Science in China Series F: Information Sciences

© 2008 SCIENCE IN CHINA PRESS

Springer

www.scichina.com
info.scichina.com
www.springerlink.com

An encoding-based dual distance tree high-dimensional index

ZHUANG Yi[1], ZHUANG YueTing[2†] & WU Fei[2]

[1] College of Computer Science & Information Engineering, Zhejiang Gongshang University, Hangzhou 310018, China;
[2] College of Computer Science, Zhejiang University, Hangzhou 310026, China

The paper proposes a novel symmetrical encoding-based index structure, which is called EDD-tree (for encoding-based dual distance tree), to support fast *k*-nearest neighbor (*k*-NN) search in high-dimensional spaces. In the EDD-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 EDD-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.

high-dimensional indexing, centroid-distance, start-distance, *k*-nearest neighbor search

1 Introduction

With the explosive increase of multimedia data on the Internet, content-based multimedia retrieval has become more important than ever before. To support the highly efficient retrieval process, it is critical to devise an effective high-dimensional index mechanism since multimedia objects are represented by high-dimensional vectors. However, due to the well known "curse of dimensionality"[1], traditional indexing methods only work well in low dimensional spaces[1]. There is a long stream of research for addressing the high-dimensional indexing problems. Existing techniques can be divided into three main categories.

Received October 24, 2006; accepted November 30, 2007; published online August 8, 2008
doi: 10.1007/s11432-008-0104-3

†Corresponding author (email: yzhuang@zju.edu.cn)

Supported by the key program of the National Natural Science Foundation of China (Grant No. 60533090), the National Natural Science Fund for Distinguished Young Scholars (Grant No. 60525108), and China-America Academic Digital Library Project

The first category is based on data and space partitioning, hierarchical tree index structure (e.g., the R-tree[2] and its variants[3,4]), etc. Although these methods generally perform well at low dimensionality, their performance deteriorates rapidly as the dimensionality increases and the degradation can be so bad that sequential scanning becomes more efficient due to the "curse of dimensionality".

The second category is to represent original feature vectors using smaller, approximate representations (e.g., VA-file[5] and IQ-tree[6]), etc. They accelerate the sequential scan by using data compression. Although they reduce the number of disk accesses, it incurs higher computational cost to decode the bit-strings.

The final category is the distance-based high-dimensional indexing schemes, such as NB-tree[7] and iDistance[8]. NB-tree is a single reference point-based scheme, in which high-dimensional points are mapped to a single-dimensional point 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 cannot significantly prune the search region; especially when the dimensionality is becoming larger. 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. In the worst case, the query sphere will intersect with all the partitions due to that the query radius covers all the high-dimensional spaces.

In this paper, we propose a high-dimensional indexing scheme based on the technique of encoding-based dual distance tree, called EDD-tree, to support progressive *k*-NN search. Specifically, in the EDD-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. Thus, 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 EDD-tree indexing facility. Compared with NB-tree[7] and iDistance[8], the EDD-tree can reduce the search region and improve the search performance.

2 EDD-tree

2.1 Problem definition & motivations

Definition 1 (start distance). Given a data point V_i, the start distance (SD for short) of V_i is the distance between it and the origin V_o, formally defined as: $\mathrm{SD}(V_i)=d(V_i, V_o)$, where $V_o=(0,0,\ldots,0)$.

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.

Definition 3 (cluster sphere). Given a cluster C_j, its corresponding cluster sphere is denoted as $\Theta(O_j, \mathrm{CR}_j)$, where O_j is the centroid of cluster C_j, and CR_j is the cluster radius.

Definition 4 (centroid distance). Given a data point V_i, its centroid distance is defined as the distance between itself and the cluster centroid O_j, which is denoted as: $\mathrm{CD}(V_i)=d(V_i, O_j)$, where $i \in [1, |C_j|]$ and $j \in [1, T]$.

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 ith 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
$\Theta(V_q, r)$	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 $\bullet$

The design of EDD-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, compared with the single-distance-metrix-based indexing methods, the search region can be further reduced by using dual metrics only using a single distance metrix.

2.2 Symmetrical DDE scheme

Definition 5 (start-slice). Given a cluster sphere $\Theta(O_j, \mathrm{CR}_j)$, the γth start-slice of it is denoted as $\mathrm{SS}(\gamma, j)$, which is based on the value of start-distance, where $\gamma \in [1, \alpha]$ and j is the ID number of the cluster sphere.

Given a data point $V_i \in \Theta(O_j, \mathrm{CR}_j)$, there are two cases to be considered in terms of the start distance. (i) When $\mathrm{SD}(V_i) \in [\mathrm{SD}(O_j), \mathrm{SD}(O_j)+\mathrm{CR}_j]$: as shown in the shaded part of the cluster sphere in Figure 1, the ID number of start-slice is decreasing with the increase of the start distance of the point in $\Theta(O_j, \mathrm{CR}_j)$. (ii) When $\mathrm{SD}(V_i) \in [\mathrm{SD}(O_j)-\mathrm{CR}_j, \mathrm{SD}(O_j)]$: as shown in the white part of the cluster sphere in Figure 1, the ID number of start-slice is increasing with the increase of the start distance of the point in $\Theta(O_j, \mathrm{CR}_j)$.

Definition 6 (centroid-slice). Given a cluster sphere $\Theta(O_j, \mathrm{CR}_j)$, the μth centroid-slice of it is denoted as $\mathrm{CS}(\mu, j)$, 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 shows an example encoding scheme of centroid-slice in $\Theta(O_j, \mathrm{CR}_j)$, in which the ID number of centroid-slice decreases with the increase of the centroid distance of the point in $\Theta(O_j, \mathrm{CR}_j)$.

Specifically, all data points are first grouped into T clusters by using a k-means clustering algorithm, for each cluster sphere, it can be equally partitioned into α start-slices and β centroid-slices, where $\alpha/2+1=\beta$. Then a data point V_i can be modeled as a four-tuple:

$$V_i :: \langle i, \mathrm{CID}, \mathrm{SD\_ID}, \mathrm{CD\_ID} \rangle, \qquad (1)$$

where i is the ID number of V_i, CID is the cluster ID number which V_i belongs to, SD_ID is the slice ID 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, $j \in [1,T]$.

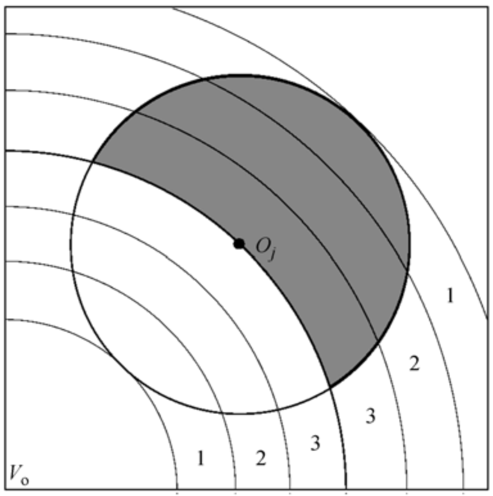

Figure 1 An example of start-slice.

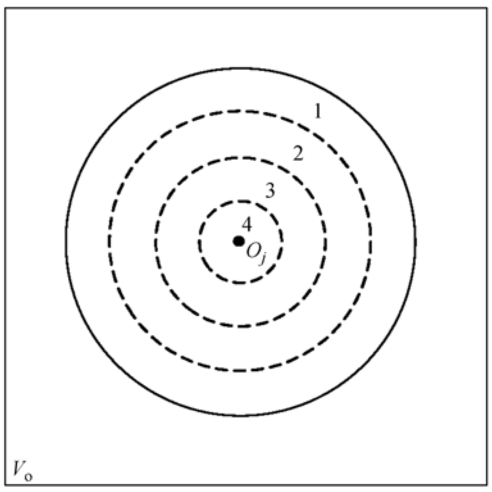

Figure 2 An example of centroid-slice.

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

$$\mathrm{SD\_ID}(V_i) = \frac{\alpha}{2} - \left\lceil \frac{|\mathrm{SD}(V_i) - \mathrm{SD}(O_j)|}{2CR_j/\alpha} \right\rceil,$$
$$\mathrm{CD\_ID}(V_i) = \left\lceil \frac{\mathrm{CR}_j - \mathrm{CD}(V_i)}{\mathrm{CR}_j/\beta} \right\rceil + 1, \tag{2}$$

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:

$$\begin{aligned}\mathrm{UID}(V_i) &= c\times \mathrm{SD\_ID}(V_i) + \mathrm{CD\_ID}(V_i) \\ &= c\times\left[\frac{\alpha}{2} - \left\lceil \frac{|\mathrm{SD}(V_i) - \mathrm{SD}(O_j)|}{2\mathrm{CR}_j/\alpha} \right\rceil + 1\right] + \left\lceil \frac{\mathrm{CR}_j - \mathrm{CD}(V_i)}{\mathrm{CR}_j/\beta} \right\rceil + 1,\end{aligned} \tag{3}$$

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

Based on this encoding scheme, we can get the uniform encoding identifier of each data point by double slicing the cluster sphere. Figure 3 illustrates an example of the encoding scheme, in which the number of start-slices is 6 and the number of centroid-slices is 4. Figure 5 shows the algorithmic description of this scheme.

2.3 The data structure

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

$$\begin{aligned}\mathrm{key}(V_i) &= \mathrm{UID}(V_i) + \mathrm{CD}(V_i)/\mathrm{MCD} \\ &= c\times \mathrm{SD\_ID}(V_i) + \mathrm{CD\_ID}(V_i) + \mathrm{CD}(V_i)/\mathrm{MCD}.\end{aligned} \tag{4}$$

Since CD(V_i) may be larger than 1, the value of CD(V_i) needs to be normalized into the range of [0, 1] through dividing by the maximal CD(MCD)[1)] which is a constant. Thus, it is guaranteed that

1) 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 $\sqrt{d}$.

the search range of the uniform ID and centroid-distance of each point will not be overlapping.

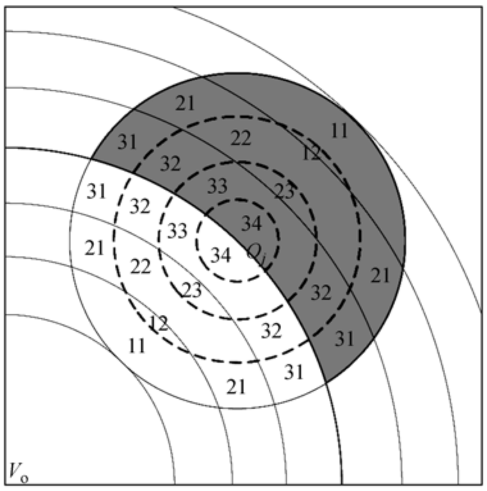

Figure 3 An example of encoding scheme.

```
Algorithm 1. The dual-distance-driven encoding
Input: $\Omega$: the data point set;
    $\alpha$: the number of start-slices in each cluster sphere;
    $\beta$: 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 $\Omega$ are grouped into $T$ clusters using $k$-means clustering algorithm;
2. for $j$:=1 to $T$ do
3.     for $V_i \in \Theta(O_j, \mathrm{CR}_j)$ do
4.         the centroid distance and start distance of it are computed;
5.         the ID numbers of the start- and centroid-slices where $V_i$ belongs to is identified by eq. (2);
6.         the uniform ID of $V_i$ (DDE($V_i$)) can be derived by eq. (3);
7.     end for
8. end for.
```

Figure 4 The encoding algorithm.

Eq. (4) shows the index key expression of V_i. However, due to 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 1, such two data points should satisfy the criteria that: $(\mathrm{SD}(O_j)-\mathrm{SD}(V_i))\times(\mathrm{SD}(O_j)-\mathrm{SD}(V_j)) < 0$. 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:

$$\mathrm{Ukey}(V_i)=\begin{cases}\mathrm{SCALE\_1}+\mathrm{key}(V_i), \text{ if } \mathrm{SD}(V_i)\in[\mathrm{SD}(O_j),\mathrm{SD}(O_j)+\mathrm{CR}_j];\\ \mathrm{SCALE\_2}+\mathrm{key}(V_i), \text{ if } \mathrm{SD}(V_i)\in[\mathrm{SD}(O_j)-\mathrm{CR}_j,\mathrm{SD}(O_j));\end{cases}$$

$$\mathrm{key}(V_i)=\begin{cases}\mathrm{SCALE\_1}+c\times\left(\dfrac{\alpha}{2}-\left\lceil\dfrac{|\mathrm{SD}(V_i)-\mathrm{SD}(O_j)|}{2\mathrm{CR}_j/\alpha}\right\rceil\right)+\left\lceil\dfrac{\mathrm{CR}_j-d(V_i,O_j)}{\mathrm{CR}_j/\beta}\right\rceil+1+\dfrac{\mathrm{CD}(V_i)}{\mathrm{MCD}},\\ \qquad\text{if } \mathrm{SD}(V_i)\in[\mathrm{SD}(O_j),\mathrm{SD}(O_j)+\mathrm{CR}_j];\\ \mathrm{SCALE\_2}+c\times\left(\dfrac{\alpha}{2}-\left\lceil\dfrac{|\mathrm{SD}(V_i)-\mathrm{SD}(O_j)|}{2\mathrm{CR}_j/\alpha}\right\rceil\right)+\dfrac{\mathrm{CR}_j-d(V_i,O_j)}{\mathrm{CR}_j/\beta}+1+\dfrac{\mathrm{CD}(V_i)}{\mathrm{MCD}},\\ \qquad\text{if } \mathrm{SD}(V_i)\in[\mathrm{SD}(O_j)-\mathrm{CR}_j,\mathrm{SD}(O_j)).\end{cases} \tag{5}$$

Finally, the n index keys obtained by eq. (5) are inserted into a partition-based B^+-tree, as discussed next. Figure 5 shows the detailed steps of constructing an EDD-tree index. Note that the routine TransValue(V_i) is a distance transformation function as given in eq. (5), and BInsert (dist, bt) is a standard B^+-tree insert procedure.

```
Algorithm 2.    EDD-tree index construction
Input:   $\Omega$: the data point set;
Output: bt(1 to $T$): the index for EDD-tree;
1.  The data points in $\Omega$ are grouped into $T$ clusters using $k$-means clustering algorithm;
2.  for $j$:=1 to $T$ do
3.      bt($j$)$\leftarrow$newEDDFile();    *create index header file*/
4.      for each data point $V_i \in \Theta(O_j, \mathrm{CR}_j)$ do
5.          the centroid distance and start distance of each data point are computed;
6.          the ID number of $V_i$ is obtained according to algorithm 1;
7.          Key($V_i$)=TransaValue($V_i$);
8.          BInsert(Key($V_i$), $bt(j)$);    /* insert it to $B^+$-tree*/
9.      end for
10.     return bt($j$)
11. end for.
```

Figure 5 The EDD-tree index construction algorithm.

3 *k*-NN search with EDD-tree

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

Definition 7 (low bound of start-slice). Given two intersected spheres $\Theta(V_q, r)$ and $\Theta(O_j, \mathrm{CR}_j)$, the low bound of start-slice in $\Theta(O_j, \mathrm{CR}_j)$ is the affected start-slice which is the closest to the boundary of $\Theta(O_j, \mathrm{CR}_j)$ in terms of start distance, denoted as LBS(j).

Definition 8 (upper bound of start-slice). Given two intersected spheres $\Theta(V_q, r)$ and $\Theta(O_j, \mathrm{CR}_j)$, the upper bound of start-slice in $\Theta(O_j, \mathrm{CR}_j)$ is the affected start-slice which is the farthest to the boundary of $\Theta(O_j, \mathrm{CR}_j)$ in terms of start distance, denoted as UBS(j).

Definition 9 (low bound of centroid-slice). Given two intersected spheres $\Theta(V_q, r)$ and $\Theta(O_j, \mathrm{CR}_j)$, the low bound of centroid-slice in $\Theta(O_j, \mathrm{CR}_j)$ is the affected centroid-slice which is the farthest to the cluster centre O_j, denoted as LBC(j).

Definition 10 (upper bound of centroid-slice). Given two intersected spheres $\Theta(V_q, r)$ and $\Theta(O_j, \mathrm{CR}_j)$, the upper bound ID of centroid- slice in $\Theta(O_j, \mathrm{CR}_j)$ is the affected centroid-slice which is the closest to the cluster centre O_j, denoted as UBC(j).

Now we focus on how to get the LBS, UBS, LBC and UBC. For a query sphere $\Theta(V_q, r)$ in Figure 6, the corresponding search range of its start-distance is [SD(V_q)–r, SD(V_q)+r]. Similarly, the search range of the centroid-distance is [CD(V_q)–r, CD(V_q)+r]. As mentioned in section 2.2, a cluster sphere (e.g., $\Theta(O_j, \mathrm{CR}_j)$) 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)$\leqslant$UBS(j)) may be affected (intersected). Therefore we can derive the ID number of the low bound start-slice (LBS) and upper bound centroid-slice (UBS) in the jth cluster sphere by the process described immediately below.

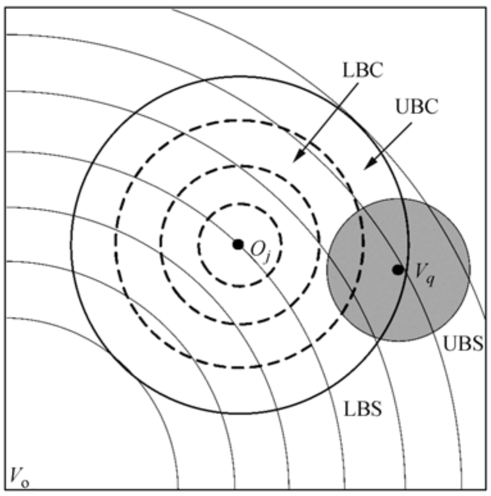

Figure 6 The choice of LBS, UBS, LBC and UBC *of* $\Theta(V_q, r)$ in $\Theta(O_j,\mathrm{CR}_j)$.

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

$$\mathrm{LBS}(j)=\begin{cases}\left\lceil \dfrac{\mathrm{SD}(V_q)-r-\mathrm{SD}(O_j)+\mathrm{CR}_j}{2\mathrm{CR}_j/\alpha}\right\rceil+1, & \text{if } \mathrm{SD}(O_j)-\mathrm{CR}_j<\mathrm{SD}(V_q)-r<\mathrm{SD}(O_j)+\mathrm{CR}_j;\\ 1, & \text{if } \mathrm{SD}(V_q)-r\leqslant \mathrm{SD}(O_j)-\mathrm{CR}_j;\end{cases} \tag{6}$$

$$\mathrm{UBS}(j)=\begin{cases}\left\lceil \dfrac{\mathrm{SD}(V_q)+r-\mathrm{SD}(O_j)+\mathrm{CR}_j}{2\mathrm{CR}_j/\alpha}\right\rceil+1, & \text{if } \mathrm{SD}(V_q)+r<\mathrm{SD}(O_j)+\mathrm{CR}_j;\\ \alpha, & \text{if } \mathrm{SD}(V_q)+r\geqslant \mathrm{SD}(O_j)+\mathrm{CR}_j.\end{cases} \tag{7}$$

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:

$$\mathrm{UBC}(j)=\begin{cases}\left\lceil \dfrac{\mathrm{CD}(V_q)-r}{2\mathrm{CR}_j/\beta}\right\rceil+1, & \text{if } 0<\mathrm{CD}(V_q)-r<\mathrm{CR}_j,\\ 1, & \text{if } \mathrm{CD}(V_q)-r\leqslant 0,\end{cases} \tag{8}$$

$$\mathrm{LBC}(j)=\begin{cases}\left\lceil \dfrac{\mathrm{CD}(V_q)+r}{2\mathrm{CR}_j/\beta}\right\rceil+1, & \text{if } \mathrm{CD}(V_q)+r<\mathrm{CR}_j,\\ \beta, & \text{if } \mathrm{CD}(V_q)+r\geqslant \mathrm{CR}_j.\end{cases} \tag{9}$$

As an example, Figure 6 shows a cluster sphere $\Theta(O_j, \mathrm{CR}_j)$ which is divided into 4 centroid-slices and 6 start-slices. The two affected start-slices that are the closest or the farthest to the boundary of $\Theta(O_j, \mathrm{CR}_j)$ are the 1st and the 3rd start-slices respectively, denoted as LBS(j)=1, UBS(j)=3. Similarly, we have LBC(j)=1, UBC(j)=2.

Figure 7 shows the whole search process. The k-NN search starts with a small radius, and step by step, the radius is increased to form a bigger query sphere iteratively (line 3). Once the number of candidate data points is larger than k, the search stops, and the ($|S|-k-1$) data points which are

```
**Algorithm 3.**   $k$-NN 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.              $V_{\text{far}}\leftarrow$Farthest($S$, $V_q$);
8.              $S\leftarrow \bar{S}V_{\text{far}}$;
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 $\Theta(O_j, \text{CR}_j)$ do
14.     if $\Theta(O_j, \text{CR}_j)$ dose not intersect with $\Theta(V_q, r)$ then
15.         break;
16.     else
17.         if $\text{SD}(V_q)-r \geqslant \text{SD}(O_j)$ then     /*above the blue line*/
18.             $S2\leftarrow$USearch($V_q$, $r$, $j$);
19.         else if $\text{SD}(V_q)+r \leqslant \text{SD}(O_j)$ then   /*below the blue line*/
20.             $S2\leftarrow$LSearch($V_q$, $r$, $j$);
21.         else                           /*intersects with the blue 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 $\Theta(O_j, \text{CR}_j)$ 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. $\text{left}\leftarrow \text{SCALE\_1}+c\times\text{LBS}(j)+\text{LBC}(j)+(d(V_q,O_j)-r)/\text{MCD}$;
31. $\text{right}\leftarrow \text{SCALE\_1}+c\times\text{UBS}(j)+\text{UBC}(j)+\text{CR}_j/\text{MCD}$;
32. $S3\leftarrow$BRSearch[left, right, $j$];
33. for each 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. return $S3$;

LSearch($V_q$, $r$, $j$)
36. $\text{left}\leftarrow \text{SCALE\_2}+c\times\text{LBS}(j)+\text{LBC}(j)+(d(V_q,O_j)-r)/\text{MCD}$;
37. $\text{right}\leftarrow \text{SCALE\_2}+c\times\text{UBS}(j)+\text{UBC}(j)+\text{CR}_j/\text{MCD}$;
38. $S3\leftarrow$BRSearch[left, right, $j$];
39. for each point $V_i\in S3$ do
40.     if $d(V_q, V_i)>r$ then $S3\leftarrow S3-V_i$;    /*$V_i$ is removed from $S3$*/
41. return $S3$.
```

Figure 7 k-NN search algorithm.

the farthest to the query one are identified (lines 6—7) and removed from the intermediate answer set S (line 8). 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 neighbor data points of V_q are returned. It is worth mentioning that 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 jth sliced index.

4 Theoretical analysis

As mentioned in section 3, a k-NN query is completed through iteratively performing a range search. For simplicity, we investigate and compare the EDD-tree with other two competitors such as NB-tree[7] and iDistance[8] based on a range query with the same radius for the three index schemes.

4.1 Search region

For NB-tree, assume the range search is of centre V_q and radius r, then according to ref. [7], the search region involved can be derived as

$$\overline{\Theta\left(V_{\mathrm{o}},\mathrm{SD}(V_q)-r\right)}\cap\Theta\left(V_{\mathrm{o}},\mathrm{SD}(V_q)+r\right), \tag{10}$$

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

Similar to NB-tree, the search region of iDistance[8] is as follows:

$$\sum_{j=1}^{t}\left(\overline{\Theta(O_j,d(V_q,O_j)-r)}\cap\Theta(O_j,\mathrm{CR}_j)\right). \tag{11}$$

As discussed in section 2.2, our EDD-tree is based on the NB-tree and iDistance, therefore its search region 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 EDD-tree, it is assumed that the query sphere $\Theta(V_q, r)$ intersects with t cluster spheres, where $t \leqslant T$. When the number of slices is large enough, the search region of EDD-tree can thus be represented as follows:

$$\left(\overline{\Theta(V_{\mathrm{o}},d(V_q,V_{\mathrm{o}})-r)}\cap\Theta(V_{\mathrm{o}},d(V_q,V_{\mathrm{o}})+r)\right)\cap\sum_{j=1}^{t}\left(\overline{\Theta(O_j,d(V_q,O_j)-r)}\cap\Theta(O_j,\mathrm{CR}_j)\right). \tag{12}$$

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 EDD-tree. Then we have 1) $S3\supseteq S4\supseteq S1$; 2) $S2\supseteq S4\supseteq S1$; 3) $S4=S2\cap S3$. In other words, for EDD-tree, there is no false dismissal to be introduced.

Proof. Given a query sphere $\Theta(V_q, r)$, the search regions of NB-tree and iDistance are represented by eqs. (10) and (11) respectively. Based on the analysis of the two equations, we observe that both of search regions contain $\Theta(V_q, r)$. That is to say, no false dismissal is introduced by either of the two index schemes (i.e., iDistance and NB-tree). Furthermore, when the number of start-slices(α) is large enough, the search region of EDD-tree is the intersection of its two counterparts generated by NB-tree and iDistance. In other words, the candidate point set generated by EDD-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 EDD-tree.

4.2 Cost model

Let f be average fanout of a node in B$^+$-tree, h_j be the height of the jth B$^+$-tree (sliced-index), T_S be disk seek time, T_L be disk latency time, T_T be disk transfer time and T_{TOTAL} be total query time.

As mentioned before, we use B$^+$-tree as a basic index structure for EDD-tree. The number of candidate data points in the jth cluster sphere to be searched is

$$\mathrm{NUM}(j) = \frac{\mathrm{Vol}(\Theta(O_j, \mathrm{CR}_j))}{\sum_{i=1}^{T} \mathrm{Vol}(\Theta(O_i, \mathrm{CR}_i))} \times n. \tag{13}$$

Both h_j and NUM(j) should be met in eq. (14) below,

$$f \times (f+1)^{h_j - 1} = \mathrm{NUM}(j). \tag{14}$$

By solving eq. (14), the height of the jth sliced-index (B$^+$-tree) is as follows:

$$h_j = \left\lceil \frac{\lg \mathrm{NUM}(j) - \lg f}{\lg(f+1)} \right\rceil + 1. \tag{15}$$

For a range search in a sliced-index of EDD-tree, the total query cost consists of three parts: the first part is to search from the root node to the leaf node, the second part is the range search among the leaf nodes with a range limit, and the last one is the refinement process for the candidate data points returned by the range search. The total number of leaf nodes in the jth sliced index is $\lceil \mathrm{NUM}(j)/f \rceil$. Thus, for a range search in the jth sliced index, the total number of candidate data points accessed can be derived as follows:

$$\mathrm{num}(j) = \frac{\mathrm{Vol}\Big(\big(\overline{\Theta(V_o, \mathrm{SD}(V_q) - r)} \cap \Theta(V_o, \mathrm{SD}(V_q) + r)\big) \cap \overline{\Theta(O_j, d(V_q, O_j) - r)} \cap \Theta(O_j, \mathrm{CR}_j)\Big)}{\sum_{i=1}^{T} \mathrm{Vol}\big(\Theta(O_i, \mathrm{CR}_i)\big)} \times n, \tag{16}$$

where Vol(•) refers to the volume of •.

Based on the above analysis, the total cost (height + number of leaf nodes + refinement) for a range search across t affected clusters, denoted by T_{FIL}, can be represented below:

$$T_{FIL} = \sum_{j=1}^{t} \left[\left(\left\lceil \frac{\lg \mathrm{NUM}(j) - \lg f}{\lg(f+1)} \right\rceil + 1 + \left\lceil \frac{\mathrm{num}(j)}{f} \right\rceil \right) \times (T_S + T_L + T_T) \right]. \tag{17}$$

Then the total cost of refinement process is a range search in EDD-tree denoted as T_{REF}, which can be calculated as follows:

$$T_{REF} = \sum_{j=1}^{t} \mathrm{num}(j) \times T_C, \tag{18}$$

where T_C is the average CPU cost of the comparison between any two data points.

Now by combining eq. (13) with eq. (18) together, we can get the final range query cost under EDD-tree as

$$T_{TOTAL} = \sum_{j=1}^{t} \left[\left(\left\lceil \frac{\lg \mathrm{NUM}(j) - \lg f}{\lg(f+1)} \right\rceil + 1 + \left\lceil \frac{\mathrm{num}(j)}{f} \right\rceil \right) \times (T_S + T_L + T_T) + T_C \times \mathrm{num}(j) \right]. \tag{19}$$

Eq. (19) shows that the range query cost of the EDD-tree is mainly proportional to the number of data points while it is reciprocal to the number of entries in a node.

5 Experimental results

In this section, we report an empirical performance study conducted to extensively evaluate the effectiveness of EDD-tree, and we compare it with the following competitive techniques: X-tree, iDistance, NB-tree and VA-file. We have implemented EDD-tree, NB-tree, iDistance, VA-file and X-tree in C language and used B^+-tree as the single dimensional index structure. All the experiments are executed on a Pentium IV CPU at 2.0 GHz with 256 Mbytes memory and index page size is fixed to 4096 Bytes.

To verify the effectiveness of EDD-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, containing 32-dimensional image features extracted from 68040 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 10^5 points distributed uniformly in the range of [0, 1] in each dimension. In our evaluation, its dimensionalities range from 8 to 64. In our evaluation, we use the number of page accesses and the CPU time as the performance metrics.

5.1 Effect of the number of dimensions

As the first experiment, we study the effect of the dimensionality on the performance of 10-NN search using 10^5 uniform dataset with the dimensionality ranging from 16 to 64. It is shown from Figure 8 that the EDD-tree outperforms the other methods in terms of CPU and I/O cost as the dimensionality increases. This is because the EDD-tree can more effectively reduce the search region than other methods. 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 EDD-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.

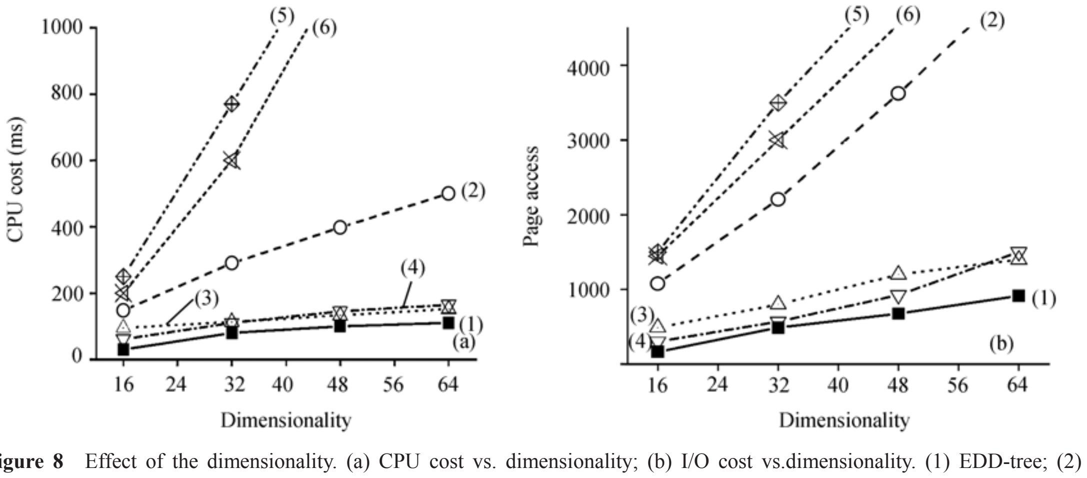

Figure 8 Effect of the dimensionality. (a) CPU cost vs. dimensionality; (b) I/O cost vs.dimensionality. (1) EDD-tree; (2) NB-tree; (3) VA-file; (4) iDistance; (5) X-tree; (6) Seq-scan.

5.2 Effect of data size

In this experiment, we use the two data sets to measure the performance behaviors of 10-NN queries with varying number of data points. Figures 9 and 10 show the query performance of the

EDD-tree is much better than that of other competitors in terms of CPU and I/O cost respectively. The reason is described in section 4.1. It is interesting to note that the performance gap between the VA-file and other techniques such as the EDD-tree and iDistance becomes larger since it is a CPU-intensive operation during query process.

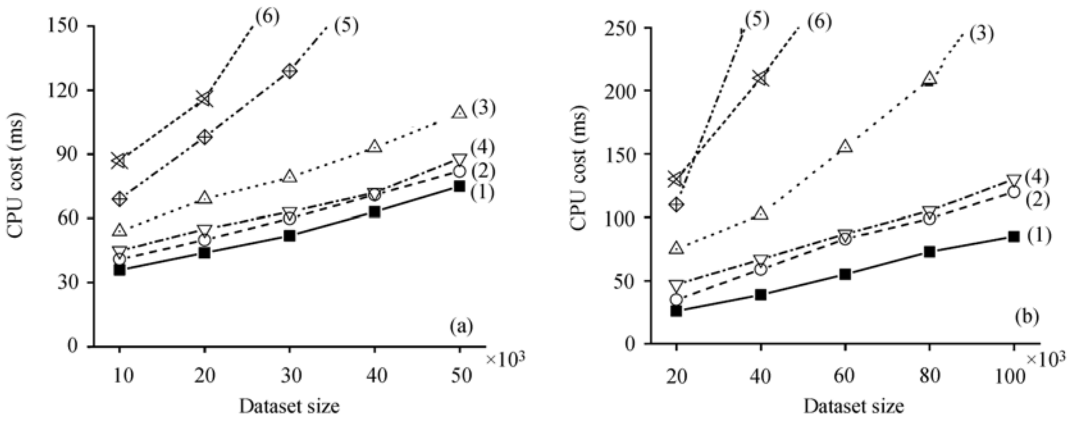

Figure 9 CPU cost vs. dataset size. (a) Real dataset; (b) uniform dataset. (1) EDD-tree; (2) NB-tree; (3) VA-file; (4) iDistance; (5) X-tree; (6) Seq-scan.

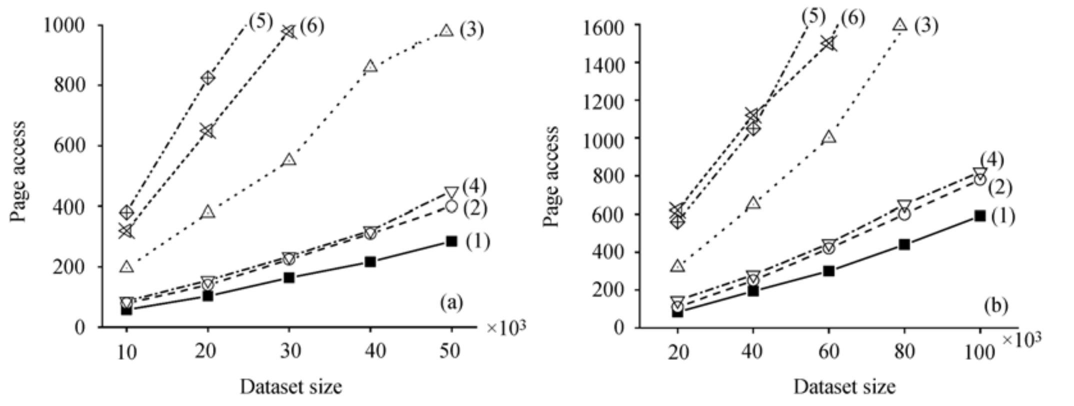

Figure 10 I/O cost vs. dataset size. (a) Real dataset; (b) uniform dataset. (1) EDD-tree; (2) NB-tree; (3) VA-file; (4) iDistance; (5) X-tree; (6) Seq-scan.

5.3 Performance behavior of *k*-NN search

In this series of experiments, we still use the two kinds of data sets 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 Figures 11 and 12, the CPU and I/O costs of EDD-tree are both better than that of other methods for all data sets, when *k* ranges from 10 to 100. 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 and I/O costs becomes smaller as *k* increases.

5.4 Effect of *T* on *k*-NN search

In this experiment, we study the effect of the number of clusters (*T*) on the efficiency of 10-NN search by using the 10^5 16-D uniform dataset. Figure 13 shows that with the increase of *T*, the efficiency of the *k*-NN search (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., 60), the significant overlaps of different cluster spheres lead to the high cost of I/O and CPU in the *k*-NN search. Therefore we should treat *T* as a tuning factor.

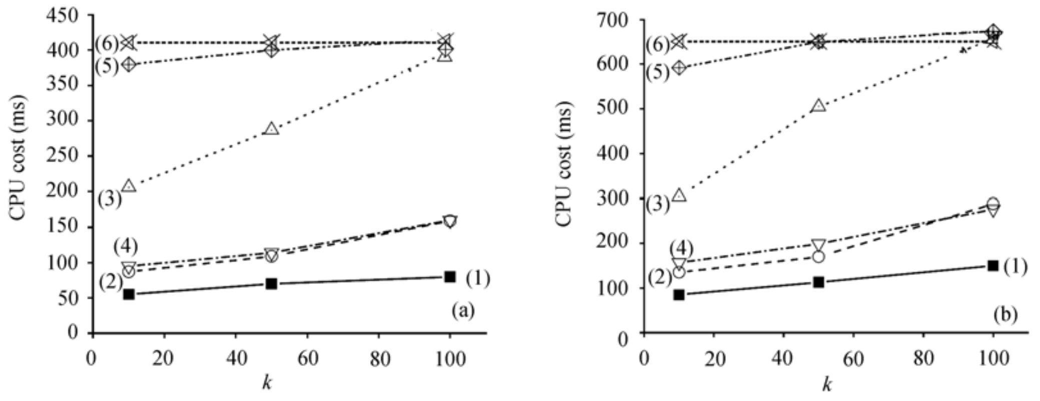

Figure 11 *k* vs. CPU cost. (a) Real dataset; (b) uniform dataset. (1) EDD-tree; (2) NB-tree; (3) VA-file; (4) iDistance; (5) X-tree; (6) Seq-scan.

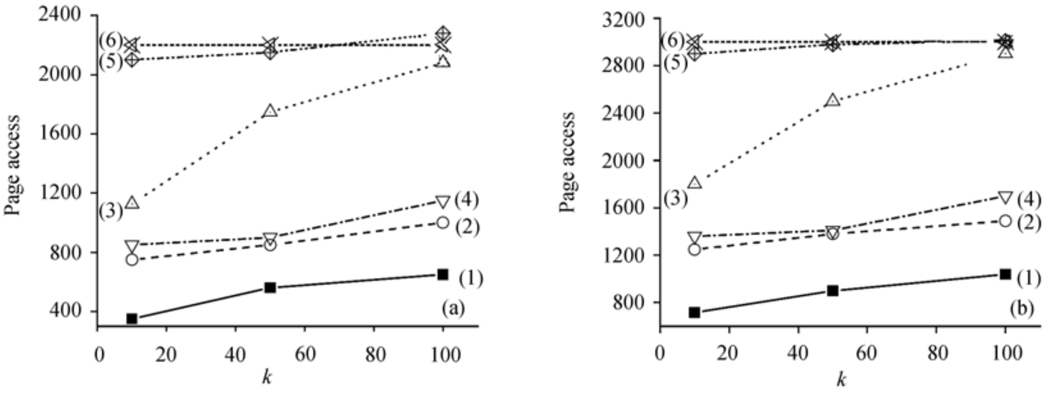

Figure 12 *k* vs. I/O cost. (a) Real dataset; (b) uniform dataset. (1) EDD-tree; (2) NB-tree; (3) VA-file; (4) iDistance; (5) X-tree; (6) Seq-scan.

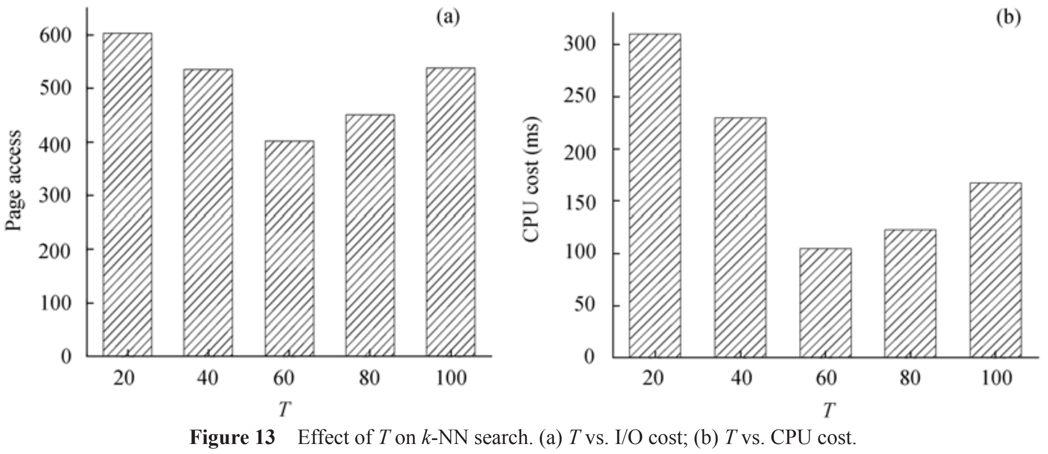

Figure 13 Effect of *T* on *k*-NN search. (a) *T* vs. I/O cost; (b) *T* vs. CPU cost.

6 Conclusion

In this paper, we have presented a novel high-dimensional indexing scheme, which we term as encoding-based dual-distance-tree (EDD-tree). Three main steps are taken in building an EDD-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 partition-based B^+-tree. Both theoretical analysis and experimental studies verify the effectiveness of the EDD-tree.

1 Bohm C, Berchtold S, Keim D. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Comput Surv, 2001, 33(3): 322－373

2 Guttman A. R-tree: a dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. Boston: ACM Press, 1984. 47－54

3 Beckmann N, Kriegel H P, Schneider R, et al. The R*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of ACM SIGMOD International Conference on Management of Data. Atlantic City: SIGMOD Record, 1990, 19(2). 322－331

4 Berchtold S, Keim D A, Kriegel H P. The X-tree: an index structure for high-dimensional data. In: Proceedings of the 22th International Conference on Very Large Data Bases. India: Morgan Kaufmann, 1996. 28－37

5 Weber R, Schek H, Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings of the 24th International Conference on Very Large Data Bases. New York: Morgan Kaufmann Publishers, 1998. 194－205

6 Berchtold S, Bohm C, Kriegel H P, et al. Independent quantization: an index compression technique for high-dimensional data spaces. In: Proceedings of the 16th International Conference on Data Engineering. USA: IEEE Computer Society, 2000. 577－588

7 Fonseca M J, Jorge J A. NB-Tree: an indexing structure for content-based retrieval in large databases. In: Proceedings of the 8th International Conference on Database Systems for Advanced Applications. Kyoto: IEEE Computer Society, 2003. 267－274

8 Jagadish H V, Ooi B C, Tan K L, et al. iDistance: an adaptive B^+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst, 2005, 30(2): 364－397