UB-H: an unbalanced-hierarchical layer binary-wise construction method for high-dimensional data

Sun-Young Ihm¹ · So-Hyun Park² · Young-Ho Park³

Received: 12 May 2020 / Accepted: 12 November 2020 / Published online: 6 January 2021
© The Author(s) 2021

Abstract

Cloud computing, which is distributed, stored and managed, is drawing attention as data generation and storage volumes increase. In addition, research on green computing, which increases energy efficiency, is also widely studied. An index is constructed to retrieve huge dataset efficiently, and the layer-based indexing methods are widely used for efficient query processing. These methods construct a list of layers, so that only one layer is required for information retrieval instead of the entire dataset. The existing layer-based methods construct the layers using a convex hull algorithm. However, the execution time of this method is very high, especially in large, high-dimensional datasets. Furthermore, if the total number of layers increases, the query processing time also increases, resulting in efficient, but slow, query processing. In this paper, we propose an unbalanced-hierarchical layer method, which hierarchically divides the dimensions of input data to increase the total number of layers and reduce the index building time. We demonstrate that the proposed procedure significantly increases the total number of layers and reduces the index building time, compared to existing methods through the various experiments.

Keywords Green computing · Cloud computing · Convex hull · Unbalanced layer

Mathematics Subject Classification 68U35

✉ Young-Ho Park
yhpark@sm.ac.kr

Sun-Young Ihm
sunnyihm@dgu.ac.kr

So-Hyun Park
shpark@sm.ac.kr

¹ Development of Smart Community Policing System Research Center, Dongguk University, Jung-gu, Seoul, 04620, Republic of Korea

² Bigdata Using Research Center, Sookmyung Women's University, Seoul, 04310, Republic of Korea

³ Department of IT Engineering, Sookmyung Women's University, Seoul, 04310, Republic of Korea

1 Introduction

Cloud computing has recently received increased attention from both research communities and IT industries, especially for large-scale data management systems as the number of services is large and is also increasing fast [1–3]. Cloud computing is virtually distributed computing, consisting of a data server with data providers and customers [4]. There are several studies focusing on efficiently processing large amounts of data in cloud environments [5–7], as well as studies investigating green computing, an area analyzing efficient use of computer resources. Murugesan [8] defined green computing as “the research and practice that efficiently and effectively design, manufacture, use and dispose of computers, servers, and related subsystems and communications systems with minimal or no environmental impact.” That is, to achieve an algorithmic efficiency is the main purpose of green computing, alongside improving the energy efficiency to enhance the quality of service [9, 10]. Because the cloud environment has a large amount of data transmission, the need to tackle green computing is even more urgent. Ihm et al. [11] has defined three requirements for designing a suitable algorithm: (1) the algorithm must consider using of limited computer resources, such as time and memory, (2) the algorithm must deal with data whose characteristics and distribution change over time, and (3) the designed algorithm has to provide computer resources in an energy-efficient and economical way.

Top-k query processing can be used to efficiently retrieve a large amount of data stored in the cloud by returning k items that match the user’s needs. To quickly obtain the query processing result, the indexes must be created in the form of a convex hull. A convex hull is a set of boundary points (the minimal outermost points that contain all the points of a given dataset in a d -dimensional space); a well-studied object in computational geometry. Convex hull computation is widely used in shape analysis, pattern recognition, collision detection, top-k query processing, machine learning, and more [12].

Figure 1 shows a basic example of cloud computing. Data owners upload their data to the cloud and customers download the data uploaded by data owners.

Fig. 1 A simple example of cloud computing. Data owners upload their data to the cloud, while clients download this data from the cloud

When clients access the uploaded data, top-k query processing can be used to efficiently retrieve the data.

Motivating Example: Consider a client who wants to buy a used car. Used cars have various attributes such as price, manufacturer, model, mileage, grade, fuel, color and transmission. Companies that own used cars upload used car information to the cloud. The client wants to search for a used car by searching for and comparing two used cars as candidate cars. Among the car attributes listed, the client wants to search by the mileage and price to find a used car with a low mileage and a low price. In general, a large amounts of data are uploaded to the cloud, but in this example, it is assumed that a total of 16 used cars are uploaded to the cloud. Figure 2a shows the result of the convex hull obtained by mapping 16 used cars in two dimensions, for the price and mileage attributes. A total of three layers were created, with seven used cars {a, b, f, h, j, o, p} in the first layer, five used cars {c, d, g, k, n} in the second layer, and four used cars {e, i, l, m} in the third layer. Because the client wants to consider two used cars as candidates, top-k query processing retrieves two used cars that match the client's criteria in the first and second layers. That is, two candidate cars were retrieved for the client from 12 prospective used cars, among the 16 used cars in total.

However, when the number of training dataset is large and their dimension is high, the process of convex hull computation is time-consuming [13]. There have been various studies published, which aim to reduce the convex hull computation time [14–16]. Many existing studies have focused on reducing the computation time for constructing accurate convex hulls; however, these still do not resolve time complexity issues. Furthermore, some recent studies have proposed significantly reducing the computation time by constructing an approximate convex hull. The unbalanced (UB)-layer [14] method is the result of one of the latest studies in constructing approximate convex hulls that is applicable to higher-dimensional and larger databases. Figure 2 shows two lists of layers constructed

(a) A list of layers constructed by convex hull computation **(b)** A list of layers constructed by UB-Layer

Fig. 2 Comparison between lists of layers built using a convex hull procedure and UB-Layer

using (a) a balanced convex hull procedure and (b) UB-Layer, from the motivating example. A list of layers in the balanced form is shown in Fig. 2a. For comparison, a list of layers in unbalanced form is shown in Fig. 2b. The UB-Layer constructs an approximate layer to the convex hull; however, it constructs more layers. In the motivating example, the convex hull procedure constructs three layers, whilst the UB-Layer constructs four layers from the same dataset. In other words, if we consider two layers as in the motivating example, the top two results can be obtained by searching only 9 out of 16 items, via the UB-Layer.

However, the UB-Layer only constructs layer lists as an approximate form to the convex hull; the first layer of the UB-Layer does not always completely contain the other layers. Therefore, the data that users want may not be listed in the correct layer. However, in recent years the amount of data being processed has become so large that users prefer approximate and fast results, rather than perfectly accurate and slower results in some applications. When searching for used cars, as in the example, the user wants results that are close to their requirements, but closer to their requirements than the slow results. In particular, the speed of retrieval is even more important in a cloud computing environment because the movement time of the data must also be considered.

The number of layers is also important, as mentioned in the example when constructing layer-based indexes for high-dimensional data. Because data is retrieved by layers at the query, a large amount of data in a single layer means that the query processing time is high. UB-Layer reduces the computation time compared to the convex hull procedure; however, for large and high-dimensional databases, it still suffers from a small number of layers and a long index building time.

In this paper, we propose a hierarchical UB-layer method, called UB-Hierarchical (UB-H), which reduces the index building time and increases the number of layers of UB-Layer. The contributions of this paper are summarized as follows:

- We propose a method called UB-H to divide the dimensions of a dataset by a hierarchical method, improving upon the computation time of UB-Layer. UB-H divides the dataset's dimensions until it has the smallest possible dimensions required to compute the convex hull. Further, we construct an index with a greater total number of layers for efficient query processing.
- We show the performance advantages of the proposed UB-H through various experiments. We compare the index building time, the total number of layers, and accuracy of UB-H with previous methods (UB-Layer and the convex hull method).

The remainder of this paper is organized as follows. We describe the existing work relevant to this study in the Sect. 2. We formally define the problem being addressed in the Sect. 3, and present the proposed method in the Sect. 4. We describe the experimental results which compare the proposed method with previous methods in the Sect. 5. We summarize and conclude the paper in the Sect. 6.

2 Related studies

In this section, we explain convex hull computation methods and discuss existing work. We first describe the convex hull method in Sect. 2.1 and UB-Layer methods in Sect. 2.2. We also describe related studies that use convex hull computation for cloud and green computing in Sect. 2.3.

2.1 Convex hull methods

The typical methods for convex hull computation relevant to the method presented here are the onion technique [17], hybrid-layer (HL)-Index [18, 19], approximate convex hull (aCH)-Index [20]. The convex hull comprises a structure that encloses other data objects. The onion technique builds a primary layer from the input dataset by finding edge objects. It then builds a secondary layer based on the remaining data in the same way, building the remaining layers sequentially. The HL-Index is a combination of layer-based and list-based index construction methods designed to improve the onion method. In the aCH-Index, an approximate method is proposed that first creates a skyline layer, then divides the input dataset using a grid-partitioning algorithm with virtual points. The aCH-Index is constructed by combining each convex hull from the partitioned datasets.

There are also further methods that improve convex hull computation, including GPU processing, R-trees, and approximate methods. CudaHull [12] is a parallel algorithm, which uses CUDA programming model, for calculating the convex hull of a set of points in 3D. A randomized approximate convex hull algorithm has also been proposed [15], which has an acceptable execution time to compute the convex hull for high-dimensional data. Liu et al. [21] proposed the visual-attention-imitation convex hull algorithm; a fast convex hull algorithm using information on the extreme points of a point set. Moreover, Ramli et al. [15] proposed a real-time fuzzy regression analysis method based on the convex hull algorithm.

The convex hull has also been applied in multiple studies and fields. The algorithm SPHERE [22] used a convex hull for the k-regret query to attain the lower bound of the maximum regret ratio. Meanwhile, Peng et al. constructed a convex hull to find candidate points for the k-regret query [23]. Mouratidis and Tang [24] introduced an uncertain top-k query to report all options when uncertain preferences are given, that applies directly to a general convex hull.

2.2 UB-Layer method

The existing convex hull methods have the advantage of being able to perform query processing in all directions; however, they suffer from a very long index construction time. The existing methods focus on reducing the execution time and not increasing the number of layers. UB-Layer [14] is an unbalanced layer-based indexing method that reduces the construction time of the convex hull. The outer layer of the UB-Layer does not enclose other data objects as shown in Fig. 2b.

UB-Layer constructed by dividing the input dataset into sub dataset with divided-dimension first. Next, the algorithm creates divided-convex hull based on the sub dataset which has divided-dimension and builds UB-Layer by combining each divided-convex hull. Its index construction time is 0.74–99 times that of the convex hull, and its average precision is 50%. However, the number of layers is still too small for efficient querying.

2.3 Convex hull methods in cloud and green computing

Recently, cloud computing has added a new dimension to the traditional means of computation, data storage, and service applications [25, 26]. However, the enormous worldwide computing levels have a direct impact on the environment, so numerous studies have been conducted to reduce this negative impact [27]. Improvements in performance involving the disk input/output, CPU, and memory reduction can also reduce overall energy usage. Green computing is a study area that covers the whole computing lifecycle, with current green computing trends focusing on efficient utilization of resources [28]. For example, Cao et al. [29] use a convex hull as a selection method to save energy and improve computing performance in parallel computational biology applications. And the near convex hull is proposed in [32] which is quickly formed by merely determining several special locations and has much lower computational complexity. A new filtering method for the convex hull in two dimensions is proposed for accelerating the computation of the convex hull [33].

3 Problem definition

In this section, we formally define the problem of layer-based index construction methods. In this study, we construct the index as a list of layers, enabling efficient queries. An input dataset (DS) has n data objects with d real-value attributes, $A_1, A_2, \dots, A_d$. Every object in the DS can be considered as a point in d -dimensional space. Table 1 summarizes the notation used throughout this paper; the symbols that have not yet been introduced will be explained in Sect. 4.

Table 1 The notation

Symbols	Definitions
DS	An input dataset
O	A data object in DS
N	The number of data objects in DS
D	The number of attributes of O (the dimension of DS)
A_i	The i th attribute of DS ($1 \leq i \leq d$)
$sub-DS$	The partitioned dataset with divided dimensions

4 UB-H: Unbalanced hierarchical layer

In this section, we propose an approximate layer-based index building method for high-dimensional and large databases, called UB-H. The computation time of the convex hull increases rapidly when the dimensions of the input dataset increase. UB-H is a method that minimizes the index construction time, by hierarchically dividing the dimensions of the input dataset. In Sect. 4.1, we give an overview of UB-H, and then proceed to explain each of its steps in detail in Sects. 4.2, 4.3, and 4.4.

4.1 Overview

UB-H is constructed by following three steps: (1) hierarchically dividing the dimensions, (2) building the sub-convex hull, and (3) UB-layering. First, we divide the dimensions of the DS into k sub-datasets (*sub-DSs*) ($1 \leq k \leq d/2$). The proposed method divides the attributes hierarchically until there are two or three divided attributes to maximally reduce the execution time. Next, we build m sub-convex hulls (*sub-CHs*) in each *sub-DS*, and finally combine the *sub-CHs* (whilst removing duplicate objects) to construct the UB-H layer index.

4.2 Hierarchical dimension division step

In this section, we explain the first step of the proposed method: hierarchical dimension division. Figure 3 shows an example of a hierarchical dimension dividing step that divides an eight-dimensional dataset into four two-dimensional sub-datasets. Figure 3a shows a DS , which consists of seven objects with eight attributes each. The result of the first dividing phase is shown in Fig. 3b. DS is divided into two sub-datasets, $sub-DS_1$ and $sub-DS_2$, with four attributes each. We divide the attributes based on a UB-SelectAttribute algorithm [14] and consider the main attributes. In this paper, we assume that the attribute weights are the same, for simplicity. Figure 3c, d show the results of the second and third dividing phases, which hierarchically divide the attributes of $sub-DS_1$ and $sub-DS_2$. Thus, the proposed method hierarchically partitions the dimensions into subsets of only two or three dimensions.

Table 2 shows the Hier-dividing algorithm for hierarchical dimension partitioning. The inputs of the algorithm are DS , a set of d -dimensional data objects, with d denoting the number of attributes. The result of the Hier-dividing algorithm is *sub-DSs*, which are sets of data objects with partitioned dimensions. First, the number of dimension k to be divided into is obtained in line 1, and k *sub-DSs* are produced in lines 2–3. Next, if the DS is not an empty set, it is split in line 4 until there are two dimensions, and the data of each divided object in $sub-DS_i$ are saved in the next line. Finally, the *sub-DS*, which is a dimensionally divided dataset, is returned and the algorithm ends. An example of the result of the hierarchical division step is shown

Fig. 3 An example of the dimension hierarchical division step on a dataset of seven objects, each with eight attributes. After three dimension hierarchical division steps, the objects are split into subsets of two attributes

Table 2 The Hier-dividing algorithm for dimension partitioning

Algorithm: Hier-dividing

Input:

- (1) DS : a set of d -dimensional objects
- (2) d : the number attributes

Output:

- (1) $sub-DS$ s: a set of dimensionally divided data

Algorithm:

1. Compute the total number of divided dimension $k = d / 2$
 2. **FOR** $i = 0$ **TO** $k-1$ **DO**
 3. Initialize a dimension-divided dataset $sub-DS_i$
 4. **WHILE** $DS \neq \{\}$ **DO**
 5. Divide the dimension until $d=2$, for all objects in DS
 6. Save the divided objects in $sub-DS_i$. $1 \leq i < k$ $!$
 7. **RETURN** $sub-DS$
-

Fig. 4 An example of the result of dimension hierarchical division step, expressing Fig. 3d in 2D coordinates

in Fig. 4, with sub-DS_1 , sub-DS_2 , sub-DS_3 , and sub-DS_4 of Fig. 3d expressed in two-dimensional coordinates in Fig. 4.

4.3 Building a sub-convex hull step

In this section, we explain the second step of the proposed method: the building of the *sub-CH*. In this step, we build the *sub-CHs* in each *sub-DS* resulting from the aforementioned Hier-dividing algorithm. It is possible to compute a convex hull in each *sub-DS* because the dimensions of every *sub-DS* are either two or three. The result of building a *sub-CH* is shown in Fig. 5. The objects O_1 , O_4 , O_5 , and O_7 are computed as $\text{sub-CH}_1[1]$, and objects O_2 , O_3 , and O_6 are computed as $\text{sub-CH}_1[2]$ from sub-DS_1 . In addition, the objects O_1 , O_4 , O_5 , and O_7 are computed as $\text{sub-CH}_2[1]$ and objects O_2 , O_3 , and O_6 are computed as $\text{sub-CH}_2[2]$ from sub-DS_2 . The objects O_1 , O_4 , O_5 , and O_7 are computed as $\text{sub-CH}_3[1]$, and objects O_2 , O_3 , and O_6 are computed as $\text{sub-CH}_3[2]$ from sub-DS_3 . Objects O_1 , O_4 , and O_7 are computed

Fig. 5 Example of the building sub-convex hull step, from the data in Fig. 4

as *sub-CH₄[1]*, objects O₂, O₃, and O₆ are computed as *sub-CH₄[2]*, and O₅ is computed as *sub-CH₄[2]* from *sub-DS₄*.

4.4 UB-layering step

In this section, we explain the last step of the proposed method, the UB-layering step. Here, we construct the UB-H layer by combining the sub-CHs from each sub-DS, while removing duplicate objects. The proposed method produces the sub-DS by dividing the dimension of the data objects in DS; thus, duplicates can occur in each sub-DS. Figures 6, 7, and 8 display the UB-Layering Step for our example data. First, we construct the first layer, *sub-CH_i[1]*, by computing the convex hull in each sub-DS. The results of constructing sub-CHs in each sub-DS are shown in Fig. 6: the first layer *sub-CH₁[1]* in *sub-DS₁* includes {O₁, O₄, O₅, O₇}, the first *sub-CH₂[1]* in *sub-DS₂* includes {O₁, O₄, O₅, O₇}, the first *sub-CH₃[1]* in *sub-DS₃* includes {O₁, O₄, O₅, O₇}, and the first *sub-CH₄[1]* in *sub-DS₄* includes {O₁, O₄, O₇}. Next, we construct the first layer of UB-H by combining sub-CHs in each sub-DS. The result of the first round of the UB-layering step shows the formation of the first layer UB-H[1] {O₁, O₄, O₅, O₇}, also in Fig. 6.

$$\begin{aligned}
 \text{sub-CH}_1[1] &= \{O_1, O_4, O_5, O_7\} \cup \text{sub-CH}_2[1] = \{O_1, O_4, O_5, O_7\} \\
 &\cup \text{sub-CH}_3[1] = \{O_1, O_4, O_5, O_7\} \cup \text{sub-CH}_4[1] = \{O_1, O_4, O_7\} \\
 &= \text{UB-H}[1] = \{O_1, O_4, O_5, O_7\}
 \end{aligned}$$

Fig. 6 An example of the result of the UB-layering step (the first round)

Figure 7 shows the second round of the UB-layering step. We construct the second layer of UB-H using the same process as the first round in each sub-DS. We compute the second layer in each sub-DS and combine sub-CH₁, sub-CH₂, sub-CH₃, and sub-CH₄ as {O₂, O₃, O₆}. Next, we compared our generated set to UB-H[1], in order to remove duplicate objects. In the second round of this example, there are no duplicate objects to be removed, so the generated set {O₂, O₃, O₆} becomes the second layer UB-H[2].

Figure 8 shows the third round of the UB-layering step. We construct the third layer of UB-H using the same process as the previous round in each sub-DS. We compute the third layer in each sub-DS and combine sub-CH₁, sub-CH₂, sub-CH₃, and sub-CH₄ to obtain the set {O₅}. Next, we compare our generated set to our two

Fig. 7 An example of the result of the UB-layering step (the second round). As there are no duplicate objects between the generated set and UB-H[1], the generated set becomes the second layer UB-H[2]

generated layers, UB-H[1] and UB-H[2], to remove duplicate objects. The object O_5 is already included in the layer UB-H[1]. Therefore, we removed the duplicate object, and UB-H[3] became an empty layer. Finally, if there are no objects left on each sub-DS, we complete the UB-layering step.

Table 3 shows the ConstructingUB-H algorithm and the inputs of the algorithm are DS , a set of d -dimensional data objects, with d denoting the number of attributes. The result of the ConstructingUB-Layer algorithm is $sub-DSs$, which are sets of data objects with partitioned dimensions. We check the size of the input dimension d in line 1. If d is less than 4, the algorithm does not act

Fig. 8 An example of the result of the UB-layering step (the third round). As the single object generated set was a duplicate of an item in a generated layer, it is discarded such that UB-H[3] is the empty set

because it is impossible to divide such a dataset. In this case, we construct the convex hull on line 2 and the algorithm ends. If d is greater than or equal to 4, it executes the Hier-dividing algorithm for dividing the dimension of input data. Next, we compute the sub convex hull in each sub dimension-divided dataset. In line 7–8, we compute the UB-layering step by combining each sub convex hulls and in line 9, we finally construct the result UB-H layer.

Table 3 The ConstructingUB-H algorithm**Algorithm: ConstructingUB-Layer****Input:**

- (1) DS : a set of d -dimensional objects
- (2) d : the number attributes

Output:

- (1) $UB-H$: the result layer

Algorithm:

1. **IF** $d < 4$ **THEN**
2. Construct a convex hull of DS
3. **Break**
4. $sub-DS = Hier-dividing(DS, d)$ /* sub-DS: dimension-divided dataset */
5. **FOR** $i = 0$ **TO** $k-1$ **DO** /* k : the total number of $sub-DS$ */
6. Compute the sub convex hull $sub-CH[c]_i$ over $sub-DS_i$.
 /* c : the number of convex hull layers */
7. **FOR** $j = 0$ **TO** $c-1$
8. $UB-H[j] = sub-CH[0] \cup sub-CH[1] \cup \dots \cup sub-CH[k-1]$
 $\quad \quad \quad - UB-H[j-1]$ /* $j-1 \geq 0$ */
9. $UB-H = UB-H[0] \cup UB-H[1] \cup \dots \cup UB-H[c-1]$
10. **RETURN** $UB-H$

5 Analysis

In this section, we analyze the time complexity of UB-H in comparison with the convex hull method. For ease of analysis, we used a uniform object distribution. For the amount of input data, n , the time complexity of building one convex hull is given by [30, 31],

$$time_{convexhull} = \begin{cases} O(n \log v), & \text{if } d = 2, 3 \\ O\left(n \times \frac{v^{\lceil \frac{d}{2} \rceil}}{\lceil \frac{d}{2} \rceil!}\right), & \text{if } d \geq 4 \end{cases} \quad (1)$$

where d signifies the number of dimensions and v is the number of data objects that constitute the convex hull layer. The time complexity of UB-H is shown in Eq. (2), and is determined by the amount of input data n , the dimension number d , and the number of data objects v constituting the convex hull. The time complexity of UB-H is the same as that of the convex hull method when d is two or three because UB-H does not divide the dimensions in these cases. When d is greater than or equal to four, we construct the layers by dividing the dimensions. Here, c_1 and c_2 are constants representing the cost of dividing and combining dimensions, respectively.

$$\begin{aligned}
 time_{UB-H} &= \begin{cases} O(n \log v), & \text{if } d = 2, 3 \\ O\left(C1 + n \times \frac{v^{\left\lceil d \times \frac{1}{2} \right\rceil}}{\left\lceil d \times \frac{1}{2} \right\rceil!}\right) \times 2 + C2, & \text{if } d \geq 4 \end{cases} \\
 &= \begin{cases} O(n \log v), & \text{if } d = 2, 3 \\ O\left(2n \times \frac{v^{\left\lceil \frac{d}{4} \right\rceil}}{\left\lceil \frac{d}{4} \right\rceil!}\right) + C1 + C2, & \text{if } d \geq 4 \end{cases} \quad (2) \\
 &= \begin{cases} O(n \log v), & \text{if } d = 2, 3 \\ O\left(2n \times \frac{v^{\left\lceil \frac{d}{4} \right\rceil}}{\left\lceil \frac{d}{4} \right\rceil!}\right), & \text{if } d \geq 4 \end{cases}
 \end{aligned}$$

6 Experimental result

In this section, we first explain the data and environment used in the performance evaluation, and then we present the experimental results. For the experiments, we compared the performance of UB-H and the existed methods, UB-Layer, and the convex hull method, in terms of the computing time of index building, total number of layers, and accuracy. The index building time was measured in wall clock time, while we compared the number of data points included in the layer to calculate accuracy. We compared the total number of layers for the query efficiency. UB-H, UB-Layer and convex hull are layer-based index building method for top- k query processing. More layers means fewer objects in one layer and it is more efficient because we retrieve smaller objects with same number of layers while query processing. We used the same accuracy equation as was proposed in [14], and took our input data from the HL-Index data generator [18]. We performed the experiment by varying the data quantities and number of attributes. For the experiment, we constructed the UB-H, UB-Layer, and convex hull using C++, and conducted all experiments with an Intel i5-760 quad core processor, running on a 2.80 GHz PC with Linux OS and 16 GB of main memory. Table 4 summarizes the variables used for experiments.

Experiment 1 The index building time comparison as dimension d is varied (data size $N=10,000$).

Figure 9 shows the index building time of UB-H, UB-Layer, and the convex hull method when dimension d is varied between 4 and 8. As the number of dimensions increases, the index building time of the convex hull algorithm

Table 4 The variables for UB-H experiments

Experiment	Variables	
	Dataset	UNIFORM
Experiment 1: The index building time comparison as dimension d is varied	N	10,000
	d	6–8
Experiment 2–1: The total number of layers comparison as dimension d is varied	N	10,000
	d	6–8
Experiment 2–2: The total number of layers comparison as dimension d is varied	N	10,000
	d	4–12
Experiment 3: The accuracy comparison as dimension d is varied	N	10,000
	d	4–8
Experiment 4: The index building times comparison as dimension d was varied	N	200
	d	4–10
Experiment 5: The total number of layers comparison as dimension d was varied	N	200
	d	4–10
Experiment 6: The accuracy comparison as dimension d was varied	N	200
	d	4–10

Fig. 9 The result of comparison of index building time from three different methods ($6 \leq d \leq 8$)

increases exponentially. Once a dimension of 9 is reached, it is impossible to construct a convex hull. However, the index building time of UB-H increases logarithmically. The UB-H reduces the index building time of the UB-Layer by 1.3 times on average, and by 66 times on average when compared to the convex hull. Figure 9b shows the results on a logarithmic scale.

Experiment 2-1 The total number of layers comparison as dimension d is varied (data size $N=10,000$).

Figure 10 shows a comparison of the total number of layers created by the UB-H, UB-Layer, and convex hull methods as dimension d is varied between 6

Fig. 10 The result of comparison of the total number of layers produced by three methods ($6 \leq d \leq 8$)

and 8. Compared to the convex hull method, the UB-Layer and UB-H constructs, on average, 5.4 and 19.5 times more layers, respectively.

Experiment 2-2 The total number of layers comparison as dimension d is varied (data size $N=10,000$).

Figure 11 shows the comparison between UB-H and UB-Layer as dimension d is varied from 4 to 12. On average, UB-Layer constructs 54.2 layers and UB-H constructs 101.1 layers, hence the total number of layers created by UB-H is about two times more than UB-Layer. A large total number of layers means less data in a single layer, which is advantageous in achieving fast query processing.

Experiment 3 The accuracy comparison as dimension d is varied (data size $N=10,000$).

Fig. 11 The result of comparison of the total number of layers produced by two methods ($4 \leq d \leq 12$)

Fig. 12 The result of comparison of the accuracy of UB-H and UB-Layer methods ($6 \leq d \leq 8$)

Fig. 13 The result of comparison of index building time ($N=200$, $4 \leq d \leq 10$)

Figure 12 compares the accuracy of UB-H and UB-Layer as dimension d is varied from 6 to 8. We define the accuracy is 100% when the number of data in the first layer of convex hull and UB-H are same [14]. We could not compare accuracy at dimensions higher than 9 because the convex hull calculation could not be executed. The UB-H shows 32% accuracy on average; however, the accuracy of the proposed method improves when the number of dimensions increases. That is because the number objects which are included in one layer of convex hull became more and more when the number of dimensions increases. Therefore, the proposed method provides more accurate results in high-dimensional data.

Experiment 4 The index building time comparison as dimension d was varied (data size $N=200$).

Figure 13 shows the index building time as a wall clock time for the UB-H, UB-Layer, and convex hull methods when dimension d is varied between 4 and 10. To allow high-dimensional data to be compared, we set the data size N to 200. As the number of dimensions increases, the index building time of the convex

hull algorithm increases exponentially. However, the index building time of UB-H increases logarithmically. The UB-H reduces the index building time of the UB layer by 2.5 times on average, and by 6,073 times on average when compared to the convex hull method. Figure 12b shows the results on a logarithmic scale.

Experiment 5 The total number of layers comparison as dimension d was varied (data size $N=200$).

Figure 14 shows the comparison of the total number of layers between the UB-H, UB-Layer, and convex hull methods as dimension d is varied between 4 and 10. To allow method comparison with high-dimensional data, we again set the data size N to 200. Compared to the convex hull method, UB-Layer and UB-H constructs 1.7 and 2.7 times more layers on average, respectively. In addition, UB-H constructs 1.6 times more layers than UB-Layer on average.

Experiment 6 The accuracy comparison as dimension d was varied (data size $N=200$).

Figure 15 compares the accuracy of UB-H and UB-Layer methods as dimension d is changed from 4 to 10. In order to compare the methods using high-dimensional data, we lowered the data size N to 200, and kept it fixed. The accuracy is 100% if all input data from first in the constructed convex hull are included [14]. The UB-H shows 87% accuracy on average, while the accuracy of the proposed method improves when the number of dimensions increases. Therefore, the proposed method provides more accurate results in high-dimensional data.

Fig. 14 The result of comparison of the total number of layers in three methods ($N=200$, $4 \leq d \leq 10$)

Fig. 15 The result of comparison in accuracy between UB-H and UB-Layer methods ($N=200$, $4 \leq d \leq 10$)

7 Conclusion

Recently, the amount of data generation and storage increase rapidly, therefore the importance of cloud computing and green computing is increases. For efficient data management, an index is generally constructed, and the layer based indexing method is one of representative method. In this paper, we propose an unbalanced-hierarchical (UB-H) layer. This method increases the total number of layers and reduces the index building time, compared to the UB-Layer and the convex hull method. The proposed method first divides the dimensions of input data hierarchically into two or three sub-datasets. Next, we build the sub-convex hull in each sub-dataset and construct the final UB-H as an index by combining each sub-convex hull. The experimental results show that UB-H is constructed faster than the existing methods and has a greater number of layers.

The proposed method is very efficient in applications that require fast results, yet do not necessitate completely accuracy; for the instance in hotel and used car searches. However, this method in its current form is not suitable for applications that require accuracy. The proposed method improves the computation cost with maintaining the accuracy as the number of dimensions increases, however, it is not quite accurate. In future, we will study the algorithm to improve the accuracy of the proposed method. We will also apply the proposed method to real-world applications.

Funding This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.2016-0-00406, SIAT CCTV Cloud Platform). This research was supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2018R1A5A7023490).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Liu J, Yang J, Xiong L, Pei J (2017) Secure Skyline Queries on Cloud Platform. In: Proceedings of 2017 IEEE 33rd international conference on data engineering (ICDE), San Diego, CA, USA, pp 19–22, IEEE
2. Gao H, Xu Y, Yin Y, Zhang W, Li R, Wang X (2019) Context-aware QoS prediction with neural collaborative filtering for internet-of-things services. *IEEE Int Things J*. <https://doi.org/10.1109/JIOT.2019.2956827>
3. Yin Y, Xia J, Li Y, Xu Y, Xu W, Yu L (2019) Group-wise itinerary planning in temporary mobile social network. *IEEE ACCESS* 7:83682–83693
4. Kaur M, Mahajan M (2016) An improved security mechanism for protecting data in mobile cloud environment. *Int J Adv Sci Technol* 89:37–44
5. Im S, Hwang H, Ouyang J (2015) SolidStream: NGS encoding utility with high speed and high efficiency considering transmission bandwidth of cloud environments. *Int J Control Autom* 8(8):223–232
6. Yuan H, Li C, Du M (2016) Research on fuzzy clustering method for cloud computing task scheduling. *Int J Control Autom* 9(11):421–428
7. Bachhav A, Kharat V, Shelar M (2017) Query optimization for databases in cloud environment: a survey. *Int J Database Theory Appl* 10(6):1–12
8. Murugesan S (2008) Harnessing green IT: principles and practices. *IEEE IT Prof* 10(1):24–33
9. Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis, a framework for stream classification and clustering. *J Mach Learn Res* 11:1601–1604
10. Raza K, Patle VK, Arya S (2012) A review on green computing for eco-friendly and sustainable IT. *J Comput Intell Electr Syst* 1:1–14
11. Ihm SY, Nasridinov A, Lee JH, Park YH (2014) Efficient duality-based subsequent matching on time-series data in green computing. *J Supercomput* 69:1039–1053
12. Stein A, Geva E, EL-Sana J (2012) CudaHull: fast parallel 3D convex hull on the GPU. *Comput Graph* 36:265–271
13. Ding S, Nie X (2018) A fast algorithm of convex hull vertices selection for online classification. *IEEE Trans Neural Netw Learn Syst* 29:792–806
14. Ihm SY, Hur JH, Park YH (2017) An indexing method to construct unbalanced layers for high-dimensional data in mobile environments. *Wirel Commun Mob Comput* 2017:1–13
15. Ramli AA, Watada J, Pedrycz W (2011) Real-time fuzzy regression analysis: a convex hull approach. *Eur J Oper Res* 210:606–617
16. Ruano A, Khosravani HR, Ferreira PM (2015) A randomized approximation convex hull algorithm for high dimensions. *ifac-papersonline* 48:123–128
17. Chang YC, Bergman L, Castelli V, Li CS, Lo ML, Smith JR (2000) The onion technique: indexing for linear optimization queries. In: Proceedings of the international conference on management of data (SIGMOD), Dallas, Texas, USA, 15–18 May, ACM, pp 391–402
18. Heo JS, Cho J, Whang KY (2010) The hybrid-layer index: a synergic approach to answering top-k queries in arbitrary subspaces. In: Proceedings of the 26th international conference on data engineering (ICDE), Long Beach, CA, USA, 1–6 March; IEEE, pp 445–448
19. Heo JS, Cho J, Whang KY (2013) Subspace top-k query processing using the hybrid-layer index with a tight bound. *Data Knowl Eng* 83:1–19

20. Ihm SY, Nasridinov A, Park YH (2014) An efficient index building algorithm for selection of aggregator nodes in wireless sensor networks. *Int J Distrib Sens Netw* 2014:1–8
21. Liu R, Fang B, Tang UU, Wen J, Qian J (2012) A fast convex hull algorithm with maximum inscribed circle affine transformation. *Neurocomputing* 77:212–221
22. Xie M, Wong RC, Li J, Long C, Lall A (2018) Efficient k-Regret Query Algorithm with Restriction-free Bound for any Dimensionality. In: *Proceedings of the 2018 international conference on management of data*, pp 959–974, Houston, TX, USA, 10–15 June, ACM
23. Peng P, Wong RC (2014) Geometry approach for k-regret query. In: *Proceedings of the IEEE 30th international conference on data engineering*, Chicago, IL, USA, 31 March–4 April, IEEE
24. Mouratidis K, Tang B (2018) Exact processing of uncertain top-k queries in multi-criteria settings. *Proc VLDB Endow* 11(8):866–879
25. Armbrust M, Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2010) A view of cloud computing. *Commun ACM* 53(4):50–58
26. Sarkar S, Misra S (2016) Theoretical modelling of fog computing: a green computing paradigm to support IoT applications. *IET Netw* 5(2):23–29
27. Kochhar N, Garg A (2011) Eco-Friendly Computing: Green Computing. *Int J Comput Bus Res*, 2(2)
28. Soomro TR, Sarwar M (2012) Green computing: from current to future trends. *Int J Hum Soc Sci* 6(3):455–458
29. Cao Z, Easterling DR, Watson LT, Li D, Cameron KW, Feng WC (2010) Power saving experiments for large-scale global optimization. *Int J Parallel Emerg Distrib Syst* 25(5):381–400
30. Barber B, Dobkin P, Huhdanpaa H (1996) The quickhull algorithm for convex hulls. *ACM Trans Math Softw* 22:469–483
31. Klee V (1966) Convex polytopes and linear programming. In: *Proceedings of the IBM scientific computing symposium: combinatorial problems*, pp 123–158
32. Liu X, Wang T, Jia W, Liu A, Chi K (2019) Quick convex hull-based rendezvous planning for delay-harsh mobile data gathering in disjoint sensor networks. *IEEE Trans Syst Man Cybern Syst*, pp 1–11
33. Ferrada H, Navarro CA, Hitschfeld N (2020) A filtering technique for fast Convex Hull construction in R2. *J Comput Appl Math* 364:1–12

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.