Indexing High-Dimensional Data in Dual Distance Spaces: *A Symmetrical Encoding Approach*

Yi Zhuang[1] Yueting Zhuang[1] Qing Li[2] Lei Chen[3] Yi Yu[4]
[1]College of Computer Science, Zhejiang University, P.R.China
[2]Dept of Computer Science, City University of Hong Kong, HKSAR, P.R.China
[3]Dept of Computer Science, Hong Kong University of Science and Technology, HKSAR, P.R.China
[4]Graduate School of Humanity and Science, Nara Women's University, Japan
{zhuangyi,yzhuang}@zju.edu.cn, itqli@cityu.edu.hk, leichen@cse.ust.hk, yuyi@ics.nara-wu.ac.jp

ABSTRACT

Due to the well-known dimensionality curse problem, search in a high-dimensional space is considered as a “hard” problem. In this paper, a novel symmetrical encoding-based index structure, which is called EHD-Tree (for *symmetrical Encoding-based Hybrid Distance Tree*), is proposed to support fast *k*-Nearest-Neighbor (*k*-NN) search in high-dimensional spaces. In an EHD-Tree, all data points are first grouped into clusters by a *k*-Means clustering algorithm. Then the uniform ID number of each data point is obtained by a dual-distance-driven encoding scheme in which each cluster sphere is partitioned twice according to the dual distances of *start*- and *centroid-distance*. Finally, the uniform ID number and the centroid-distance of each data point are combined to get a uniform index key, the latter is then indexed through a partition-based B^+-tree. Thus, given a query point, its *k*-NN search in high-dimensional spaces can be transformed into search in a single dimensional space with the aid of the EHD-Tree index. Extensive performance studies are conducted to evaluate the effectiveness and efficiency of our proposed scheme, and the results demonstrate that this method outperforms the state-of-the-art high dimensional search techniques such as the X-Tree, VA-file, iDistance and NB-Tree, especially when the query radius is not very large.

1. INTRODUCTION

With the explosive increase of multimedia data on the Internet, content-based image or video retrieval has become more important than ever before. For example, when people read the news on the web, he or she may want to find an interesting picture or video related to the news. Using this picture or video as an example, he or she may want to find *similar* images or videos. With consideration of the large scale data available on the web and requests from potentially large number of users, it is critical to devise a mechanism that can speed up the search process. However, due to the complexity of multimedia objects, these objects are represented by high-dimensional vectors where each entry of the vector is a representative feature. As a consequence, we need to find indexing techniques for these high dimensional data. Due to the well known high dimensional curse problem [1],

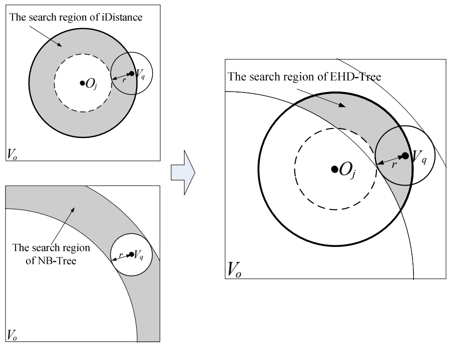

The search region of iDistance
O_j
r
V_q
V_o
The search region of EHD-Tree
O_j
r
V_q
V_o
The search region of NB-Tree
r
V_q
V_o

Figure 1. The comparison of three search regions

traditional indexing methods such as K-d-tree [2] and R-tree [3] only work well in low dimensional spaces (e.g., up to 12-16 dimensions [1]). Therefore, the design of efficient indexing techniques for high-dimensional data remains to be an active research area [1].

In this paper, we propose a high-dimensional indexing scheme based on the technique of *symmetrical Encoding-based Hybrid Distance Tree*, called EHD-Tree, to support progressive *k*-NN search. Specifically, in an EHD-Tree, all data points are first grouped into clusters by using *k*-Means clustering. Then, the uniform ID(*UID*) number of each point is obtained by a dual-distance-driven encoding scheme in which each cluster sphere is partitioned twice according to the dual distances: *start*- and *centroid-distance*. Finally, the uniform index key of each data point is obtained through linearly combing its *UID* with the centroid distance together, and is indexed by a partition-based B^+-tree. Using the B^+-tree structure to index a high-dimensional space brings forth many strengths, including fast search, dynamic update, height balanced structure and so on. It also makes it easier to graft our technique on top of any existing commercial relational database. Thus, given a query data point V_q and the number *k*, the *k*-Nearest Neighbor search of V_q in a high-dimensional space is transformed into search in a single dimensional space with the aid of the EHD-Tree indexing facilities.

Figure 1 compares intuitively the search region of our EHD-Tree with those of the counterparts like NB-Tree [18] and iDistance [19]. As we know, iDistance and NB-Tree only adopt single distance metric, viz., *the centroid-distance or start-distance respectively* (to be elaborated in Section 3.1) to prune the search region. EHD-Tree, on the other hand, aims to accommodate the dual distance metrics via using a symmetrical encoding scheme, so as to further reduce the search region (ref. the shadow region of the right part of Figure 1). Based on the observation, it is clear that pruning by dual metrics is more promising on reducing the search space, which is confirmed by the theoretically analysis to be detailed in Section 4. However, it is a non-trivial problem to combine two metrics for pruning, because simply using one after another will not achieve a better pruning effect than that by using only one. In this paper, we propose a novel encoding schema which combines the two metrics effectively, which can be seamlessly incorporated into a B^+-tree. An extensive performance study is then conducted to evaluate the EHD-Tree's effectiveness and efficiency. Our results on various data sets show that the proposed technique has better performance than that of X-tree [6], VA-file [9], NB-Tree [18] and iDistance [19], especially when the query radius is not very large.

The primary contributions of this paper are listed as follows:

1. We propose a symmetrical encoding scheme by double partitioning the cluster sphere according to the dual distance metrics (i.e., *start-distance* and *centroid-distance*). Thus the uniform ID number of each point can be obtained to serve as an integral part of its uniform index key.

2. We design a uniform index key by linearly combing the uniform ID number of each point with the centroid distance together, which enables the corresponding range of these two values to be non-overlapping. Furthermore, based on this encoding scheme, we propose a *symmetrical Encoding-based Hybrid Distance Tree*(EHD-Tree) to facilitate highly efficient *k*-NN search.

3. We present a theoretical analysis and comparison on the search cost of the proposed method and related indexing methods, and give a cost model for the proposed indexing method.

The rest of this paper is organized as follows. We survey the related work in Section 2. In Section 3, we describe EHD-Tree, which is devised to dramatically improve the query performance of the *k*-NN search. In Section 4, we give a theoretical analysis for EHD-Tree. In Section 5, we report several extensive experiments conducted to evaluate the efficiency and effectiveness of the EHD-Tree index and compare it with its counterparts. We conclude the paper in Section 6.

2. RELATED WORK

There is a long stream of research for addressing the high-dimensional indexing problems [1]. Existing techniques can be divided into four main categories.

The first category is based on data and space partitioning, hierarchical tree index structure (e.g., the R-tree [3] and its variants [4, 5, 6, 7, 8]), etc. Although these methods generally perform well at low dimensionality, their performance deteriorates rapidly as the dimensionality increases due to the "*dimensionality curse*".

The second category is to represent original feature vectors using smaller, approximate representations (e.g., VA-file [9], IQ-tree [10] and A-tree [11]), etc. The VA-file [9] accelerates the sequential scan by using data compression. Although the VA-file reduces the number of disk accesses, it incurs higher computational cost to decode the bit-strings. The IQ-tree [10] is also an indexing structure along the lines of the VA-file, which maintains a flat directory containing the minimum bounding rectangles of the approximate data representations. A-tree [11] is yet another tree structure based on the Virtual Bounding Rectangles(VBRs) which are approximations of minimal bounding rectangles (MBRs) and data objects.

The third category is to use a metric-based method [12] as an alterative direction for high-dimensional indexing. Examples include MVP-Tree [13], M-Tree [14] and Slim-Tree [16], Omni –family technique [17], etc.

The final category is the transformation-based high-dimensional indexing schemes, such as the Pyramid Technique [15]. The Pyramid Technique is efficient for window queries, but performs less satisfactorily for *k*-NN queries. Most recently, NB-tree [18] and iDistance [19] are proposed to support B^+-tree-based *k*-NN search. NB-tree is a single reference point-based scheme, in which high-dimensional points are mapped to a single-dimension by computing their distance from the origin individually. Then these distances are indexed using a B^+-tree on which all subsequent operations are performed. The drawback of NB-Tree is that it can not significantly prune the search region; especially when the dimensionality is becoming larger, the pruning capability of it can be so poor that the number of candidate points returned by the first round becomes too large to be filtered effectively. iDistance [19] is proposed by selecting some reference points in order to further prune the search region so as to improve the query efficiency, and is testified to be superior to M-Tree [14] and Omni-family [17] empirically [19]. However the query efficiency of iDistance relies largely on clustering and partitioning the data and is significantly affected if the choice of partition scheme and reference data points is not appropriate.

3. THE EHD-TREE

In order to reduce the search region and speed up the *k*-NN queries, in this section, we present a novel high-dimensional indexing technique called the symmetrical *Encoding-based Hybrid Distance Tree* (EHD-Tree for short).

3.1 Preliminaries & Motivations

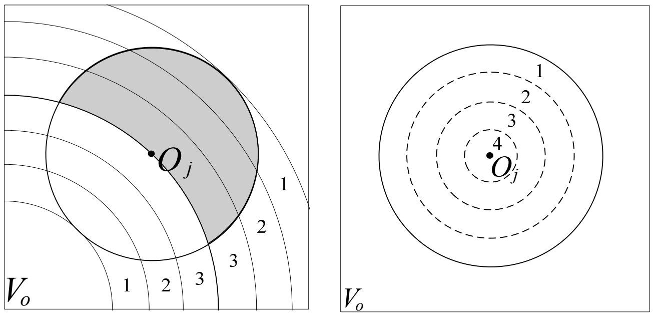
The design of EHD-Tree is motivated by the following observations. First, the (*dis*)similarity between data points can be derived and ordered based on their distances to a reference data point. Second, a distance is essentially a single dimensional value which enables us to reuse existing single dimensional indexing schemes such as B^+-tree. Third, as shown in Figure 1, it is hard to effectively reduce the search region by only using a single distance metric (e.g., *start-distance* [18] or *centroid-distance* [19]). The basic idea behind EHD-Tree is to nicely combine the two distance metrics together by using a novel symmetrical encoding scheme to obtain a uniform index key expression, so as to further reduce the search region.

The list of symbols to be used in the rest of paper is summarized in Table 1.

TABLE 1: Meaning of Symbols Used

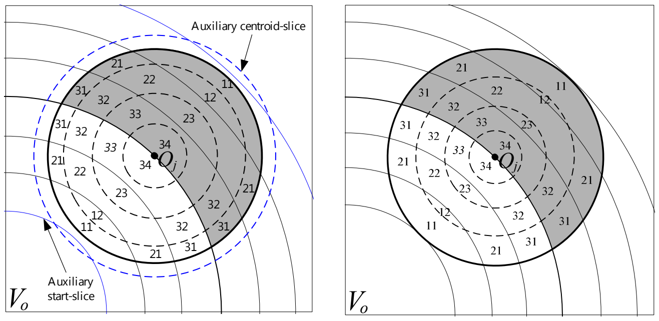
Symbols	Meaning
Ω	a set of data points
V_i	the *i*-th data point and $V_i \in \Omega$
D	number of dimensions
n	number of data points in Ω
V_q	a query data point user submits
α	number of the start slices in a cluster sphere
β	number of the centroid slices in a cluster sphere
$\Theta(V_q,r)$	the query sphere with centre V_q and radius r
$d(V_i,V_j)$	the distance between two points
$\lceil \bullet \rceil$	the integral part of $\bullet$

Without loss of generality, we assume Euclidean distance as the



(a). An example start-slices (b). An example centroid-slices

Figure 2. The dual-distance-driven encoding for the cluster sphere $\Theta(O_j,CR_j)$



(a). Global Partition (b). Local Partition

Figure 3. The two partition methods for the cluster sphere $\Theta(O_j,CR_j)$

distance function, denoted as $d(V_i,V_j)$, although other distance functions also apply for EHD-Tree.

DEFINITION 1(START DISTANCE). *Given a data point V_i, its Start Distance (SD for short) is the distance between it and the origin $V_o(0,0,..,0)$, formally defined as:*

$$SD(V_i)=d(V_i,V_o) \quad (1)$$

Assuming that n data points are grouped into T clusters, the centroid O_j of each cluster C_j is first obtained, where $j\in[1,T]$. We model a cluster as a tightly bounded sphere described by its *centroid* and *radius*.

DEFINITION 2 (CLUSTER RADIUS). *Given a cluster C_j, the distance between its centroid O_j and the data point which has the longest distance to O_j is defined as the cluster radius of C_j, denoted as CR_j.*

Given a cluster C_j, the cluster sphere of it is denoted as $\Theta(O_j, CR_j)$, where O_j is the centroid of cluster C_j, and CR_j is the cluster radius.

DEFINITION 3 (CENTROID DISTANCE). *Given a data point V_i, its centroid distance is defined as the distance between itself and the cluster centroid O_j, and is denoted as:*

$$CD(V_i)=d(V_i,O_j) \quad (2)$$

where $i\in[1,\|C_j\|]$ *and* $j\in[1,T]$.

3.2 Symmetrical DDE Scheme

As mentioned before, for each data point V_i in a cluster sphere, it is a non-trivial task to simply combine the start-distance [18] and centroid-distance [19] of V_i to get its uniform index key. To address this problem, we propose a *Dual-Distance-Encoding* scheme, called *DDE,* to obtain a new uniform index key through double "slicing" the cluster sphere in terms of the values of the *start-* and *centroid-distance*, which can be used to further reduce the search region in the high-dimensional space

Specifically, all data points are first grouped into T clusters by using a k-Means clustering algorithm, then the start- and centroid-distances of each data point are computed, thus V_i can be modeled as a four-tuple:

$$V_i ::= <i, CID, SD, CD> \quad (3)$$

where i refers to the i-th data point and CID is the ID of the cluster that V_i belongs to.

DEFINITION 4 (START SLICE). *Given a cluster sphere* $\Theta(O_j,CR_j)$ *the γ-th start-slice of it is denoted as $SS(\gamma,j)$, which is based on the value of start-distance, where $\gamma\in[1,\alpha]$ and j is the ID number of the cluster sphere.*

Figure 2(a) shows an example of encoding scheme of start-slice in the cluster sphere $\Theta(O_j,CR_j)$. Given a data point $V_i\in\Theta(O_j,CR_j)$, there are two cases to be considered in terms of the start distance:

(i) When $SD(V_i)\in[SD(O_j),SD(O_j)+CR_j]$: as shown in the shaded part of the cluster sphere in Figure 2(a), the ID number of start-slice is decreasing with the increase of the start distance of the point in $\Theta(O_j,CR_j)$.

(ii) When $SD(V_j)\in[SD(O_j)-CR_j,SD(O_j)]$: as shown in the white part of the cluster sphere in Figure 2(a), the ID number of start-slice is increasing with the increase of the start distance of the point in $\Theta(O_j,CR_j)$.

DEFINITION 5 (CENTROID SLICE). *Given a cluster sphere $\Theta(O_j,CR_j)$, the μ-th centroid-slice of it is denoted as $CS(\mu,j)$, which is based on the value of centroid-distance, where $\mu\in[1,\beta]$ and j is the ID number of the cluster sphere.*

Similarly, Figure 2(b) shows an example encoding scheme of centroid-slice in $\Theta(O_j,CR_j)$, in which the ID number of centroid-slice decreases with the increase of the centroid distance of the point in $\Theta(O_j,CR_j)$.

In order to effectively encode the data points in a cluster sphere, we propose two partition methods, namely *Global Partition* and *Local Partition*, for comparison purpose.

DEFINITION 6 (GLOBAL PARTITION). *Given a cluster sphere $\Theta(O_j,CR_j)$,its global partition is to equally slice the cluster sphere into β centroid-slices in terms of the thickness value (∇) of slice based on the centroid distance.*

As a result, α start-slices are made in $\Theta(O_j,CR_j)$, as shown in Figure 3(a), where $\beta=\lceil CR_j/\nabla\rceil+1$,α is an even integer and $\alpha/2+1=\beta, j\in[1,T]$.

DEFINITION 7 (LOCAL PARTITION). *Given a cluster sphere $\Theta(O_j,CR_j)$, its local partition is to equally slice the cluster sphere into α start-slices.*

Consequently, β centroid-slices are made in $\Theta(O_j,CR_j)$, as shown in Figure 3(b), where α is an even number and $\alpha/2+1=\beta, j\in[1,T]$.

For the example shown in Figure 3, the number of the start slices in the cluster sphere is 6 and the number of the centroid slices is 4. Note that, different from the local partition (c.f., Figure 3(b)), it is impossible for the global partition (c.f., Figure 3(a)) of $\Theta(O_j,CR_j)$ to exactly slice the cluster sphere based on the centroid distance. Therefore the auxiliary start- and centroid-slices are introduced, which are shown as blue circle in Figure 3(a).

Based on the above two partition methods, a data point V_i also can be re-written by a four-tuple:

$$V_i ::= <i, CID, SD\_ID, CD\_ID> \quad (4)$$

where i is the ID number of V_i, CID is the cluster ID number where V_i belongs to, SD_ID is the slice ID number in terms of the start-distance-based slicing method and CD_ID is the slice ID number in terms of the centroid-distance-based slicing method.

For a data point V_i in the corresponding cluster sphere, the ID number of the start slice and centroid slice V_i falls in can be derived as follows:

$$\begin{cases} SD\_ID(V_i) = \frac{\alpha}{2} - \left\lceil \frac{|SD(V_i) - SD(O_j)|}{2CR_j/\alpha} \right\rceil \\ CD\_ID(V_i) = 1 + \left\lceil \frac{CR_j - CD(V_i)}{CR_j/\beta} \right\rceil \end{cases} \quad (5)$$

where α is an even integer and $\alpha/2+1=\beta$.

Therefore the uniform ID number of V_i can be represented by linearly combining the SD_ID and CD_ID, as follows:

$$\begin{aligned} UID(V_i) &= c \times SD\_ID(V_i) + CD\_ID(V_i) \\ &= c \times \left(\frac{\alpha}{2} - \left\lceil \frac{|SD(V_i) - SD(O_j)|}{2CR_j/\alpha} \right\rceil \right) + 1 + \left\lceil \frac{CR_j - CD(V_i)}{CR_j/\beta} \right\rceil \end{aligned} \quad (6)$$

where c is a constant to linearly stretch the SD_ID and CD_ID.

Based on this encoding scheme, we get the uniform encoding identifier of each data point by double slicing the cluster sphere, as shown by Algorithm 1.

```
Algorithm 1. The Dual-Distance-driven Encoding
Input:  Ω: the data point set,
        α: the number of start slices in each cluster sphere;
        β: the number of centroid slices in each cluster sphere;
Output: DDE(1 to n): the encode representation for n points;
1. The data points in Ω are grouped into T clusters using k-Means
   clustering algorithm
2. for j:=1 to T do
3.     for $V_i \in \Theta(O_j, CR_j)$ do
5.         the centroid distance and start distance of it are computed;
6.         the ID numbers of the start- and centroid-slices where $V_i$ belongs
           to is identified by Eq.(5);
7.         the uniform ID of $V_i$ ($DDE(V_i)$) can be derived by Eq.(6);
8.     end for
9. end for
```

Figure 4. The symmetrical encoding algorithm

3.3 The Data Structure

Based on the above definitions, we can get the index key of V_i as follows:

$$\begin{aligned} key(V_i) &= UID(V_i) + CD(V_i)/MCD \\ &= c \times \left(\frac{\alpha}{2} - \left\lceil \frac{|SD(V_i) - SD(O_j)|}{2CR_j/\alpha} \right\rceil \right) + 1 + \left\lceil \frac{CR_j - CD(V_i)}{CR_j/\beta} \right\rceil + \frac{CD(V_i)}{MCD} \end{aligned} \quad (7)$$

Since $CD(V_i)$ may be larger than one, it should be normalized into the range of [0,1] through division by the Maximal CD(MCD)[Ψ] which is a constant. Thus, it is guaranteed that the search range of the uniform ID and centroid-distance of each point will not be overlapping.

Eq. (7) shows the index key expression of V_i. However, due to

[Ψ] Since MCD is a maximal value of Euclidean distance of two data points, MCD is set $\sqrt{2}$ for real life data, otherwise MCD is set $\sqrt{d}$.

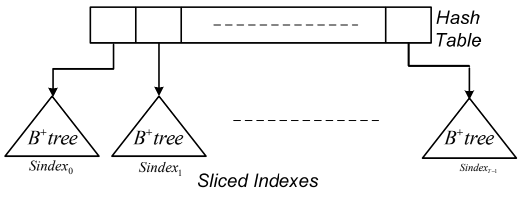


Figure 5. EHD-Tree index architecture

the symmetrical encoding scheme, it is possible that there exist two different data points (e.g., V_i and V_j) in the same cluster sphere, whose index keys are identical. As shown in Figure 2(a), such two data points should satisfy the criteria that: $(SD(O_j)\text{-}SD(V_i))\times(SD(O_j)\text{-}SD(V_j))<0$. To avoid the overlapping of the index keys, a uniform index key ($UKey$) expression is proposed by adding two constants (i.e., $SCALE\_1$ and $SCALE\_2$) to linearly extend the range value of the index key, as follows:

$$UKey(V_i) = \begin{cases} SCALE\_1 + key(V_i), \ if \ SD(V_i) \in [SD(O_j), SD(O_j) + CR_j] \\ SCALE\_2 + key(V_i), \ if \ SD(V_i) \in [SD(O_j) - CR_j, SD(O_j)) \end{cases}$$

$$= \begin{cases} SCALE\_1 + c \times \left(\frac{\alpha}{2} - \left\lceil \frac{|SD(V_i) - SD(O_j)|}{2CR_j/\alpha} \right\rceil \right) + 1 + \left\lceil \frac{CR_j - CD(V_i)}{CR_j/\beta} \right\rceil + \frac{CD(V_i)}{MCD} \\ \quad if \ SD(V_i) \in [SD(O_j), SD(O_j) + CR_j] \\ SCALE\_2 + c \times \left(\frac{\alpha}{2} - \left\lceil \frac{|SD(V_i) - SD(O_j)|}{2CR_j/\alpha} \right\rceil \right) + 1 + \left\lceil \frac{CR_j - CD(V_i)}{CR_j/\beta} \right\rceil + \frac{CD(V_i)}{MCD} \\ \quad if \ SD(V_i) \in [SD(O_j) - CR_j, SD(O_j)) \end{cases} \quad (8)$$

Finally, the n index keys obtained using Eq.(8) are inserted into a partitioned B$^+$-tree, as to be discussed next.

3.4 Building EHD-TREE

Figure 5 shows the index structure of the EHD-Tree which is composed of a hash table and T sliced indexes, where T is the total number of clusters. A cluster sphere corresponds to a sliced index which is named by the cluster ID (viz., CID). The hash table above the sliced indexes is to map the query sphere to the corresponding affected cluster ones (*sliced index*) based on CID.

Figure 6 shows the detailed steps of constructing an EHD-Tree index. Note that the routine ***TransValue***(V_i) is a distance transformation function as given in Eq.(8), and ***BInsert***(dist,*bt(j)*) is a standard B^+-tree insert procedure, where j=1,2,..,T.

```
Algorithm 2. EHD-Tree Index Construction
Input:  Ω: the data point set;
Output: bt(1 to T): the index for EHD-Tree;
1. The data points in Ω are grouped into T clusters using k-Means
   clustering algorithm
2. for j:=1 to T do
3.     bt(j)←newEHDFile();    /*create index header file */
4.     for each data point $V_i$ in the j-th cluster do
5.         the centroid- and start-distance of each data point are computed;
6.         the ID number of $V_i$ is obtained according to algorithm 1;
7.         Key($V_i$)=TransValue($V_i$);
8.         BInsert(Key($V_i$),bt(j));   /* insert it to B+-tree */
9.     end for
10.    return bt(j)
11. end for
```

Figure 6. The EHD-Tree index construction algorithm

As shown in Figure 5, the index keys of the data points in a cluster are indexed by a sliced index. Thus T clusters corresponds to T sliced indexes. Given a query data point V_q, the affected clusters can be quickly accessed through a hash table.

3.5 Index Update Algorithm

We now investigate the problem of EHD-Tree update through a relatively simple algorithm. The detailed steps are shown in Figure 7. Given a new data point V_{new}, the routine ***ClusterID***(V_{new})[Ψ] returns the ID number of the cluster that V_{new} belongs to(line1), and then the index key of this new data point is obtained by the same transformation function as shown in Eq. (8)(line2). Furthermore, the index key of V_{new} is inserted into the corresponding sliced index of EHD-Tree (line3). Finally the updated index is returned (line4).

```
Algorithm 3. EHD-tree Index update
Input: $V_{new}$: a new data point, SI(1 to T): the EHD-Tree index;
Output: SI(1 to T): updated EHD-Tree index;
1.  cid←ClusterID($V_{new}$);
2.  dist←TransValue($V_{new}$);
3.  BInsert( dist, SI(cid));
4.  return SI(1 to T)
```

Figure 7. Update algorithm of EHD-Tree index

3.6 k-NN Search with EHD-Tree

For n high-dimensional data points, k nearest neighbor search is the most frequently used search method which retrieves the k most similar data points(in terms of distance) to a given data point. In this section, we focus on k-NN search for high-dimensional data points with the help of EHD-Tree mechanism.

3.6.1 Picking a Value of LB and UB

Assuming that a query sphere $\Theta(V_q,r)$ intersects with a cluster sphere $\Theta(O_j,CR_j)$, we need to examine which "slices" in $\Theta(O_j,CR_j)$ intersect with $\Theta(V_q,r)$.

DEFINITION 8 (LOW BOUND OF START-SLICE). *Given two intersected spheres $\Theta(V_q,r)$ and $\Theta(O_j,CR_j)$, the low bound of start-slice in $\Theta(O_j,CR_j)$ is the start-slice which is the closest to the boundary of $\Theta(O_j,CR_j)$ in terms of start distance, denoted as* ***LBS***(*j*).

DEFINITION 9 (UPPER BOUND OF START-SLICE). *Given two intersected spheres $\Theta(V_q,r)$ and $\Theta(O_j,CR_j)$, the upper bound of start-slice in $\Theta(O_j,CR_j)$ is the start-slice which is the farthest to the boundary of $\Theta(O_j,CR_j)$ in terms of start distance, denoted as* ***UBS***(*j*).

DEFINITION 10 (LOW BOUND OF CENTROID-SLICE). *Given two intersected spheres $\Theta(V_q,r)$ and $\Theta(O_j,CR_j)$, the low bound of centroid-slice in $\Theta(O_j,CR_j)$ is the the centroid-slice which is the farthest to the cluster centre O_j, denoted as* ***LBC***(*j*).

DEFINITION 11 (UPPER BOUND OF CENTROID-SLICE). *Given two intersected spheres $\Theta(V_q,r)$ and $\Theta(O_j,CR_j)$, the upper bound of centroid-slice in $\Theta(O_j,CR_j)$ is the centroid-slice which is the closest to the cluster centre O_j, denoted as* ***UBC***(*j*).

DEFINITION 12 (CENTROID-HYPERPLANE). Given a cluster sphere $\Theta(O_j,CR_j)$, its centroid hyperplane is a sphere $\Theta(V_o,SD(O_j))$ which is across the cluster centre O_j, where V_o is the origin.

Now we focus on how to get the ***LBS***, ***UBS***, ***LBC*** and ***UBC***. For a query sphere $\Theta(V_q,r)$ in Figure 8, the corresponding search range of its start-distance is $[SD(V_q)-r, SD(V_q)+r]$. Similarly, the search of the centroid-distance is $[CD(V_q)-r, CD(V_q)+r]$. As mentioned in Section 3.2, a cluster sphere (e.g., $\Theta(O_j,CR_j)$) is "sliced" into α start-slices and β centroid-slices. Once a query sphere intersects with the cluster sphere, some continuous "slices" (e.g., from the *LBS*(*j*)-th *"slice"* to the *UBS*(*j*)-th *"slice"*, where *LBS*(*j*)≤*UBS*(*j*)) may be affected (*intersected*). Therefore we can derive the ID number of the low bound start-slice(*LBS*) and upper bound slice (*UBS*) in the *j*-th cluster sphere by the process described immediately below:

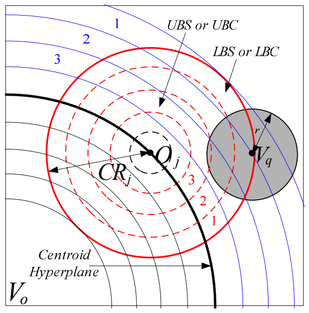


Figure 8. The choice of *LBS, UBS, LBC* and *UBC of* $\Theta(V_q,r)$ in $\Theta(O_j,CR_j)$

For the corresponding *LBS* of $\Theta(V_q,r)$, there are four cases to be considered in terms of the start-distance:

(i). if $SD(V_q)+r \ge SD(O_j)+CR_j$ *or* $SD(V_q)-r > SD(O_j)$, then

$$LBS(j)=1,\quad UBS(j)=\frac{\alpha}{2}-\left\lceil\frac{|SD(V_q)-r-SD(O_j)|}{2CR_j/\alpha}\right\rceil \tag{9}$$

(ii). if $SD(V_q)+r < SD(O_j)+CR_j$ and $SD(V_q)-r > SD(O_j)$ *or* $SD(V_q)-r > SD(O_j)-CR_j$ and $SD(V_q)+r < SD(O_j)$, then

$$LBS(j)=\frac{\alpha}{2}-\left\lceil\frac{|SD(V_q)+r-SD(O_j)|}{2CR_j/\alpha}\right\rceil, UBS(j)=\frac{\alpha}{2}-\left\lceil\frac{|SD(V_q)-r-SD(O_j)|}{2CR_j/\alpha}\right\rceil \tag{10}$$

(iii). if $SD(O_j)-r < SD(V_q) < SD(O_j)+r$, then

$$LBS(j)=\frac{\alpha}{2}-\left\lceil\frac{|SD(V_q)+r-SD(O_j)|}{2CR_j/\alpha}\right\rceil and\ \frac{\alpha}{2}-\left\lceil\frac{|SD(V_q)-r-SD(O_j)|}{2CR_j/\alpha}\right\rceil,$$

$$UBS(j)=\frac{\alpha}{2} \tag{11}$$

Note that, in this case, $\Theta(V_q,r)$ is intersected with the centroid hyperplane of $\Theta(O_j,CR_j)$, therefore there exists two values of *LBS* due to the symmetry.

(iv). if $SD(V_q)-r \le SD(O_j)-CR_j$ and $SD(V_q)+r < SD(O_j)$, then

$$LBS(j)=1,\quad UBS(j)=\frac{\alpha}{2}-\left\lceil\frac{|SD(V_q)+r-SD(O_j)|}{2CR_j/\alpha}\right\rceil \tag{12}$$

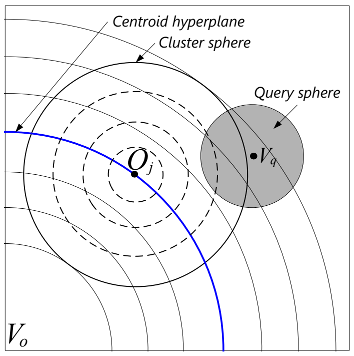
Similarly, we can get the ID numbers of the low bound (*LBC*) and the upper bound (*UBC*) of the centroid-slices, which are shown below:

$$LBC(j)=\begin{cases}1+\left\lceil\frac{|CD(V_q)-r|}{CR_j/\beta}\right\rceil, & if\ CD(V_q)+r<CR_j\ and\ CD(V_q)>r\\ 1+\left\lceil\frac{|CD(V_q)+r|}{CR_j/\beta}\right\rceil, & if\ CD(V_q)+r<CR_j\ and\ CD(V_q)\le r\\ 1, & if\quad CD(V_q)+r\ge CR_j\end{cases} \tag{13}$$

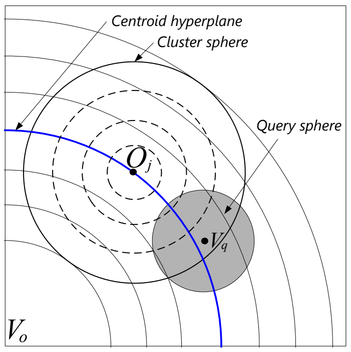
$$UBC(j)=\begin{cases}1+\left\lceil\frac{CD(V_q)-r}{CR_j/\beta}\right\rceil, & if\ CD(V_q)-r>0\\ \beta, & if\ CD(V_q)-r\le 0\end{cases} \tag{14}$$

As an example, Figure 8 shows a cluster sphere $\Theta(O_j,CR_j)$ which is divided into 5 centroid-slices and 8 start-slices. The two

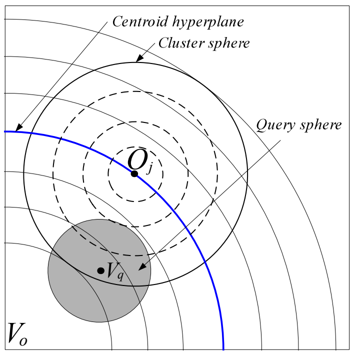
[Ψ] The cluster the V_{new} belongs to is identified by choosing the minimal distance between V_{new} and the cluster centroid. In most cases, the cluster radius need not to be changed since the new point almost falls in the corresponding cluster. Otherwise, the cluster radius needs to be enlarged.



a. $SD(V_q)-r \geq SD(O_j)$



b. $SD(V_q)-r < SD(O_j) < SD(V_q)+r$



c. $SD(V_q)+r \leq SD(O_j)$

Figure 9. The three cases in terms of the position of two spheres

affected start-slices that are closest or farthest to the boundary of $\Theta(O_j,CR_j)$ are the 1st and 3 rd start-slices respectively, denoted as LBS(j)=1, UBS(j)=3. Similarly, we have LBC(j)=1, UBC(j)=3.

3.6.2 *k*-NN Algorithm

Before presenting our *k*-NN algorithm under the EHD-Tree scheme, we first introduce some important routines. Routine ***RSearch*(*V_q,r*)** is the main range search algorithm which returns the candidate data points of range search with centre V_q and radius *r*. ***USearch*(*V_q,r*)** and ***LSearch*(*V_q,r*)** are for the implementation of the range search. ***Farthest***(*S*, V_q) returns the data point which is the farthest from V_q in the candidate data point set *S*. ***BRSearch***(*left, right, j*) is a standard B$^+$-tree range search function in the *j*-th sliced index.

As the cluster spheres in high-dimensional spaces are sliced equally in two different manners (viz., *start-distance-based* and *centroid-distance-based*), there are three cases with respect to the relative position of the query sphere and its intersected cluster sphere, as discussed below.

Case 1: $SD(V_q)-r \geq SD(O_j)$

In this case, as shown in Figure 9(a), the query sphere $\Theta(V_q,r)$ is above the blue bold line[1]. The search range [r_1, r_2] can be derived based on the following:

$r_1 = SCALE\_1 + c \times LBS(j) + LBC(j) + \frac{CD(V_q)-r}{MCD}$, and

$r_2 = SCALE\_1 + c \times UBS(j) + UBC(j) + \frac{CR_j}{MCD}$;

Case 2: $SD(V_q)-r < SD(O_j) < SD(V_q)+r$

In this case, as shown in Figure 9(b), when the query sphere $\Theta(V_q,r)$ intersects with the blue bold line, the EHD-Tree index needs to be visited twice since for the candidate data points in two different parts (i.e., above the blue bold line and below the blue bold line) of the cluster sphere $\Theta(O_j,CR_j)$, the value range of their index keys is different and discontinuous. As a result, we conclude that the two sub ranges [r_{11}, r_{12}] and [r_{21}, r_{22}] for search can be determined by the following:

$r_{11} = SCALE\_1 + c \times LBS(j) + LBC(j) + \frac{CD(V_q)-r}{MCD}$,

$r_{12} = SCALE\_1 + c \times UBS(j) + UBC(j) + \frac{CR_j}{MCD}$; and

$r_{21} = SCALE\_2 + c \times LBS(j) + LBC(j) + \frac{CD(V_q)-r}{MCD}$,

$r_{22} = SCALE\_2 + c \times UBS(j) + UBC(j) + \frac{CR_j}{MCD}$

Case 3: $SD(V_q)+r \leq SD(O_j)$

In this case, as shown in Figure 9(c), the query sphere is below the blue bold line, so the search range [r_1, r_2] is determined by the following:

$r_1 = SCALE\_2 + c \times LBS(j) + LBC(j) + \frac{CD(V_q)-r}{MCD}$, and

$r_2 = SCALE\_2 + c \times UBS(j) + UBC(j) + \frac{CR_j}{MCD}$.

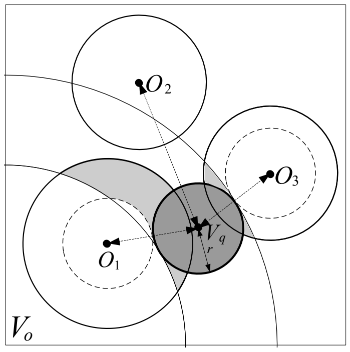


Figure 10. Search regions for *k*-NN query of EHD-Tree

We are now at the position to present our *k*-NN algorithm under the EHD-Tree mechanism. In Figure 10, the deep shaded circle represents a query sphere $\Theta(V_q,r)$ with V_q as the centre and *r* as the radius and the light shaded region (i.e., *search region*) should be checked. As shown in Figure 11, the whole search process is performed in three steps: first, when a user submits a query data point V_q, the search starts with a small radius, and step by step, the radius is increased to form a bigger query sphere iteratively (line3). Once the number of candidate data points is larger than *k*, the search stops, and the (|*S*|-*k*-1) data points which are farthest to the query one are identified (lines 6-7) and removed from the intermediate answer set *S* (line8). Note that the symbol |*S*| denotes the total number of candidate points in *S*, and the candidate points in *S* are the points whose distances to the query point V_q are less than or equal to the query radius *r*. In this way, the *k* nearest

[1] The blue bold line in Figure 9 refers to the centroid-hyperplane of $\Theta(O_j, CR_j)$.

neighbor data points of V_q are returned.

```
Algorithm 4. kNN Search
Input: a query data point $V_q$, $k$, $\Delta r$
Output: query results $S$
1.  $r \leftarrow 0$, $S \leftarrow \Phi$;                        /* initialization */
2.  while ($|S|<k$)
3.      $r \leftarrow r+\Delta r$;
4.      $S \leftarrow$ RSearch($V_q$,$r$);
5.      if ($|S|>k$) then
6.          for count:=1 to $|S|-k-1$ do
7.              $V_{far} \leftarrow$ Farthest($S$, $V_q$);
8.              $S \leftarrow S - V_{far}$;
9.          end for
10.     end if
11. end while

RSearch($V_q$,$r$)
12. $S1 \leftarrow \Phi$, $S2 \leftarrow \Phi$;                    /* initialization */
13. for each cluster sphere $\Theta(O_j, CR_j)$ do
14.     if $\Theta(O_j, CR_j)$ dose not intersect with $\Theta(V_q, r)$ then
15.         break;
16.     else
17.         if $SD(V_q) - r \geq SD(O_j)$ then          /* above the blue bold line */
18.             $S2 \leftarrow$ USearch($V_q$,$r$, $j$);
19.         else if $SD(V_q) + r \leq SD(O_j)$ then     /* below the blue bold line */
20.             $S2 \leftarrow$ LSearch($V_q$,$r$, $j$);
21.         else                                   /* intersects with the blue bold line */
22.             $S2 \leftarrow$ USearch($V_q$,$r$, $j$);
23.             $S2 \leftarrow S2 \cup$ LSearch($V_q$,$r$, $j$);
24.         end if
25.         $S1 \leftarrow S1 \cup S2$;
26.         if $\Theta(O_j, CR_j)$ contains $\Theta(V_q, r)$ then end loop;
27.     end if
28. end for
29. return S1;                                  /* return candidate data points */

USearch($V_q$,$r$, $j$)
30. $left \leftarrow SCALE\_1 + c \times LBS(j) + LBC(j) + (CD(V_q) - r)/MCD$;
31. $right \leftarrow SCALE\_1 + c \times UBS(j) + UBC(j) + CR_j/MCD$;
32. $S3 \leftarrow$ BRSearch[$left$, $right$, $j$];
33. for each data point $V_i \in S3$ do
34.     if $d(V_q, V_i) > r$ then $S3 \leftarrow S3 - V_i$;      /* $V_i$ is removed from S3 */
35. end for
36. return S3;

LSearch($V_q$,$r$, $j$)
37. $left \leftarrow SCALE\_2 + c \times LBS(j) + LBC(j) + (CD(V_q) - r)/MCD$;
38. $right \leftarrow SCALE\_2 + c \times UBS(j) + UBC(j) + CR_j/MCD$;
39. $S3 \leftarrow$ BRSearch[$left$, $right$, $j$];
40. for each data point $V_i \in S3$ do
41.     if $d(V_q, V_i) > r$ then $S3 \leftarrow S3 - V_i$;      /* $V_i$ is removed from S3 */
42. end for
43. return $S3$;
```

Figure 11. *k*-NN search algorithm

4. THEORETICAL ANALYSIS

As mentioned in Section 3.6, a *k*-NN query is completed through iteratively performing a range search. To have a basic understanding on the performance of our EHD-Tree indexing vis-à-vis other existing indexing schemes like NB-tree [18] and iDistance [19], we investigate and compare these schemes through their search regions involved in range search with a same radius for the three index schemes. Additionally, we give a proof in the later part of this section that EHD-Tree will not introduce false dismissals.

For NB-tree, assume the range search is of centre V_q and radius r, then according to [18], the search region involved can be derived as:

$$\overline{\Theta(V_o, SD(V_q) - r)} \cap \Theta(V_o, SD(V_q) + r) \tag{15}$$

where $\overline{\bullet}$ means the complementary space of that $\bullet$ stands for.

Similar to NB-tree, the search region of iDistance [19] is as follows:

$$\sum\nolimits_{j=1}^{t} \left(\overline{\Theta(O_j, CD(V_q) - r)} \cap \Theta(O_j, CR_j) \right) \tag{16}$$

As discussed in Section 3.3, our EHD-Tree is based on the NB-tree and iDistance, therefore the search region of it is the intersection of the two search regions of these two methods (NB-tree and iDistance). Without loss of generality, for the range search of EHD-Tree, assume that the query sphere $\Theta(V_q,r)$ intersects with t' cluster spheres, and $t' \leq t \leq T$ since the number of cluster spheres intersected with the query sphere may be larger or equal to t due to the variation of α. Note that in iDistance's range search with the same radius r, t refers to the number of cluster spheres intersected with the query one. Additionally, for EHD-Tree, the ID numbers of the low bound and the upper bound of start-slices in the j-th cluster sphere are determined by Eqs. (9-12) respectively. The search region of EHD-Tree (cf. Figure 10) can thus be represented as follows:

$$\sum_{i=1}^{t_1} \left(\overline{\Theta(V_O, \sigma(O_i))} \cap \Theta(V_O, \delta(O_i)) \cap \gamma(i) \right) + \sum_{j=1}^{t_2} \left(\overline{\Theta(V_O, \delta(j))} \cap \Theta(V_O, \sigma(j)) \cap \gamma(j) \right) +$$
$$\sum_{k=1}^{t_3} \left(\overline{\Theta(V_O, \sigma(k))} \cap \Theta(V_O, SD(O_k)) \cap \overline{\Theta(V_O, SD(O_k))} \cap \Theta(V_O, \delta(k)) \cap \gamma(k) \right) \tag{17}$$

where $\sigma(i) = \frac{2CR_i}{\alpha} \times LBS(i) + SD(O_i) - CR_i$, $\delta(i) = \frac{2CR_i}{\alpha} \times UBS(i) + SD(O_i) - CR_i$, $\gamma(i) = \overline{\Theta(O_i, CD(V_q) - r)} \cap \Theta(O_i, CR_i)$. $t' = t_1 + t_2 + t_3$, where t_1, t_2 and t_3 are the numbers of cluster spheres which $\Theta(V_q,r)$ is below, above and intersecting with their corresponding centroid hyperplanes respectively.

COROLLARY 1. *For a range query with centre V_q and radius r, when α is large enough, the search region of EHD-Tree is the intersection of the search regions of the other two indexing methods (i.e., iDistance and NB-tree).*

PROOF: Based on Eq. (17) which determines the search region of EHD-Tree with centre V_q and radius r, we can get the following:

$$\begin{aligned} &\lim_{\alpha\to\infty} \left(\frac{2CR_j}{\alpha} \times UBS(j) + SD(O_j) - CR_j \right) \\ &= \lim_{\alpha\to\infty} \left\{ \frac{2CR_j}{\alpha} \times \left(\frac{\alpha}{2} - \left\lceil \frac{SD(V_q) + r - SD(O_j)}{2CR_j/\alpha} \right\rceil \right) + SD(O_j) - CR_j \right\} \\ &= SD(V_q) + r \end{aligned} \tag{18}$$

Similarly we can get the following equation:

$$\lim_{\alpha\to\infty} \left(\frac{2CR_j}{\alpha} \times LBS(j) + SD(O_j) - CR_j \right) = SD(V_q) - r \tag{19}$$

By combining Eqs. (17) and (19), we get the final search region of EHD-Tree as shown below:

$$\begin{aligned} &\lim_{\alpha\to\infty} \begin{pmatrix} \sum_{i=1}^{t_1} \left(\overline{\Theta(V_O, \sigma(O_i))} \cap \Theta(V_O, \delta(O_i)) \cap \gamma(i) \right) + \sum_{j=1}^{t_2} \left(\overline{\Theta(V_O, \delta(j))} \cap \Theta(V_O, \sigma(j)) \cap \gamma(j) \right) \\ \sum_{k=1}^{t_3} \left(\overline{\Theta(V_O, \sigma(k))} \cap \Theta(V_O, SD(O_k)) \cap \overline{\Theta(V_O, SD(O_k))} \cap \Theta(V_O, \delta(k)) \cap \gamma(k) \right) \end{pmatrix} \\ &= \sum_{i=1}^{t} \begin{pmatrix} \overline{\Theta(V_O, SD(V_q) - r)} \cap \Theta(V_O, SD(V_q) + r) \cap \\ \overline{\Theta(O_i, CD(V_q) - r)} \cap \Theta(O_i, CR_i) \end{pmatrix} \end{aligned} \tag{20}$$

For the same range search, the search regions of NB-tree and iDistance can be determined according to Eqs. (15) and (16) respectively. It is obvious that when the number of start-slices(α) is large enough, the search region of EHD-Tree is the intersection of these two search regions, which is much smaller than that of NB-tree and iDistance individually. □

THEOREM 1. *Given a query point V_q and a query radius r, let S1 be the candidate answer set obtained by the sequential scan, S2 be the candidate point set obtained by iDistance, S3 be the candidate point set obtained by NB-tree and S4 be the candidate point set obtained by EHD-Tree. Then we have (1). $S3 \supseteq S4 \supseteq S1$; (2). $S2 \supseteq S4 \supseteq S1$; (2). $S4=S2 \cap S3$. In other words, for EHD-Tree, there are no false dismissals to be introduced.*

PROOF. (1). Given a query sphere $\Theta(V_q,r)$, the search regions of NB-tree and iDistance are represented by Eq. (15) and Eq. (16) respectively. Based on the analysis of the two Equations, we observe that these two search regions both contain $\Theta(V_q,r)$. That is to say, no false dismissals are introduced by either of the two index schemes (i.e., iDistance and NB-tree). Furthermore, based on the COROLLARY 1, when the number of start-slices (α) is large enough, the search region of EHD-Tree is the intersection of its two counterparts generated by NB-tree and iDistance. In other words, the candidate point set generated by EHD-Tree is the intersection of the two candidate point sets generated by iDistance and NB-tree. Therefore, we can derive that (1). $S3 \supseteq S4 \supseteq S1$; (2). $S2 \supseteq S4 \supseteq S1$; (3). $S4=S2 \cap S3$. All of these mean that there are no false dismissals to be introduced by EHD-Tree. □

5. EXPERIMENTAL RESULTS

In this section, we report an empirical performance study conducted to extensively evaluate the effectiveness of EHD-Tree, and we compare it with the following competitive techniques: X-Tree, iDistance, NB-Tree and VA-file.

5.1 Experiment Setup

We have implemented EHD-Tree, NB-Tree, iDistance, VA-file and X-tree in C language and used B^+-tree as the single dimensional index structure. It is worth mentioning that as empirically testified in [19], iDistance is better than M-Tree [14] and Omni-family [17] for various data distributions. So we only use iDistance for comparison with EHD-Tree. All the experiments are executed on a Pentium IV CPU at 2.0GHz with 256 Mbytes memory and index page size is fixed to 4096 Bytes.

To verify the effectiveness of EHD-Tree, three types of data set have been used as experimental data:

(1) The real data set comprises of the Color Histogram dataset available from the *UCI KDD Archive* [20], containing 32-dimensional image features extracted from 68,040 images. All the data values of each dimension are normalized to the range of [0,1].

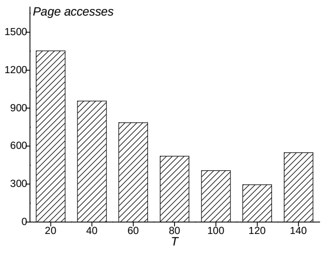
(2) The uniform data set is a random point set consisting of the points distributed uniformly in the range of [0,1] in each dimension. In our evaluation, the size of the data set is 100,000 with its dimensionalities ranging from 8 to 64.

(3) The clustered data set is a synthetic data set whose default number of clusters is 30 and the cluster centers are randomly generated in each cluster: the data follows the normal distribution with the default standard deviation of 0.05. The data size and its dimensionalities are the same as that of the uniform dataset.

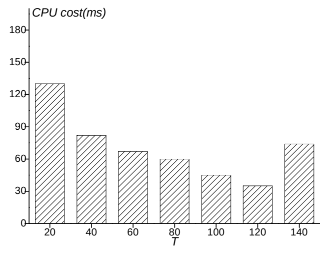
In our evaluation, we use the number of page accesses and the CPU time as the performance metrics. All the experimental performances are measured in terms of the average disk page access, as well as the CPU time over various types of 100 queries. For VA-file, no buffer is used and the number of bits per dimension is set 8.

5.2 Effect of *T*

In the first experiment, we study the effect of the number of clusters (*T*) on the efficiency of range search by using the 100, 000 16-D uniform dataset. Figures 12(a) and 12(b) show that with the increase of *T*, the query efficiency (including the I/O and CPU cost) first increases gradually since the average search region is reducing as the number of clusters increases. Once *T* exceeds a threshold (e.g., 120), the significant overlaps of different cluster spheres lead to the high cost of I/O and CPU in the search. Therefore we should treat *T* as a tuning factor. Empirically, *T* can be set 120 as the optimal number of clusters.



(a). *T vs.* I/O cost



(b). *T vs.* CPU cost

Figure 12. Effect of *T* on *k*-NN search

5.3 Comparison of Two Partition Methods

In this experiment, we give a performance comparison between the two partition methods (viz., *Global Partition* and *Local Partition*). Figure 13 shows that the query efficiency for the global partition method is better than that of the local partition method when the number of centroid-slices is not too large. This is because for the local partition, the number of centroid-slices in every cluster is identical. It will definitely result in a poor pruning effect for some larger cluster spheres. With the increase of the centroid-slices, the gap between the two partition methods becomes reduced, and so much so the pruning effectiveness of both methods converges and moves towards the same.

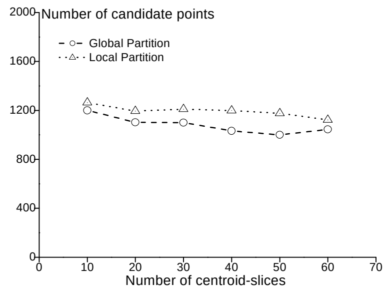


Figure 13. Global Partition *vs.* Local Partition

5.4 Effect of α

We use α to denote the number of start-slices in a cluster sphere. In this evaluation, we use the real data to study the effect of α on the efficiency of range search with an identical search radius (e.g., 0.25). Figure 14 illustrates that the number of candidate points by EHD-Tree is decreasing gradually as α increases. That is to say, the search efficiency of EHD-Tree can not improve anymore when α exceeds a threshold, (e.g., α=40). This is because with the increase of α, the number of "slices" in a cluster sphere increases too. The difference between the actual search region and the ideal search space as identified in Eq.(14) is getting smaller and smaller as α increases. The efficiency of EHD-Tree does not improve anymore once α reaches an optimal value. It is interesting to note that the search efficiency of EHD-Tree is better than that of iDistance and NB-tree no matter what value α is of, and the search efficiency of EHD-Tree does not increase anymore when $\alpha \geq 40$. Therefore, α is also a turning factor to the search optimization. Based on our experiments, we set α to 40 as the optimal value.

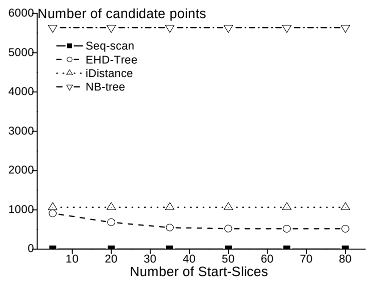
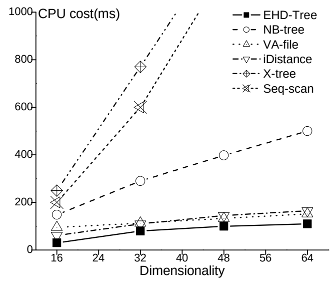


Figure 14. Effect of α

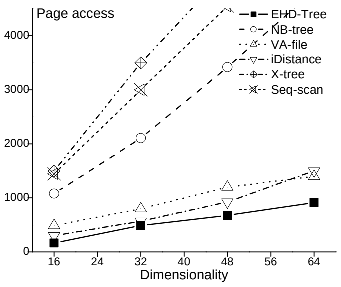
5.5 Effect of Dimensionality

Now we proceed to study the effect of the dimensionality on the performance of 10-NN search using 100,000 clustered dataset with the dimensionality ranging from 16 to 64. In the following experiments, the number of clusters is set 120, the number of start-slices is 40 and the global partition scheme is accommodated.

Figure 15 compares the query performance differences in terms of CPU and I/O cost as the dimensionality increases. It is shown that EHD-Tree outperforms the other methods since EHD-Tree can more effectively reduce the search region than other methods. As shown in Figure 15(a), both the X-tree and sequential scan perform poorly in terms of CPU cost. It is observable that the performance gap in the CPU cost between VA-file, iDistance, NB-tree and EHD-Tree becomes larger as the dimensionality keeps increasing. As shown in Figure 15(b), the I/O performance for the NB-tree degrades due to many additional internal nodes at the leaf-level to be accessed and the large number of false positives. Although iDistance involves clustering and partitioning which helps prune faster and access less I/O pages, the gap between the I/O cost of iDistance and that of EHD-Tree becomes slightly bigger as dimensionality increases, since it is hard for iDistance to effectively prune the search region by only using a single distance.



(a). CPU Cost *vs*. Dimensionality



(b). I/O Cost *vs*. Dimensionality

Figure 15. Effect of the dimensionality

5.6 Effect of Data Size

In this experiment, we use the three types of data set mentioned above to measure the performance behaviors of 10-NN queries with varying number of data points. Exactly 100 various types of 10-NN queries are performed over the real data whose cardinalities ranges from 10,000 to 50,000, and the uniform data and clustered data sets ranging from 20,000 to 100,000 respectively, with the same dimensionality (e.g., 32) for all cases.

Figure 16 shows the performance of query processing in terms of CPU cost. It is evident that EHD-Tree outperforms the other five methods in terms of the CPU cost for the three datasets. It is interesting to note that the performance gap between the tree-based method such as the X-tree and other four techniques (viz., EHD-Tree, NB-tree, iDistance and VA-file) becomes larger since it is a CPU-intensive operation during query process. We also notice that the X-tree is faster than sequential scan in the real data set due to its skewness. For the uniform and clustered datasets, the CPU cost of the five methods increase almost linearly as the dataset size increases, but the increase for X-tree is the fastest. As shown in Figure 16(b)(c), the CPU performance of the X-tree is better than that of sequential scan when the data size is small, and gets slightly worse than the sequential scan when the size of data points becomes large.

In Figure 17, the query performance evaluations with respect to the I/O cost are conducted. The experimental result reveals that EHD-Tree yields consistent performance gain and is more efficient than other methods. EHD-Tree is not very sensitive to the data volume since the increase in data size does not increase the height of B^+-tree substantially, and it can more effectively than others filter away points that are not in the answer set. It is important to mention that the gap between the I/O cost of NB-tree and that of VA-file widens as the data size increases. The I/O cost of NB-tree grows exponentially with the increase of the dataset size, and is also slightly better than the sequential scan for the uniform dataset since it almost scans all of the leaf-level nodes of the whole index. While the I/O cost of iDistance and VA-file also increases with the increase of the data size, it grows at a slower rate in which for the uniform dataset, we could see that iDistance and VA-file follows closely.

5.7 Performance Behavior of k-NN Search

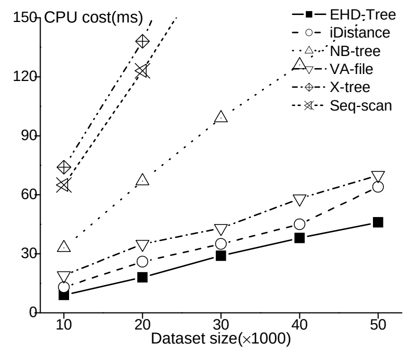
In this series of experiments, we still use the real data set (68,040 points), uniform data set (100,000 points) and clustered data set (100,000 points) in the experiments to test the effect of increasing the value of *k* in the *k*-NN search. Note that the dimensionality of these types of data is fixed to 32.

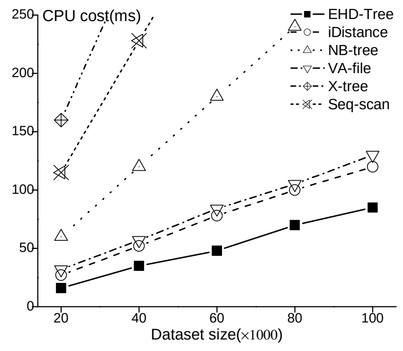
In Figure 18, the CPU cost of EHD-Tree is better than that of other methods for all data sets, especially when the query range is small (i.e., *k* is small). Among these indexes, the CPU cost of the X-tree is still the most expensive. The performances of VA-file and iDistance are pretty close to each other, and the gap between these two techniques with respect to CPU time becomes smaller as *k* increases. Based on the results of the above experiments, it is clear that the pruning effectiveness of EHD-Tree benefits more from the skewness of the real and clustered data sets.

Figure 19 demonstrates the experimental results in terms of I/O cost when *k* ranges from 10 to 100. EHD-Tree performs the best by a wide margin in terms of page access. The I/O costs of iDistance and VA-file are closely similar and NB-tree exhibits a dramatic increase in terms of page access and it finally exceeds the X-tree when *k* is 45 for the uniform data set, as shown in Figure 19(b). Figure 19 also shows an interesting scenario that the gap between EHD-Tree and iDistance is reducing with the increase of *k*. This is because the search region may cover the whole cluster when the *k* (i.e., *query radius*) is increasing.

5.8 Effect of Dynamic Insertion

In this experiment, we investigate the effect of dynamic insertion on our EHD-Tree indexing method. We first adopt a synthetic 32-dimensional clustered data set with 100,000 points to construct EHD-Tree by using 80% (80,000) of the data. We run


(a). real dataset


(b). uniform dataset

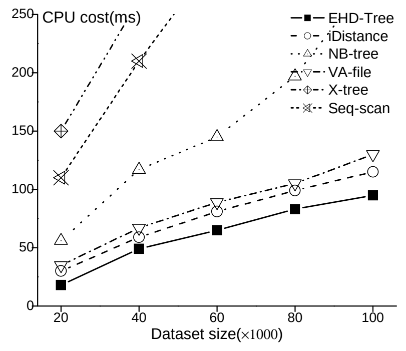
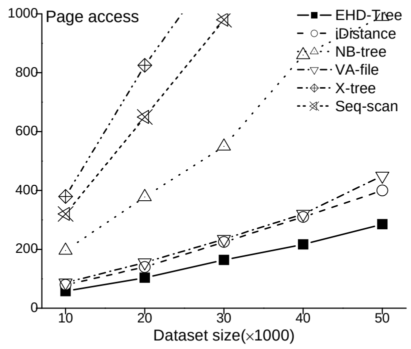
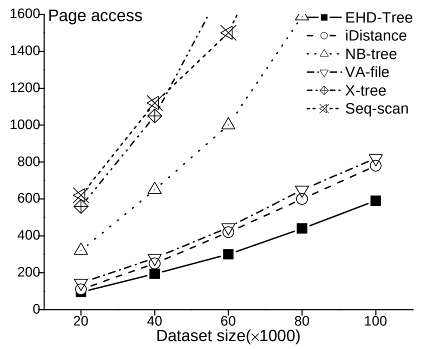

(c). clustered dataset

Figure 16. CPU Cost *vs.* Dataset Size


(a). real dataset


(b). uniform dataset

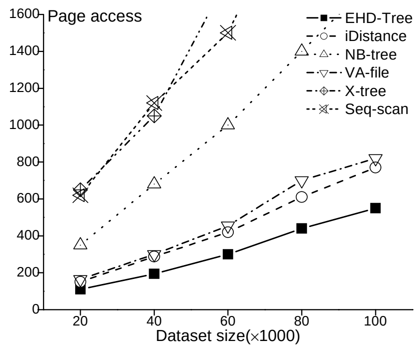
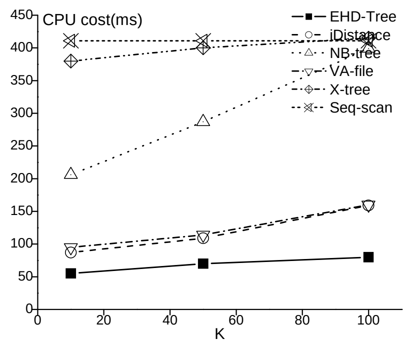
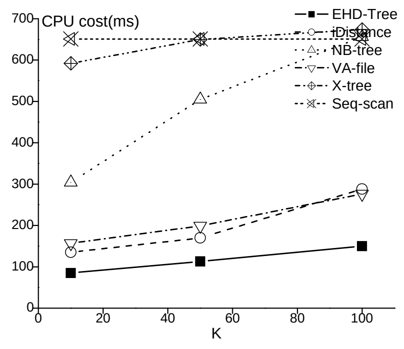

(c). clustered dataset

Figure 17. I/O Cost *vs.* Dataset Size


(a). real dataset


(b). uniform dataset

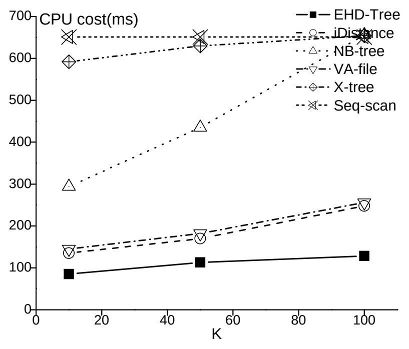
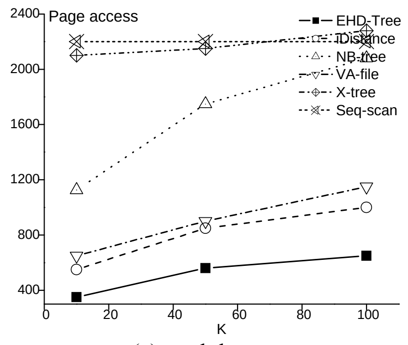
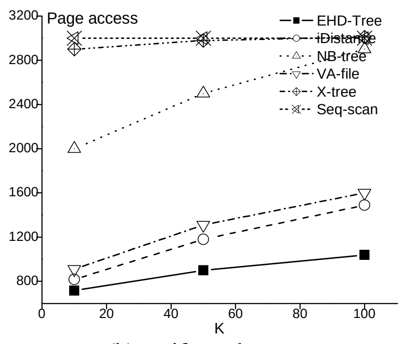

(c). clustered dataset

Figure 18. *K vs.* CPU Cost


(a). real dataset


(b). uniform dataset

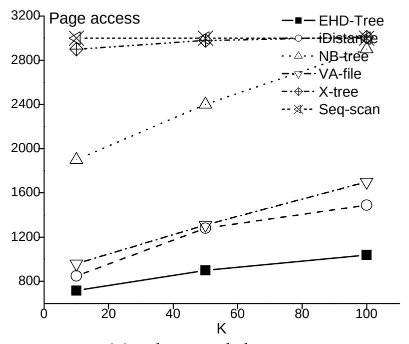

(c). clustered dataset

Figure 19. *K vs.* I/O Cost

some range queries randomly generated and record the average total response time. Then we insert another 5% of the data into the database and re-run the same queries. This process is repeated until the remaining 20% of the data all gets inserted. Meanwhile, we run the queries on the EHD-Tree built when there are 85%, 90%, ... of the data available, which corresponds to the optimal case of EHD-Tree with no updates. We compare the total response time of the two cases as shown in Figure 20. As expected, the difference between them becomes larger as more data points are inserted, but the largest difference is within 20% even for the case of having 20% of newly inserted data. Assuming that the number of clusters is optimal at first, as more data points are inserted or deleted from the data set, the value of *T* may not be optimal anymore. Hence the performance more or less gets degraded. Considering that EHD-Tree typically outperforms other techniques by a factor of 1 to 2, this deterioration is acceptable. Experiments on 10-NN queries exhibit similar results.

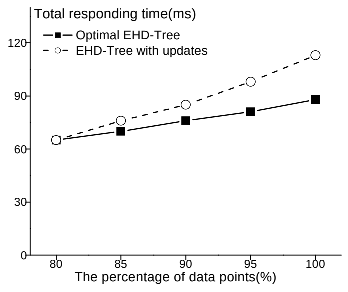


Figure 20. Effect of Dynamic Insertion

6. CONCLUSION

In this paper, we have presented a novel high-dimensional indexing scheme, which we term as *symmetrical Encoding-based Hybrid-Distance-Tree* (EHD-Tree). Three main steps are taken in building an EHD-Tree: first, all (*n*) data points are grouped into *T* clusters by using a *k*-Means algorithm. Second, the uniform ID number of each point is obtained through a small encoding effort, in which each cluster sphere is partitioned twice according to the dual distances of *start*- and *centroid-distance*. Finally, the uniform index key of each point is derived by combining its uniform ID with its centroid-distance together, and is indexed by a partitioned B^+-tree. Compared with the four most competitive techniques including iDistance, NB-Tree, X-tree, VA-file, both theoretical analysis and experimental studies have shown that EHD-Tree can more effectively prune the search space especially when the query radius is relatively small.

ACKNOWLEDGEMENTS

This work is partially supported by the key program of National Natural Science Foundation of China(No.60533090), National Science Fund for Distinguished Young Scholars (No.60525108), 973 Program (No.2002CB312101), Science and Technology Project of Zhejiang Province (2005C13032,2005C11001-05) and China-America Academic Digital Library Project.

REFERENCES

[1] Christian Böhm, Stefan Berchtold, Daniel Keim. Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases, *ACM Computing Surveys,* 2001. 33 (3).

[2] Bentley JL. Multidimensional binary search trees used for associative searching, *Communications of the ACM*, 18(9): pp. 509−517, 1975.

[3] A. Guttman, R-tree: A dynamic index structure for spatial searching, In *Proceedings of the ACM SIGMOD Conference*, pp.47-54, 1984.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, In *Proceedings of ACM SIGMOD Conference,* pp. 322-331, 1990.

[5] King-Ip Lin, H.V. Jagadish and Christos Faloutsos, The TV-tree an index structure for high-dimensional data, VLDB Journal, 1994.

[6] S. Berchtold, D.A. Keim and H.P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22th VLDB Conference*, pp. 28-37, 1996.

[7] D. A. White and R. Jain. Similarity Indexing with the SS- tree, In *Proceedings of ICDE Conference*, pp. 516-523, 1996.

[8] N. Katamaya and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of ACM SIGMOD Conference*, pp. 32-42. 1997.

[9] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th VLDB Conference*, pp. 194-205, 1998.

[10] S. Berchtold, C. Bohm, H.P. Kriegel, J. Sander, and H.V. Jagadish. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proceedings of the 16th ICDE Conference*, pp. 577-588. 2000.

[11] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proceedings of VLDB Conference*, pp. 516–526, 2000.

[12] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín, Searching in Metric Spaces, *ACM Computing Surveys*: 33(3), pp. 273-321, ACM Press, 2001.

[13] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of ACM SIGMOD Conference*, pages 357-368. 1997.

[14] P.Ciaccia, M. Patella, and P. Zezula. M-trees: An efficient access method for similarity search in metric space. In *Proceedings of the 23rd VLDB Conference*, pages 426-435. 1997.

[15] S. Berchtold, C. Bohm, and H.-P. Kriegel. The pyramid technique: Towards breaking the curse of dimensionality. In *Proceedings of SIGMOD Conference*, 1998.

[16] Traina Jr., C., Traina, A., Seeger, B., Faloutsos, Slim-trees: High Performance Metric Trees Minimizing Overlap Between Nodes, In *Proceedings of the EDBT Conference*, Konstanz, Germany, 2000.

[17] Filho, R. F. S., Traina, A., and Faloutsos, C. Similarity search without tears: The Omni family of all-purpose access methods. In *Proceedings of ICDE Conference*, pp. 623–630. 2001

[18] M J. Fonseca and J A. Jorge. Indexing High-dimensional Data for Content-Based Retrieval in Large Databases. In *Proceedings of the 8th DASSFA Conference*, Kyoto, Japan, pp. 267-274, 2003.

[19] H.V. Jagadish, B.C. Ooi, K.L. Tan, C. Yu, R. Zhang. iDistance: An Adaptive B^+-tree Based Indexing Method for Nearest Neighbor Search., *ACM Transactions on Data Base Systems*, 2005. 30(2), pp. 364-397.

[20] *UCI KDD Archive*, http://www.kdd.ics.uci.edu, 2002.