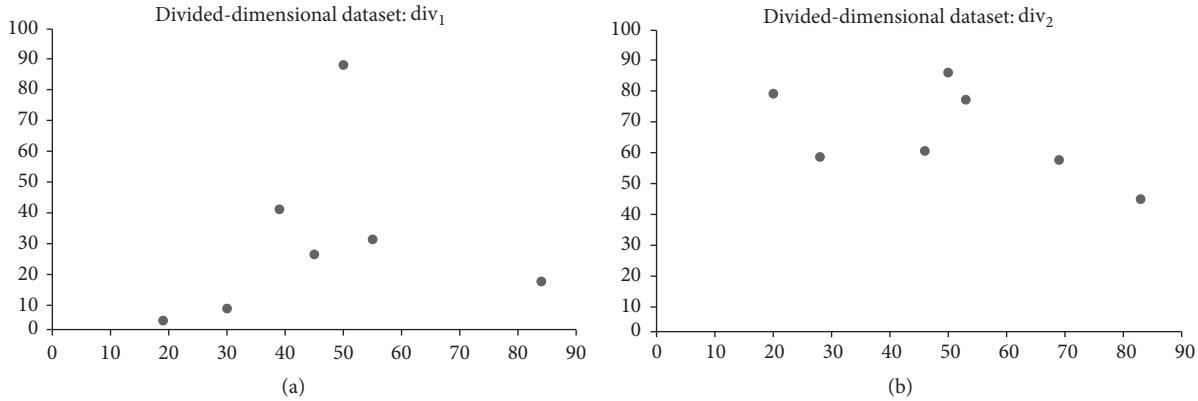


Figure 3: The result of UB-Basic algorithm with 4-dimensional dataset.

Table 2: The process of dimension division step using UB-Basic algorithm.

ID	value $\langle X_1, X_2, X_3, X_4 \rangle$		id	value $\langle X_1, X_2 \rangle, \langle X_3, X_4 \rangle$
p1	$\langle 45, 25, 83, 44 \rangle$		p1	$\langle 45, 25 \rangle, \langle 83, 44 \rangle$
p2	$\langle 50, 88, 69, 57 \rangle$		p2	$\langle 50, 88 \rangle, \langle 69, 57 \rangle$
p3	$\langle 30, 7, 46, 60 \rangle$		p3	$\langle 30, 7 \rangle, \langle 46, 60 \rangle$
p4	$\langle 19, 3, 28, 58 \rangle$	$\Longrightarrow$	p4	$\langle 19, 3 \rangle, \langle 28, 58 \rangle$
p5	$\langle 84, 16, 50, 86 \rangle$		p5	$\langle 84, 16 \rangle, \langle 50, 86 \rangle$
p6	$\langle 55, 30, 53, 77 \rangle$		p6	$\langle 55, 30 \rangle, \langle 53, 77 \rangle$
p7	$\langle 39, 40, 20, 79 \rangle$		p7	$\langle 39, 40 \rangle, \langle 20, 79 \rangle$

Input:
(1) S: a set of d-dimensional objects
(2) d: the number attributes
Output: div: a set of divided-dimensional data
Algorithm:
(1) **IF** $d < 4$ **THEN**
(2) Construct a convex hull of S.
(3) **Break**
(4) Initialized a divided-dimensional dataset div_1 and div_2.
(5) **WHILE** $S \neq \{\}$ DO
(6) divide the dimension in half for the whole objects in S.
(7) save the divided objects in the div_1 and div_2.
(8) **RETURN** div.

Algorithm 1: The UB-Basic algorithm.

with divided dimension. In lines (1) to (3), algorithm checks the number of attributes d first. If d is less than 4, the UB-Basic algorithm is not performed. Because if the dimension is less than 4, it is impossible to divide the dimension, so algorithm constructs the convex hull in the next line and terminates the algorithm. If dimension d is greater than or equal to 4, algorithm initializes div_1 and div_2 to store the divided-dimensional data in line (4). Next, if S is not an empty set, algorithm divides the dimension in half in line (6) and saves the data of the two divided-dimensional datasets in the next line in div_1 and div_2, respectively. Finally, the dataset div of the divided dimension is returned and the algorithm terminates.

Table 3: An example of the UB-SelectAttribute algorithm.

ID	value $\langle X_1, X_2, X_3, X_4, X_5, X_6 \rangle$		id	value $\langle X_1, X_3 \rangle, \langle X_2, X_4, X_5, X_6 \rangle$
p1	$\langle 44, 30, 83, 44, 23, 45 \rangle$		p1	$\langle 44, 83 \rangle, \langle 30, 44, 23, 45 \rangle$
p2	$\langle 86, 66, 69, 57, 65, 50 \rangle$		p2	$\langle 86, 69 \rangle, \langle 66, 57, 65, 50 \rangle$
p3	$\langle 30, 7, 46, 60, 26, 79 \rangle$		p3	$\langle 30, 46 \rangle, \langle 7, 60, 26, 79 \rangle$
p4	$\langle 19, 3, 28, 58, 49, 20 \rangle$	$\Longrightarrow$	p4	$\langle 19, 28 \rangle, \langle 3, 58, 49, 20 \rangle$
p5	$\langle 72, 16, 50, 86, 7, 88 \rangle$		p5	$\langle 72, 50 \rangle, \langle 16, 86, 7, 88 \rangle$
p6	$\langle 55, 30, 53, 77, 30, 82 \rangle$		p6	$\langle 55, 30 \rangle, \langle 53, 77, 30, 82 \rangle$
p7	$\langle 39, 40, 20, 79, 36, 46 \rangle$		p7	$\langle 39, 40 \rangle, \langle 20, 79, 36, 46 \rangle$

4.1.2. UB-SelectAttribute Algorithm. In this section, we also propose UB-SelectAttribute algorithm which improves a division method of the UB-Basic algorithm. The UB-SelectAttribute algorithm determines the main attributes among the various attributes of the input data and divides the dimensions based on the main attributes. Table 3 shows an example of a UB-SelectAttribute algorithm. The input six-dimensional data has six attributes, X_1 to X_6. And if X_1 and X_3 are determined as main attributes, the proposed algorithm divides six-dimensional data into two-dimensional data with main attributes X_1 and X_3 and four-dimensional data with remained attributes. Representative studies to identify key attributes include Mean Decrease Impurity (MDI), which measures the influence of variables through an average of data impurity degradation during classification. In addition, it is possible to determine the main attribute by analyzing the main property through the histogram method.

Algorithm 2 shows the UB-SelectAttribute algorithm for dimension division based on main attributes. The input of the algorithm is S, which is a set of d-dimensional data objects, and d, which means the number of dimensions, and att, which is a set of main attributes. And the result of the

Input:
(1) S: a set of d-dimensional objects
(2) d: the number attributes
(3) att: a set of the major attributes
Output: div: a set of divided-dimensional data
Algorithm:
(1) **IF** $d < 4$ **THEN**
(2) Construct a convex hull of S.
(3) **Break**
(4) **IF** the number of att < 2 **THEN**
(5) Construct a convex hull of S.
(6) **Break**
(7) **IF** (d – the number of att) < 2 **THEN**
(8) Construct a convex hull of S.
(9) **Break**
(10) Initialized a divided-dimensional dataset div1 and div2.
(11) **WHILE** $S \neq \{\}$ **DO**
(12) divide the dimension into the major attributes and the remain attributes for the whole objects in S.
(13) save the divided objects in the div1 and div2.
(14) **RETURN** div.

Algorithm 2: The UB-SelectAttribute algorithm.

UB-SelectAttribute algorithm is div, which is a set of data with divided dimension. In lines (1) to (4), the algorithm checks the number of attributes d first the same as UB-Basic algorithm. If the dimension is less than 4, it is impossible to divide the dimension, so algorithm constructs the convex hull in the next line and terminates the algorithm. Next, UB-SelectAttribute algorithm checks the number of main attributes in line (4) and also checks the number of attributes except for the main attributes, and if the numbers are less than 2, the algorithm is not performed. This is because when constructing a convex hull, at least two attributes are needed. UB-SelectAttribute algorithm divides the input data into two divided-dimensional datasets based on the main attributes. In the two cases mentioned above, which are in lines (4) and (7), it is impossible to construct the convex hull in the divided dimension data. In line (9), initialize the resulting datasets div_1 and div_2 to store the divided-dimensional data. Next, if S is not an empty set, the algorithm divides the dimension in half in line (12) and saves the data of the two divided dimensions in the next line in div_1 and div_2, respectively. Finally, the dataset div of the partitioned dimension is returned and the algorithm terminates.

4.2. Constructing Divided-Convex Hull Step. In this section, we explain the second step of UB-Layer, the constructing divided-convex hull step. In the second step, we construct a divided-convex hull for the divided-dimensional data div, which is the result of the first step. Since each divided-dimensional data consists of at least two or more dimensional data, the convex hull could be constructed. Algorithm 3 shows the ConstructingDivided-Convexhull algorithm for generating divided-convex hull. First, the input of the algorithm is div, which is the divided-dimensional data. And the

Input: div: a set of divided-dimensional data
Output: Divided-CH: the list of local convex hull layer
Algorithm:
(1) **FOR** $i = 0$ **TO** 2 **DO**
(2) **WHILE** $\text{div}_i \neq \{\}$ **DO**
(3) Compute the local convex hull Divided-CH[i] over div_i.
(4) **RETURN** Divided-CH.

Algorithm 3: ConstructingDvidied-Convexhull algorithm.

output of the algorithm is a list of divided-convex hull. The ConstructingDivided-Convexhull algorithm constructs local convex hull as Divided-CH over each divided-dimensional data div_i in lines (1)–(3). Finally, the algorithm returns with a divided-convex hull list.

4.3. Combining Step. In this section, we explain the last step of UB-Layer, the combining step. We finally combine the divided-convex hulls to build UB-Layer. In the combining step, we build the UB-Layer with no overlapping tuples in each layer. The following example # explains a building process of the UB-Layer.

Example 2. Figure 4 shows an example of building the UB-Layer with the 4-dimensional dataset. Figure 4(a) shows the input 4-dimensional data, and Figure 4(b) shows the result of the dimension division step for input data. It is divided into 2-dimensional data. Figure 4(c) shows the result of constructing divided-convex hull in each of the divided-dimensional datasets div_1 and div_2. In div_1, the divided-convex hull consists of three layers, and the divided-convex hull in div has two layers. Figures 4(d)–4(f) show the process of constructing the final layer lists by combining the divided-convex hulls. For the combining step, we first construct the first layer of UB-Layer by removing duplicated data of the first layer of divided-convex hull in each divided-dimensional data, and UB-Layer[1] includes datasets p1, p2, p4, p5, and p7. Next, in Figures 4(e) and 4(f), we construct the second and third layer of UB-Layer, respectively, and build a final UB-Layer.

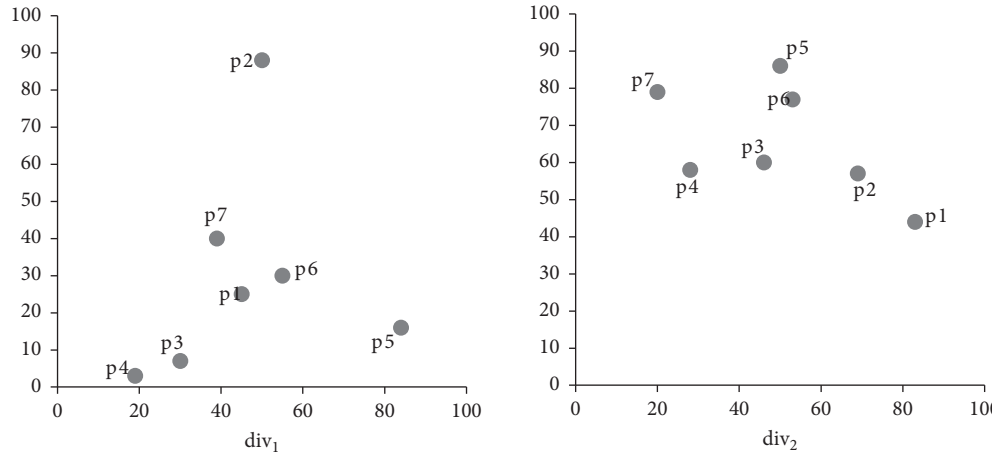
5. Performance Evaluation

In this section, we first explain the data and environment in Section 4.1 and then present the results of experiments in Section 4.2.

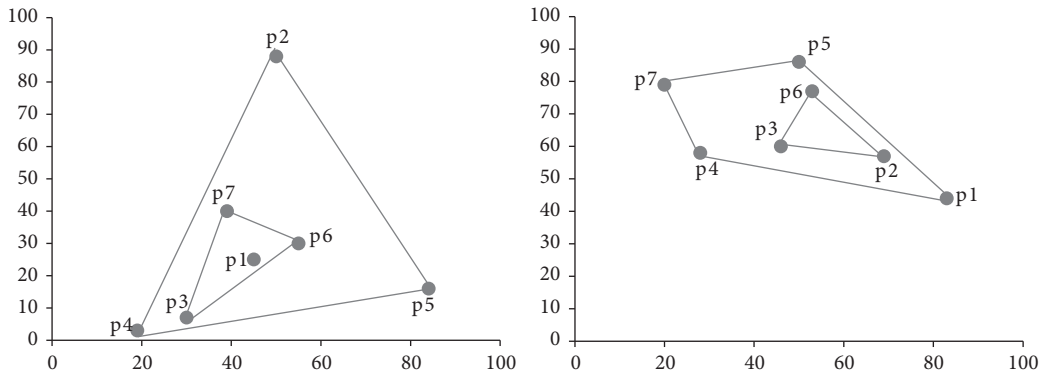
5.1. Experimental Data and Environment. In this experiment, we compare the computing time of index generation, the total number of layers, and accuracy of UB-Layer and convex hull [8]. The measure of the index generation time is wall clock time. We compare the number of data included in the layer to calculate accuracy. The equation of accuracy is shown as (1). NumOfData(CH) is the number of data included in the

id	value $\langle X_1, X_2, X_3, X_4 \rangle$
p1	⟨45, 25, 83, 44⟩
p2	⟨50, 88, 69, 57⟩
p3	⟨30, 7, 46, 60⟩
p4	⟨19, 3, 28, 58⟩
p5	⟨84, 16, 50, 86⟩
p6	⟨55, 30, 53, 77⟩
p7	⟨39, 40, 20, 79⟩

(a) An input 4-dimensional dataset



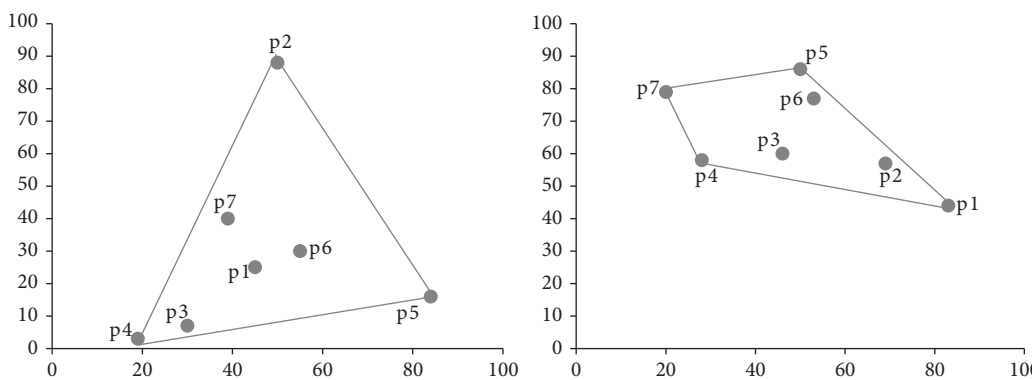
(b) The two divided-dimensional dataset



$\text{div}_1\text{-CH}[1] = \{p2, p4, p5\}$
$\text{div}_1\text{-CH}[2] = \{p3, p6, p7\}$
$\text{div}_1\text{-CH}[3] = \{p1\}$

$\text{div}_2\text{-CH}[1] = \{p1, p4, p5, p7\}$
$\text{div}_2\text{-CH}[2] = \{p2, p3, p6\}$

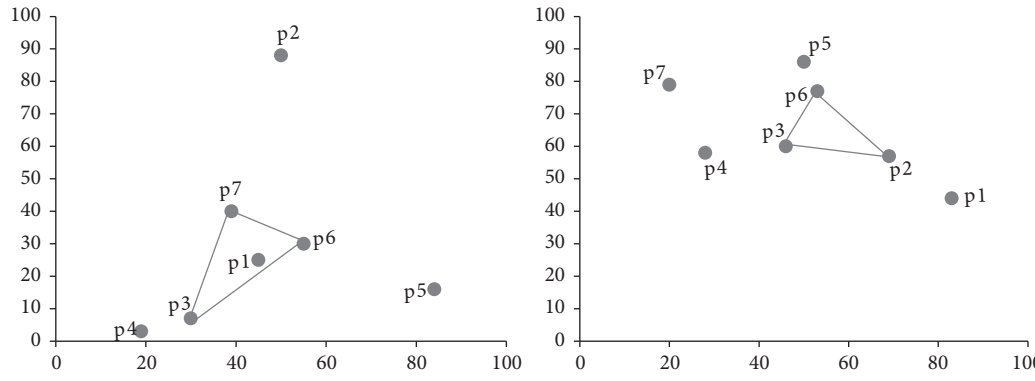
(c) The result of constructing divided-convex hull



$\text{div}_1\text{-CH}[1] = \{p2, p4, p5\} \cup \text{div}_2\text{-CH}[1] = \{p1, p4, p5, p7\}$
$= \textbf{UB-layer}[\textbf{1}] = \{\textbf{p1, p2, p4, p5, p7}\}$

(d) The result of constructing the first UB-Layer

Figure 4: Continued.

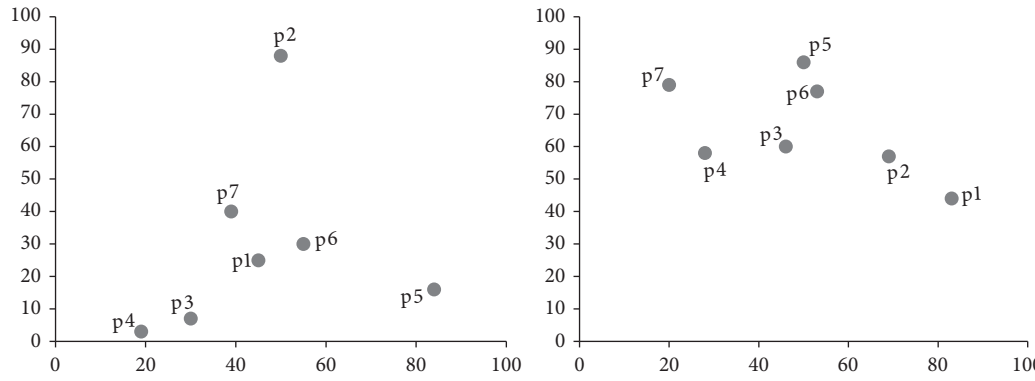


$\text{div}_1\text{-CH}[2] = \{p3, p6, p7\} \cup \text{div}_2\text{-CH}[2] = \{p2, p3, p6\}$

$- \text{UB-layer}[1] = \{p1, p2, p4, p5, p7\}$

$= \textbf{UB-layer}[\mathbf{2}] = \{\mathbf{p3, p6}\}$

(e) The result of constructing the second UB-Layer



$\text{div}_1\text{-CH}[3] = \{p1\} \cup \text{div}_2\text{-CH}[3] = \{\} - \begin{matrix}\text{UB-Layer}[1] = \{p1, p2, p4, p5, p7\} \\ \text{UB-Layer}[2] = \{p3, p6\}\end{matrix}$

$= \textbf{UB-Layer}[\mathbf{3}] = \{\}$

(f) The result of constructing the third UB-Layer

Figure 4: An example of building the UB-Layer.

first layer in convex hull. UB-Layer compares the number of data in the first layer and its similar data.

$$\text{Accuracy} = \frac{\text{NumOfData (CH)} - \text{NumofData (UB-Layer)}}{\text{NumOfData (CH)}} \times 100. \quad (1)$$

UB-Layer compares all 3-dimensional division methods. Each UB-Basic uses UB-Basic algorithm, UB-SA uses UB-SelectAttribute algorithm to construct UB-Layer. Through the experiments with synthetic data, we use input data from data generator from PL-Index [12]. Data size N goes to 10K and data dimension was converted from 2-dimensional to 8-dimensional. For the experiment, we construct UB-Layer and convex hull by C++. First, the algorithm generates the first layer and compares its accuracy rate with index generation time in order to compare the number of data to calculate accuracy rate. We conducted all the experiments on an Intel i5-760 quad core processor running at 2.80 GHz Linux PC with 16 GB of main memory.

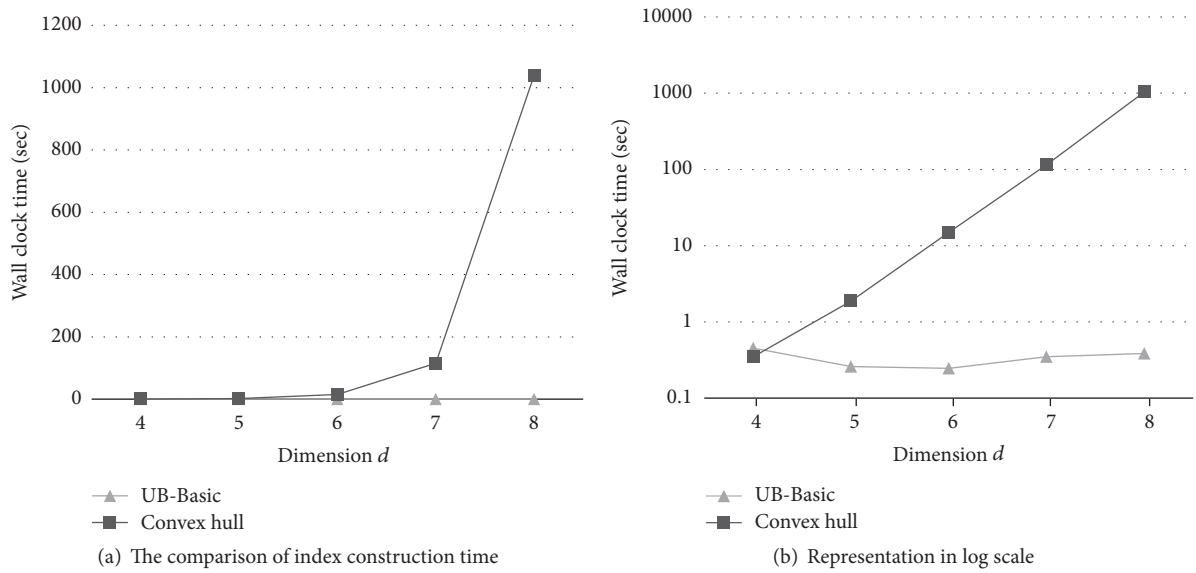
Table 4 summarizes experiments and variables in order to measure the index generation time and precision. The variables for the experiment are data size N and dimension d.

5.2. Result of Experiments. In this section, we show the experiments results of the index construction time and accuracy of the convex hull, UB-Layer, and UB-Basic which is the basic dimension division algorithm of UB-Layer and UB-SelectAttribute (UB-SA) that divides input data based on the main attributes.

Experiment 1 (the comparison of index construction time as dimension d is varied (N = 10K)). Figure 5 shows the construction time as a wall clock time of UB-Layer and convex hull when dimension d is varied from 4 to 8. As the dimension increases, the index construction time of convex hull shows an exponential increase. However, the

Table 4: UB-Layer experiment environment.

Experiment	Variable	
	Dataset	UNIFORM
Experiment 1 (the comparison of index construction time as dimension d is varied (N = 10K))	N	10K
	d	4, 5, 6, 7, 8
Experiment 2 (the comparison of the number of total layers as dimension d is varied (N = 10K))	N	10K
	d	4, 5, 6, 7, 8
Experiment 3 (the comparison of index construction time of UB-Layer methods as dimension d is varied (N = 10K, d = 4 ~ 12))	N	10K
	d	4, 5, . . . , 12
Experiment 4 (the comparison of the number of total layers of UB-Layer methods as dimension d is varied (N = 10K, d = 4 ~ 12))	N	10K
	d	4, 5, . . . , 12
Experiment 5 (the comparison of index construction time of total methods as dimension d is varied (N = 10K, d = 6 ~ 8))	N	10K
	d	6, 7, 8
Experiment 6 (the comparison of the number of total layers of total methods as dimension d is varied (N = 10K, d = 6 ~ 8))	N	10K
	d	6, 7, 8
Experiment 7 (the comparison of accuracy of UB-Layer methods as dimension d is varied (N = 10K, d = 6 ~ 8))	N	10K
	d	6, 7, 8



(a) The comparison of index construction time

(b) Representation in log scale

Figure 5: The comparison of index construction time as d is varied.

index construction time of UB-Layer increases in log scale. The index construction time of UB-Layer is improved by 0.74 to 99.35 times compared to the convex hull. The construction speed is slower than the convex hull when dimension is 4, because of the dimension dividing and merging cost. However, as dimension increases more than 5, the difference becomes larger as Figure 5(b) shows.

Experiment 2 (the comparison of the number of total layers as dimension d is varied (N = 10K)). Figure 6 shows the number of total layers of UB-Layer and convex hull as dimension d is varied from 4 to 8. Convex hull constructs 11 layers for 10K input data on average; that is, one layer includes about 909 data. The number of total layers of UB-Layer is improved by 4.2 to 11 times compared to the convex hull; that is, one layer of UB-Layer includes about 5 to 11 times less data. Therefore, UB-Layer is more efficient in query processing because the number of data in one layer of UB-Layer is less than convex hull.

Experiment 3 (the comparison of index construction time of UB-Layer as dimension d is varied (N = 10K, d = 4~12)). Figure 7 shows the comparison of the index construction time between proposed methods as dimension d is varied. We have presented UB-Basic and UB-SA for dividing dimension of the input data in Section 4. In Figure 7, UB-SA$\langle 4, x\rangle$ means that the number of the main attribute is 4, and UB-SA$\langle 5, x\rangle$ means that the main attribute is 5. The index

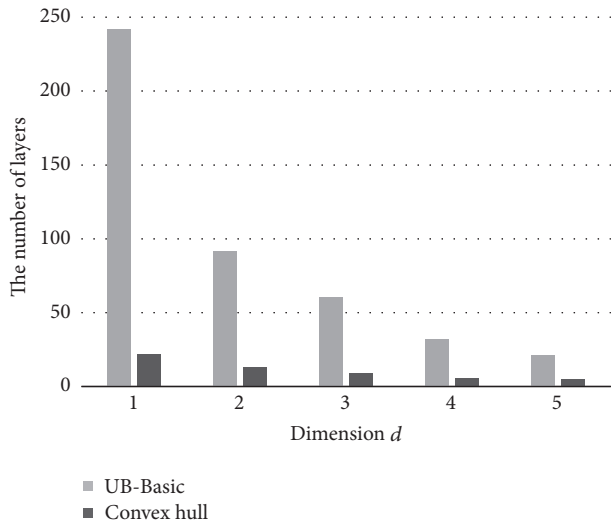
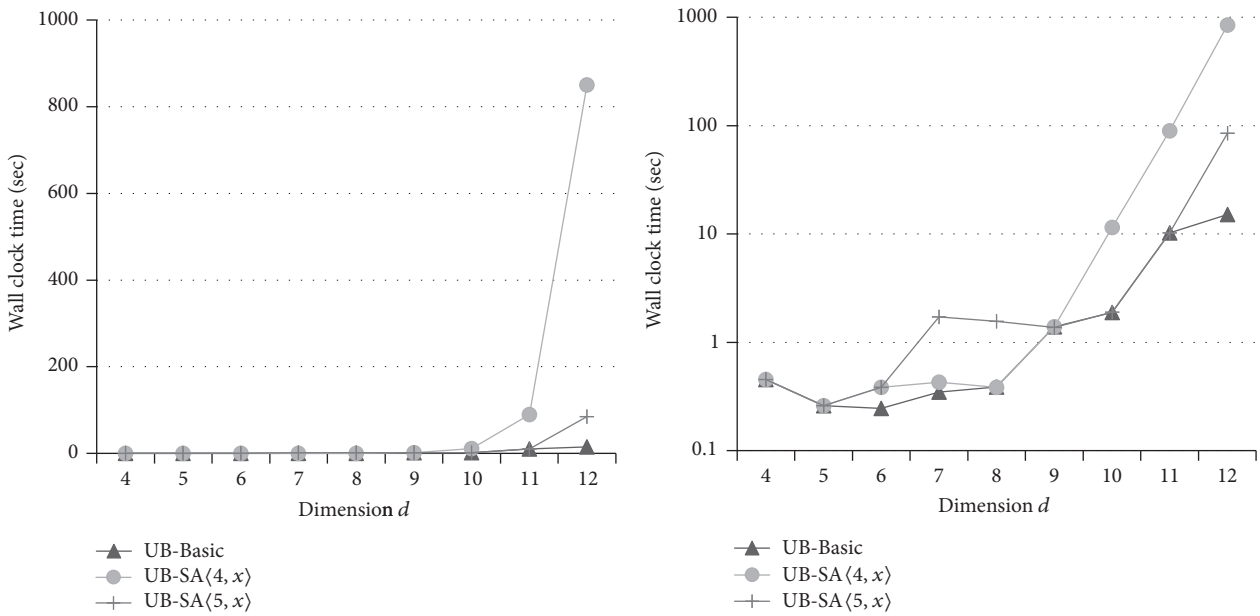


Figure 6: The comparison of the number of total layers as d is varied.



(a) The comparison of index construction time

(b) Representation in log scale

Figure 7: The comparison of index construction time of UB-Layer methods as d is varied.

construction time of UB-SA shows constant increase when the total number of attributes increases. Nevertheless, UB-SA builds indices faster than UB-Basic algorithm.

Experiment 4 (the comparison of the number of total layers of UB-Layer methods as dimension d is varied (N = 10K, d = 4~12)). Figure 8 shows the comparison of the number of total layers of two UB-Layer construction methods as dimension d is varied from 4 to 12. In the case of UB-Basic and UB-SA algorithm, the number of the layers decreases when the amount of data attribute increases. Since the algorithm reads each layer to retrieve result value in top-k query processing, the algorithms perform more efficient query processing as the number of the total layer increases.

Experiment 5 (the comparison of index construction time of total methods as dimension d is varied (N = 10K, d = 6~8)). Figure 9 shows the comparison of index construction time of proposed UB-Layer construction methods and convex hull as dimension d is varied from 6 to 8. The experiment was based on 6 to 8 dimensions because the convex hull experiment with more than 9 dimensions is impossible. We prove that the proposed methods generate indices about 70 to 396 times faster than the convex hull algorithm.

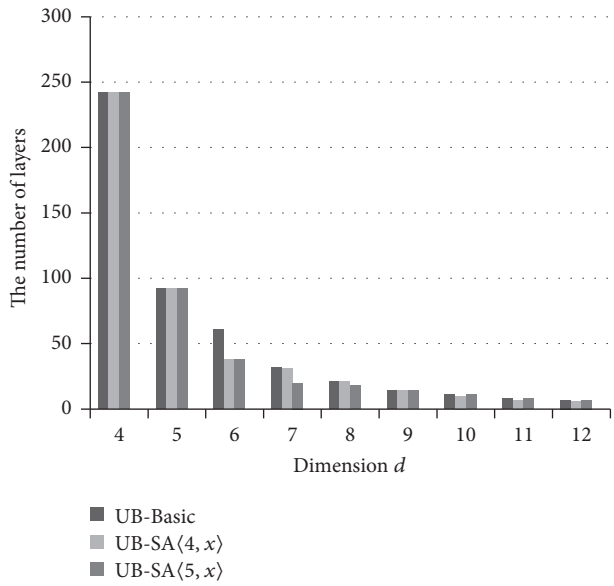
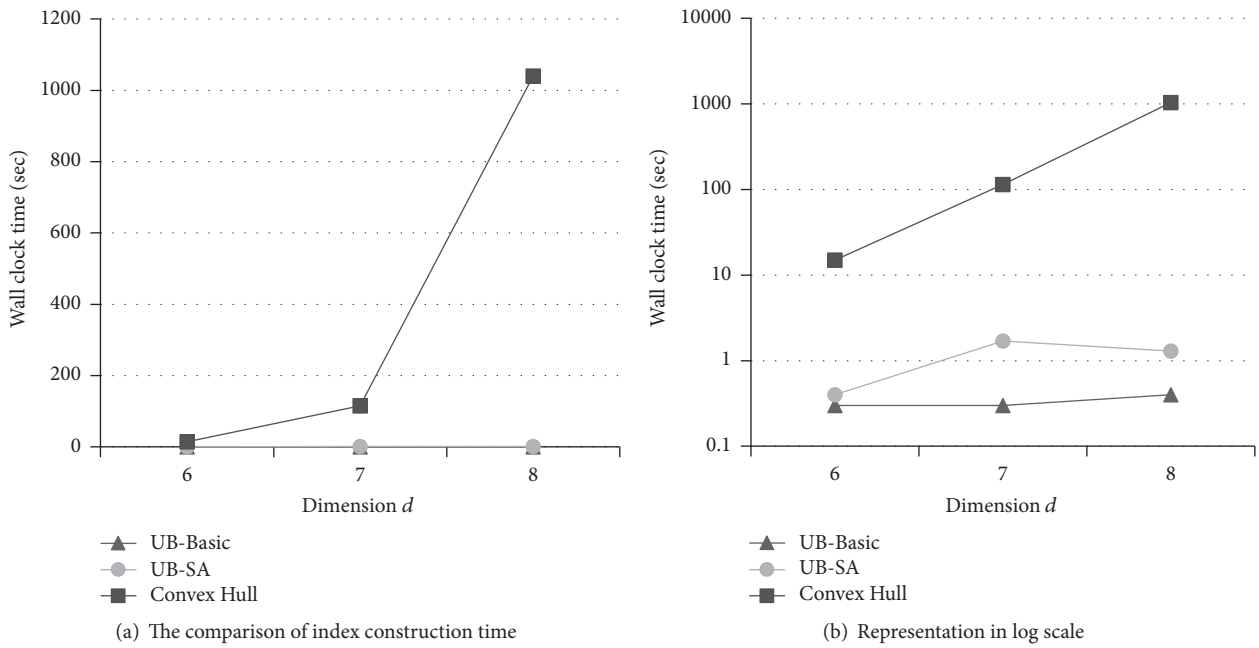


Figure 8: The comparison of the number of total layers of UB-Layer methods as dimension d is varied.



(a) The comparison of index construction time

(b) Representation in log scale

Figure 9: The comparison of index construction time of total methods as d is varied.

Experiment 6 (the comparison of the number of total layers of total methods as dimension d is varied (N = 10K, d = 6~8)). Figure 10 shows the comparison of the number of total layers of two UB-Layer construction methods and convex hull as dimension d is varied from 6 to 8. Likewise, we perform the experiment by 6 to 8 dimensions because the convex hull experiment with more than 9 dimensions is impossible. The proposed methods build more layers by 3.6 to 27 times than the convex hull algorithm. Top-k query processing performs by reading the layers one by one. Therefore, UB-Layer is more efficient to query processing, because a large number of total layers means that the number of data to be read in the query processing is small.

Experiment 7 (the comparison of accuracy of UB-Layer methods as dimension d is varied (N = 10K, d = 6~8)). Figure 11 shows the comparison of the accuracy of two UB-Layer construction methods as dimension d is varied from 6 to 8. The accuracy becomes 100% if all input data from first layer in constructed convex hull is included. The proposed methods show 50% of accuracy on average. However, the accuracy of proposed methods becomes higher when it

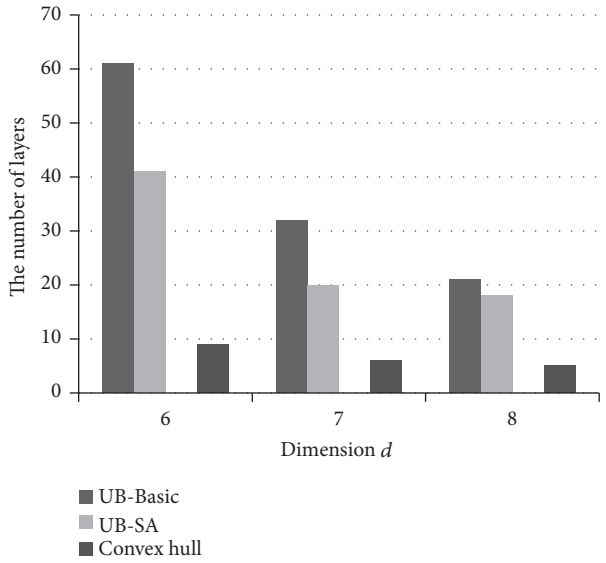


Figure 10: The comparison of the number of total layers of total methods as d is varied.

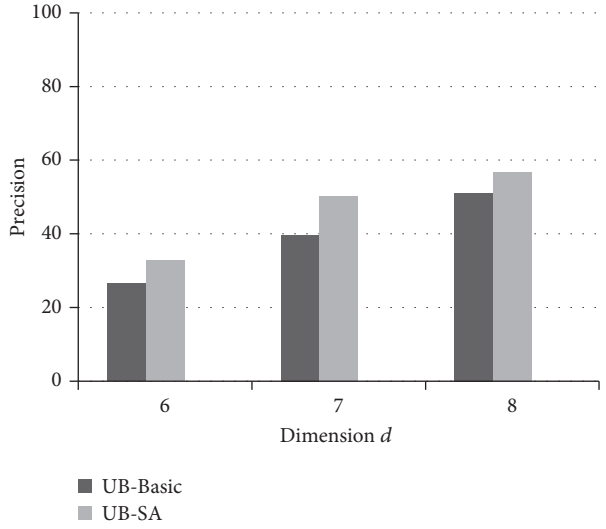


Figure 11: The comparison of accuracy of UB-Layer methods as d is varied.

comes to increase its dimension. Therefore, the proposed methods provide accurate results in high-dimensional data.

6. Conclusion

In this paper, we have proposed the UB-Layer that significantly reduces the index building time and increases the number of layers of the convex hull. The proposed method first divides the dimension of input data and constructed divided-convex hull in each divided-dimensional data. And then, it combines the divided-convex hull to build a final UB-Layer.

We have performed experiments on synthetic datasets with varying the data size and the dimension. Experimental results show the proposed method builds an index fast over high-dimensional data, whereas the convex hull could not have constructed. And UB-Layer is also more efficient for top-k query processing because it has more layers than convex hull.

However, the proposed method has some limitations. First, the optimized query processing algorithm should be studied, because UB-Layer is a method to construct an index. Second, the index construction time is greatly reduced, but some of the correct answer data is missing when it is compared to the convex hull.

As for the future work, we will study algorithms to solve these limitations of our method. We will first improve the index building time of the proposed method by dividing dimension hierarchically. Second, we will improve the precision of the UB-Layer. Third, we will analyze the time complexity of our method and compare to existing methods. Moreover, we will study about efficient top-k query processing method with UB-Layer.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean Government (MSIT) (no. R7120-17-1007, SIAT CCTV Cloud Platform).

References

[1] R. Asija and R. Nallusamy, "Healthcare SaaS based on a data model with built-in security and privacy," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 6, no. 3, pp. 1–14, 2016.

[2] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 337–348, 2009.

[3] P. Haghani, S. Michel, and K. Aberer, "The gist of everything new: personalized top-k processing over web 2.0 streams," in *Proceedings of the 19th International Conference on Information and Knowledge Management (CIKM' 10)*, pp. 489–498, October 2010.

[4] H. Xiong, M. Brodie, and S. Ma, "TOP-COP: mining TOP-K strongly correlated pairs in large databases," in *Proceedings of the 6th International Conference on Data Mining (ICDM '06)*, pp. 1162–1166, December 2006.

[5] M. A. AlZain, A. S. Li, B. Soh, and E. Pardede, "Multi-cloud data management using Shamir's Secret Sharing and Quantum Byzantine Agreement schemes," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 5, no. 3, pp. 35–52, 2015.

[6] S. Bagui and L. T. Nguyen, "Database sharding: to provide fault tolerance and scalability of big data on the cloud," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 5, no. 2, pp. 36–52, 2015.

[7] J. Yu, P. Lu, Y. Zhu, G. Xue, and M. Li, "Toward secure multikeyword top-k retrieval over encrypted cloud data," *IEEE*

Transactions on Dependable and Secure Computing, vol. 10, no. 4, pp. 239–250, 2013.

[8] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[9] G. H. Forman and J. Zahorjan, "The challenges of mobile computing," *Computer*, vol. 27, no. 4, pp. 38–47, 1994.

[10] K. Hirpara and K. Rana, "Energy-efficient constant gain Kalman filter based tracking in wireless sensor network," *Wireless Communications and Mobile Computing*, vol. 2017, Article ID 1390847, 7 pages, 2017.

[11] Y. C. Chang, L. Bergman, V. Castelli, C. S. Li, M. L. Lo, and J. R. Smith, "The onion technique: indexing for linear optimization queries," in *Proceedings of the International Conference on Management of Data (SIGMOD '00)*, pp. 391–402, 2000.

[12] J.-S. Heo, J. Cho, and K.-Y. Whang, "The hybrid-layer index: a synergic approach to answering top-k queries in arbitrary subspaces," in *Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE '10)*, pp. 445–448, March 2010.

[13] J.-S. Heo, J. Cho, and K.-Y. Whang, "Subspace top-k query processing using the hybrid-layer index with a tight bound," *Data & Knowledge Engineering*, vol. 83, pp. 1–19, 2013.

[14] S.-Y. Ihm, A. Nasridinov, and Y.-H. Park, "An efficient index building algorithm for selection of aggregator nodes in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2014, Article ID 520428, 8 pages, 2014.

[15] C. Böhm and H. P. Kriegel, "Determining the convex hull in large multidimensional databases," in *Proceedings of the Data Warehousing and Knowledge Discovery (DaWak)*, vol. 2114 of *Lecture Notes in Computer Science*, pp. 294–306, 2001.

[16] M.-K. Lee, "A new convex-hull based approach to evaluating flatness tolerance," *Computer-Aided Design*, vol. 29, no. 12, pp. 861–868, 1997.

[17] R. Liu, B. Fang, Y. Y. Tang, J. Wen, and J. Qian, "A fast convex hull algorithm with maximum inscribed circle affine transformation," *Neurocomputing*, vol. 77, no. 1, pp. 212–221, 2012.

[18] M. Tang, J.-Y. Zhao, R.-F. Tong, and D. Manocha, "GPU accelerated convex hull computation," *Computers and Graphics*, vol. 36, no. 5, pp. 498–506, 2012.

[19] A. Stein, E. Geva, and J. El-Sana, "CudaHull: fast parallel 3D convex hull on the GPU," *Computers and Graphics*, vol. 36, no. 4, pp. 265–271, 2012.

[20] M. Gao, T.-T. Cao, A. Nanjappa, T.-S. Tan, and Z. Huang, "gHull: a GPU algorithm for 3D convex hull," *ACM Transactions on Mathematical Software (TOMS)*, vol. 40, no. 1, 2013.

[21] N. Singh, R. Arya, and R. K. Agrawal, "A convex hull approach in conjunction with Gaussian mixture model for salient object detection," *Digital Signal Processing*, vol. 55, pp. 22–31, 2016.

[22] S. Ding, X. Nie, H. Qiao, and B. Zhang, "A fast algorithm of convex hull vertices selection for online classification," *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[23] P. Jiang, S. Liu, J. Liu, F. Wu, and L. Zhang, "A depth-adjustment deployment algorithm based on two-dimensional convex hull and spanning tree for underwaterwireless sensor networks," *Sensors*, vol. 16, no. 7, article 1087, pp. 1–20, 2016.

[24] A. Kundu, R. Misra, A. Kar et al., "On demand secure routing protocol using Convex-Hull & K-mean approach in MANET," in *Proceedings of the 7th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON '16)*, October 2016.

[25] R. Xu, H. Dai, F. Wang, and Z. Jia, "A convex hull based optimization to reduce the data delivery latency of the mobile elements in wireless sensor networks," in *Proceedings of the 15th IEEE International Conference on High Performance Computing and Communications and the 11th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pp. 2245–2252, November 2013.