

# Fast Answering $k$ -Nearest-Neighbor Queries over Large Image Databases Using Dual Distance Transformation

Yi Zhuang and Fei Wu

College of Computer Science, Zhejiang University, Hangzhou, 310027, P.R. China  
{zhuangyi, wufei}@cs.zju.edu.cn

**Abstract.** To support fast  $k$ -NN queries over large image database, in this paper, we propose a novel *dual distance transformation* method called DDT. In DDT, all images are first grouped into clusters by the  $k$ -Means clustering algorithm. Then the *start-* and *centroid-distances* of each image are combined to obtain the uniform index key through its dual distance transformation. Finally the keys are indexed by a  $B^+$ -tree. Thus, given a query image, its  $k$ -nearest neighbor query in high-dimensional space is transformed into a search in a single dimensional space with the aid of the DDT index. Extensive performance studies are conducted to evaluate the effectiveness and efficiency of the proposed scheme. Our 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.

## 1 Introduction

With an explosively increasing number of images on the Internet, content-based image retrieval and indexing have become more important than ever before. However, due to the very known high dimensional curse problem [1], traditional indexing methods such as R-tree [2] and X-tree [4] only work well in low dimensional space. Therefore, the design of efficient indexing techniques for high-dimensional data is still an important and active research area.

In this paper, we propose a high-dimensional image indexing scheme based on the technique of *dual-distance-transformation*, called DDT, to support the progressive  $k$ -NN search over large image databases. Specifically, in DDT, all images are first grouped into clusters by using the well known  $k$ -Means clustering algorithm. Then, the dual distance transformation of each image is performed to obtain its index key which is indexed by a  $B^+$ -tree. Thus, given a query image  $V_q$  and  $k$ , the  $k$ -Nearest Neighbor search of  $V_q$  in a high-dimensional space is transformed into a search in a single dimensional space with the aid of the DDT index. We have implemented the DDT method and conducted an extensive performance study to evaluate its efficiency. Our results show that the proposed technique has superior to X-tree [4], VA-file [5], NB-Tree [7] and iDistance [8], which are also designed for indexing high-dimensional image databases. The primary contributions of this paper are listed as follows:

1. We propose a *dual-distance-transformation*-based indexing method to facilitate the highly efficient  $k$ -NN search for large image databases.

2. We propose a uniform index key expression method, called *the dual-distance-transformation-based method*, which nicely combines the dual distance metrics (i.e., *start-distance* and *centroid-distance*) together.
3. We give a cost model for the proposed indexing method to support the query optimization.

The rest of this paper is organized as follows. Related work is surveyed in Section 2. In Section 3, a *dual-distance-transformation*-based high-dimensional indexing scheme is proposed to dramatically improve the query performance of the  $k$ -NN search. In Section 4, a cost model for DDT is derived to facilitate the query optimization. In Section 5, we performed the extensive experiments to evaluate the efficiency and effectiveness of our DDT index mechanism. Conclusions are given in the final section.

## 2 Related Work

As we know, content-based image indexing belongs to the high-dimensional indexing problem. The state-of-art techniques can be divided into three main categories [1].

The first category is based on data and space partitioning, hierarchical tree index structure, for example, the R-tree [2] and its variants [3][4], etc. Although these methods generally perform well at low dimensionality, their performance deteriorates rapidly as dimensionality increases due to the “dimensionality curse”.

The second category is to represent original feature vectors using smaller, approximate representations, e.g., VA-file [5] and IQ-tree [6], etc. The VA-file [5] accelerates the sequential scan by using data compression. For a search, although VA-file can reduce the number of disk accesses, it incurs higher computational cost to decode the bit-strings. The IQ-tree [6] is also an indexing structure along the lines of the VA-file.

The last category is to use a distance-metric-based method as an alternative direction for high-dimensional indexing, such as NB-tree [7], iDistance [8], etc. NB-tree [7] 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 respectively. Then these distances are indexed using a B<sup>+</sup>-tree on which we perform all subsequent operations. The drawback of NB-Tree is that it can not effectively prune the search region and especially when the dimensionality is becoming larger. iDistance [8] is proposed by selecting some reference points in order to further prune the search region so as to improve the query efficiency. 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 Dual Distance Transformation

In this section, we present a novel high-dimensional indexing technique called the *Dual Distance Transformation* (DDT for short) for speed up the query efficiency over large image database.

### 3.1 Preliminaries

The design of DDT is motivated by the following observations. First, the similarity between images can be derived and ordered based on their distances to a reference image. Second, a distance is essentially a single dimensional value which enables us to reuse existing single dimensional indexing schemes such as B<sup>+</sup>-tree. The list of symbols used in the rest of paper is summarized in Table 1.

**Table 1.** Meaning of Symbols Used

Symbols	Meaning
$\Omega$	a set of images
$I_i$	the $i$ -th image and $I_i \in \Omega$
$d$	the number of dimensions
$n$	the number of images in $\Omega$
$V_q$	a query image user submits
$\Theta(I_q, r)$	the query hypersphere with centre $I_q$ and radius $r$
$\Theta(O_j, CR_j)$	The $j$ -th cluster hypersphere with centre $O_j$ and cluster radius $CR_j$
$d(I_i, I_j)$	the distance between two images

**Definition 1 (START DISTANCE).** Given an image  $I_i$ , the Start Distance (SD for short) of it is the distance between image  $I_i$  and the image  $I_o(0, 0, \dots, 0)$ , formally defined as:

$$SD(I_i) = d(I_i, I_o) \quad (1)$$

Assuming that  $n$  images 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 hyper-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 image 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 hypersphere 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.

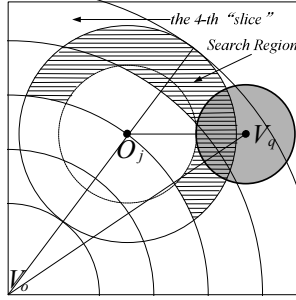
**Definition 3 (CENTROID DISTANCE).** Given an image  $I_i$ , its centroid distance is defined as the distance between itself and the cluster centroid  $O_j$ , which is denoted as:

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

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

### 3.2 The Data Structure

Based on the above definitions, we propose the *dual-distance-transformation*-based high-dimensional indexing scheme. Specifically, all images are first grouped into  $T$



**Fig. 1.** The corresponding “slices” of the cluster hypersphere  $\Theta(O_j, CR_j)$

clusters using a  $k$ -Means clustering algorithm, then the *start distance* and *centroid distance* of each image are computed, thus  $I_i$  can be modeled as a four-tuple:

$$I_i ::= \langle i, CID, SD, CD \rangle \quad (3)$$

where  $i$  refers to the  $i$ -th image and  $CID$  is the ID of the cluster that  $I_i$  belongs to.

However, for each image  $I_i$  in a cluster hypersphere, it is hard to combine the *start-distance* and *centroid-distance* of  $I_i$  to get its uniform index key. To address this problem, the *dual-distance-transformation* approach is proposed to obtain a new index key through “slicing” the cluster hypersphere. As shown in Fig.1, assume that  $\Theta(I_i, r)$  intersects with  $\Theta(O_j, CR_j)$ , we first “slice”  $\Theta(O_j, CR_j)$  equally into  $\lambda$  “pieces” according to the value of the start-distance, where  $\lambda = 4$ . Thus, for an image  $I_i$  in the  $l$ -th “slice” of  $\Theta(O_j, CR_j)$ ,

the range of its start-distance is represented:  $SD(I_i) \in \left[ SD(O_j) - CR_j + \frac{l \times 2CR_j}{\lambda}, SD(O_j) - CR_j + \frac{(l+1) \times 2CR_j}{\lambda} \right]$ , therefore the uniform index key of  $I_i$  can be defined as:

$$key(I_i) = c \times j + l + CD(I_i) / MCD \quad (4)$$

where  $l = \left\lceil \frac{\lambda \times SD(I_i) - SD(O_j) + CR_j}{2CR_j} \right\rceil + 1$  and  $MCD = \sqrt{2}$ .

---

**Algorithm 1. DDT Index Construction**

**Input:**  $\Omega$ : the image set,  $\lambda$ : the number of slices in each cluster hypersphere;

**Output:**  $bt$ : the index for DDT;

1. The images in  $\Omega$  are grouped into  $T$  clusters using  $k$ -Means clustering algorithm
  2.  $bt \leftarrow \text{newDDTFile}()$ ; /\* create index header file \*/
  3. **for**  $j=1$  to  $T$  **do**
  4.   **for** each image  $I_i$  in the  $j$ -th cluster **do**
  5.     the centroid distance and start distance of each image are computed;
  6.     the “slice” in the  $j$ -th cluster that  $I_i$  belongs to is identified;
  7.      $Key(I_i) = \text{TransValue}(I_i, j, l)$ ;
  8.      $\text{BInsert}(Key(I_i), bt)$ ; /\* insert it to B+-tree \*/
  9.   **end for**
  10. **end for**
  11. **return**  $bt$
- 

**Fig. 2.** The index construction algorithm for DDT

### 3.3 Building DDT

Fig. 2 shows the detailed steps of constructing a DDT index. Note that the routine *TransValue*( $I_i, CID, SliceID$ ) is a distance transformation function as given in Eq. (4) and *BInsert*( $key, bt$ ) is a standard B<sup>+</sup>-tree insert procedure.

### 3.4 k-NN Search Algorithm

For  $n$  high-dimensional images,  $k$ -Nearest-neighbor search is the most frequently used search method which retrieves the  $k$  most similar images to a given query image. In this section, we focus on  $k$ -NN search for high-dimensional images.

#### 3.4.1 Picking a Value of LB and UB

Assume that a query hypersphere  $\Theta(I_q, r)$  intersects with a cluster hypersphere  $\Theta(O_j, CR_j)$ , we examine which “slices” in  $\Theta(O_j, CR_j)$  intersects with  $\Theta(I_q, r)$ . Let us first introduce two definitions.

**Definition 4 (LOW BOUND ID OF “SLICE”).** Given two intersected hyperspheres  $\Theta(I_q, r)$  and  $\Theta(O_j, CR_j)$ , the low bound ID of “slice” in  $\Theta(O_j, CR_j)$  is the ID number of the “slice” which is the closest to the origin  $V_o$ , denoted as **LB**( $j$ ).

**Definition 5 (UPPER BOUND ID OF “SLICE”).** Given two intersected hyperspheres  $\Theta(I_q, r)$  and  $\Theta(O_j, CR_j)$ , the upper bound ID of “slice” in  $\Theta(O_j, CR_j)$  is the ID number of the “slice” which is the farthest from the origin  $V_o$ , denoted as **UB**( $j$ ).

For the example shown in Fig. 1, the cluster hypersphere  $\Theta(O_j, CR_j)$  is divided into 4 “slices”, the “slice” that is closest to the origin  $V_o$  is the 3<sup>rd</sup> “slice”, denoted as **LB**( $j$ )=3; similarly, **UB**( $j$ )=4;

Now we focus on the problem of how to get the **LB** and **UB**. Once a query hypersphere intersects with the cluster hypersphere, some continuous “slices” (i.e., from the **LB**( $j$ )-th “slice” to the **UB**( $j$ )-th “slice”, where  $LB(j) \leq UB(j)$ ) may be affected (i.e., being intersected). Assume that the query hypersphere  $\Theta(V_q, r)$  intersects with the cluster hypersphere  $\Theta(O_j, CR_j)$ ,  $\Theta(O_j, CR_j)$  is “sliced” into  $\lambda$  pieces. As mentioned before, the slice ID number in a cluster hypersphere is ordered ascendingly in terms of the start-distance value. So we can get the ID number of the low bound “slice”(i.e., **LB**) and upper bound “slice” (i.e., **UB**) in the  $j$ -th cluster hypersphere as follows:

$$LB(j) = \begin{cases} \left\lceil \frac{(SD(I_q) - r - SD(O_j) + CR_j) \lambda}{2CR_j} \right\rceil + 1, & \text{if } SD(O_j) - CR_j < SD(I_q) - r < SD(O_j) + CR_j \\ 1 & \text{if } SD(I_q) - r \leq SD(O_j) - CR_j \end{cases} \quad (5)$$

$$UB(j) = \begin{cases} \left\lceil \frac{(SD(I_q) + r - SD(O_j) + CR_j) \lambda}{2CR_j} \right\rceil + 1, & \text{if } SD(I_q) + r < SD(O_j) + CR_j \\ \lambda & \text{if } SD(I_q) + r \geq SD(O_j) + CR_j \end{cases} \quad (6)$$

where  $\lceil \bullet \rceil$  is an integral part of  $\bullet$ .

#### 3.4.2 k-NN Algorithm

Fig. 3 illustrates the whole  $k$ -NN search process which contains three steps: first, when a user submits a query image  $I_q$ , the 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 images is larger than  $k$ , the search stops, then the  $(|S| - k - 1)$  images which are farthest to the query one are identified (lines 6-7) and removed from  $S$  (line 8). It is worth mentioning that the symbol  $|S|$  has two meanings: (a). the total number of candidate images in  $S$ ; (b). the candidate images in  $S$  are the images whose distances to the query image  $I_q$  are less than or equal to the query radius  $r$ . In this way, the  $k$  nearest neighbor images of  $I_q$  are returned. Routine **RSearch** ( $I_q, r$ ) is the main range search algorithm which returns the candidate images of range search with centre  $I_q$  and radius  $r$ . **Search** ( $I_p, r$ ) is the implementation of the range search. **Farthest** ( $S, I_q$ ) returns the image which is the farthest from  $I_q$  in the candidate image set  $S$ . **BRSearch**(*left, right*) is a standard  $B^+$ -tree range search function.

---

**Algorithm 3.  $k$ NN Search**
**Input:** query image  $I_q, k, \Delta r$ 
**Output:** query results  $S$ 

```

1.  $r \leftarrow 0, S \leftarrow \Phi;$  /* initialization */
2. while ( $|S| < k$ )
3.    $r \leftarrow r + \Delta r;$ 
4.    $S \leftarrow \mathbf{RSearch}(I_q, r);$ 
5.   if ( $|S| > k$ ) then
6.     for count:=1 to  $|S| - k - 1$  do
7.        $I_{far} \leftarrow \mathbf{Farthest}(S, I_q);$ 
8.        $S \leftarrow S - I_{far};$ 
9.     end for
10.  end if
11. end while

RSearch( $I_q, r$ )
12.  $S1 \leftarrow \Phi, S2 \leftarrow \Phi;$ 
13. for  $j:=1$  to  $T$  do /*  $T$  is the number of clusters */
14.    $S2 \leftarrow \mathbf{Search}(I_q, r, j);$ 
15.    $S1 \leftarrow S1 \cup S2;$ 
16.   if  $\Theta(O_j, CR_j)$  contains  $\Theta(I_q, r)$  then
17.     end loop;
18.   end if
19. end for
20. return  $S1;$  /* return candidate images */

Search( $I_q, r, j$ )
21.  $left \leftarrow c \times j + LB(j) + (d(I_q, O_j) - r) / MCD;$ 
22.  $right \leftarrow c \times j + UB(j) + CR_j / MCD;$ 
23.  $S3 \leftarrow \mathbf{BRSearch}[left, right];$ 
24. for each image  $I_i$  in the candidate images  $S3$  do
25.   if  $d(I_q, I_i) > r$  then
26.      $S3 \leftarrow S3 - I_i;$  /*  $I_i$  is removed from  $S3$  */
27.   end for
28. return  $S3;$ 

```

---

**Fig. 3.**  $k$ -NN search algorithm

## 4 Cost Model

We now derive a cost model for DDT in which query cost is measured by the number of nodes accessed. Some frequently used symbols are shown in Table 2.

**Table 2.** Parameters and their meanings

Symbol	Meaning
$f$	average fanout of a node in $B^+$ -tree
$h$	the height of the $B^+$ -tree
$T_s$	disk seek time
$T_L$	disk latency time
$T_T$	disk transmission time
$T_Q$	total query time

As mentioned before, we use  $B^+$ -tree as a basic index structure for DDT. Both  $h$  and  $n$  should be met in Eq. (7) below.

$$f \times (f + 1)^{h-1} = n \tag{7}$$

By solving Eq. (7), the height of the  $B^+$ -tree is as follows:

$$h = \left\lceil \frac{\lg n - \lg f}{\lg(f + 1)} \right\rceil + 1 \tag{8}$$

For a range search in the  $j$ -th cluster hypersphere, the total number of candidate images accessed can be derived as follows:

$$num(j) = \frac{Vol\left(\left(\overline{\Theta(V_o, SD(V_s) - r)} \cap \Theta(V_o, SD(V_s) + r)\right) \cap \overline{\Theta(O_j, d(V_s, O_j) - r)} \cap \Theta(O_j, CR_j)\right)}{\sum_{i=1}^t Vol(\Theta(O_i, CR_i))} \times n \tag{9}$$

where  $Vol(\bullet)$  refers to the volume of  $\bullet$ .

Therefore once  $t$  cluster hyperspheres are affected, and combining the Eq. (8) with (9) together, the total cost ( *height + number of leaf nodes + refinement*) for a range search in DDT denoted as  $T_Q$ , is calculated as follows:

$$T_Q = \sum_{j=1}^t \left[ \left( \left\lceil \frac{\lg n - \lg f}{\lg(f + 1)} \right\rceil + 1 + \left\lceil \frac{num(j)}{f} \right\rceil \right) \times (T_s + T_L + T_T) + T_c \times num(j) \right] \tag{10}$$

where  $T_c$  is the average CPU cost of the comparison between any two images.

Eq. (10) shows that the range query cost of DDT, which is mainly proportional to the number of images while it is reciprocal to the number of entries in a node. In Eq. (10),  $T$  and  $\lambda$  are two tunable parameters which can affect DDT’s query performance. Therefore the moderate values of  $T$  and  $\lambda$  are critically important to the query optimization.

## 5 Experiments

To demonstrate the practical effectiveness of the new indexing method, we performed an extensive experimental evaluation of the DDT and compared it with the competitors: the X-tree, the VA-file, the iDistance, NB-tree and the linear scan.

We have implemented DDT, NB-Tree, iDistance, VA-file and X-tree in C language and used B<sup>+</sup>-tree as the single dimensional index structure. All the experiments are run on a Pentium IV CPU at 2.0GHz with 256 Mbytes memory and index page size is fixed to 4096 Bytes. The image data set comprises of color histogram information, containing 32-Dimensional image features extracted from 100,000 images which are downloaded by our web crawler randomly from around 5,000 websites. The values of each dimension are normalized to the range of [0,1]. In our evaluation, we use the number of page accesses and the CPU time as the performance metric.

### 5.1 Effect of Data Size

In this experiment, 100 various types of 10-NN queries are performed over the image database whose cardinalities ranges from 20,000 to 100,000 with the same dimensionality. Fig. 4a shows the performance of query processing in terms of CPU cost. It is evident that DDT outperforms the other five methods significantly in terms of the CPU cost. It is interesting to note that the performance gap between the tree-based method such as the X-tree and other four techniques, viz., DDT, NB-tree, iDistance and VA-file becomes larger since it is a CPU-intensive operation during query process. In Fig. 4b, the query performance evaluations with respect to the I/O cost are conducted. The experimental result reveals that DDT yields consistent performance and is more efficient than other methods since the increase in data size does not increase the height of B<sup>+</sup>-tree substantially, and it can more effectively filter away images that are not in the answer set than others.

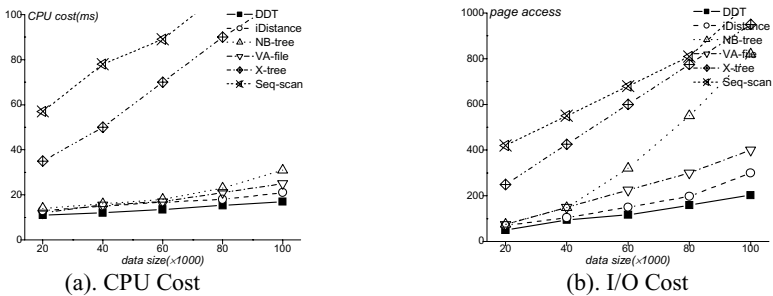


Fig. 4. Effect of Data size

### 5.2 Effect of $k$

In this experiment, we test the effect of  $k$  on the  $k$ -NN search. Fig. 5a demonstrates the experimental results in terms of I/O cost when  $k$  ranges from 10 to 100. DDT performs the best in terms of page access. The I/O costs of iDistance and VA-file are closely similar and NB-tree exhibits a dramatically increase in terms of page access and it finally exceeds the X-tree when  $k$  is 45. In Fig. 5b, the CPU cost of DDT is slightly lower than that of NB-tree for all data sets.

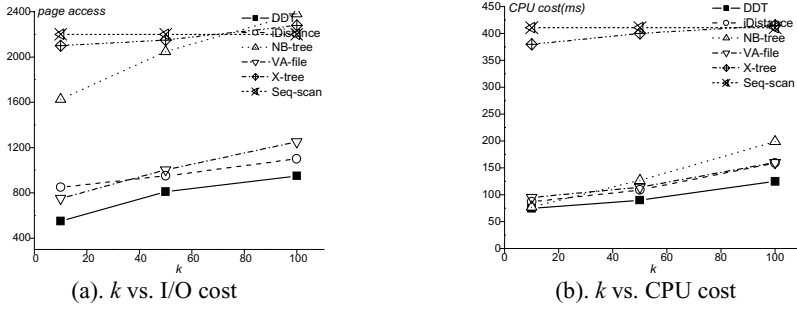


Fig. 5. Effect of  $k$

**5.3 Effect of  $T$  on the Efficiency of  $k$ -NN Search**

In this experiment, we study the effect of the number of clusters( $T$ ) on the efficiency of the  $k$ -NN search. Figs. 6a and 6b illustrate 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, the significant overlaps of different cluster hyperspheres lead to the high cost of I/O and CPU in the  $k$ -NN search. Therefore we should treat  $T$  as a tuning factor.

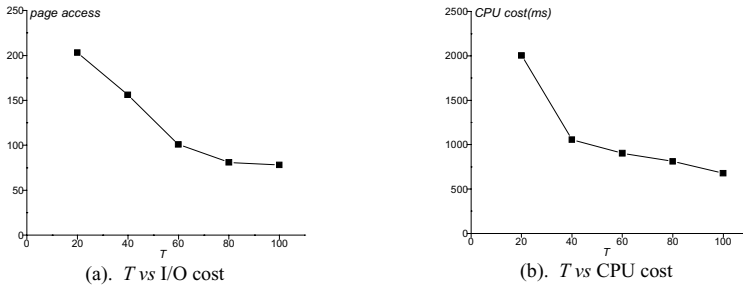


Fig. 6. Effect of  $T$  on the efficiency of  $k$ -NN search

**5.4 Effect of  $\lambda$  on the Efficiency of Range Search**

We use  $\lambda$  to denote the number of “slices” in every cluster hypersphere. In this evaluation, we study the effect of  $\lambda$  on the efficiency of range search with an identical search radius. Fig. 7 illustrates that the number of candidate images by DDT is decreasing gradually as  $\lambda$  increases. That is to say, the search efficiency of DDT could not improve anymore when  $\lambda$  exceeds a threshold, (e.g.,  $\lambda=40$ ). This is because with the increase of  $\lambda$ , the number of “slices” in a cluster hypersphere increases too. The precision of index key is getting smaller and smaller. The efficiency of DDT dose not improve anymore once  $\lambda$  reaches an optimal value. It is interesting to note that the search efficiency of DDT is better than that of iDistance and NB-tree no matter what value  $\lambda$  is of, and the search efficiency of DDT dose not increase anymore when  $\lambda=40$ . Therefore,  $\lambda$  is also a turning factor to the search optimization. Based on our experiments, we set  $\lambda$  to 40 as an optimal value empirically.

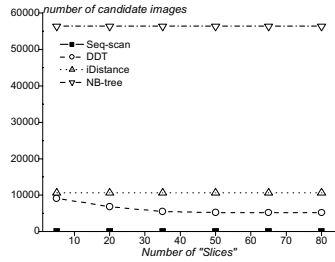


Fig. 7. Effect of  $\lambda$  on the Efficiency of Range Search

## 6 Conclusion

In this paper, we have presented a *dual-distance-transformation*-based high-dimensional image indexing scheme called *DDT*. We have shown by extensive performance studies that the proposed method is more efficient than the four most competitive techniques including *iDistance*, *NB-Tree*, *X-tree* and *VA-file*, in addition to sequential scan. Furthermore, being a  $B^+$ -tree-based index, *DDT* can be crafted easily into an existing database backend or implemented as a stored procedure.

**Acknowledgments.** This work is supported by National Natural Science Foundation of China (No.60533090, No.60525108), 973 Program (No.2002CB312101), Science and Technology Project of Zhejiang Province (2005C13032, 2005C11001-05), and China-US Million Book Digital Library Project ([www.cadal.zju.edu.cn](http://www.cadal.zju.edu.cn)).

## 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] A. Guttman, R-tree: A dynamic index structure for spatial searching, In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1984. pp.47-54.
- [3] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, 1990, The R\*-tree: An Efficient and Robust Access Method for Points and Rectangles, In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 322-331.
- [4] S. Berchtold, D.A. Keim and H.P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. 22th Int. Conf. on Very Large Data Bases*, 1996. pp. 28-37.
- [5] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. on Very Large Data Bases*, 1998. pp. 194-205.
- [6] 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 *Proc. 16th Int. Conf. on Data Engineering*, 2000. pp. 577-588.
- [7] M J. Fonseca and J A. Jorge. NB-Tree: An Indexing Structure for Content-Based Retrieval in Large Databases. In *Proc. of the 8th Int. Conf. on Database Systems for Advanced Applications*, Kyoto, Japan, 2003. pp. 267-274.
- [8] 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*, 30, 2, 2005. pp. 364-397.