Randomly Projected KD-Trees with Distance Metric Learning for Image Retrieval

Pengcheng Wu, Steven C.H. Hoi, Duc Dung Nguyen, Ying He

School of Computer Engineering, Nanyang Technological University, Singapore
{wupe0003, chhoi, nguy0051, yhe}@ntu.edu.sg

Abstract. Efficient nearest neighbor (NN) search techniques for high-dimensional data are crucial to content-based image retrieval (CBIR). Traditional data structures (e.g., kd-tree) usually are only efficient for low dimensional data, but often perform no better than a simple exhaustive linear search when the number of dimensions is large enough. Recently, approximate NN search techniques have been proposed for high-dimensional search, such as Locality-Sensitive Hashing (LSH), which adopts some random projection approach. Motivated by similar idea, in this paper, we propose a new high dimensional NN search method, called Randomly Projected kd-Trees (RP-kd-Trees), which is to project data points into a lower-dimensional space so as to exploit the advantage of multiple kd-trees over low-dimensional data. Based on the proposed framework, we present an enhanced RP-kd-Trees scheme by applying distance metric learning techniques. We conducted extensive empirical studies on CBIR, which showed that our technique achieved faster search performance with better retrieval quality than regular LSH algorithms.

1 Introduction

Similarity search plays an important role for content-based image retrieval (CBIR) systems. The images in CBIR are often represented in high-dimensional space, and the scale of images can be easily over millions or even billions for web-scale applications. These challenges have made CBIR an open challenge although it has been extensively studied for several decades.

The NN search problem for CBIR has been extensively studied in literature. A variety of data structures have been proposed for indexing data points in a low-dimensional space [15, 5, 4, 14]. For example, if data points lie in a plane, it can be shown that traditional data structures, such as kd-tree, can *exactly* solve the NN search problem with $O(\log n)$ time using only $O(n)$ space [15]. However, when the number of dimensions grows, these conventional approaches often become less efficient, a phenomenon known as the *curse of dimensionality*. Specifically, the time or space requirements of these approaches often grow exponentially with the dimensionality. For example, the approach in [5] has a nice query time of $O(d^{O(1)} \log n)$, however, it costs about $O(n^{O(d)})$ space, making it impractical for large applications. While there exist some efficient data structures using only linear or sublinear space [4, 14], the best query time of these approaches is of $O(min(2^{O(d)}, dn)$, which is not better than a simple exhaustive linear search even for moderate dimension d. Until now, researchers have yet to find an efficient

solution that can solve the *exact* high-dimensional NN search problem beyond the exponential dependence on the dimensionality.

Recently, instead of pursuing the *exact* NN search, researchers have attempted to adopt some *approximation* approaches [10, 8, 13, 2] that remove the exponential dependence on dimensionality. The basic idea is that: instead of finding the nearest point p to the query point q, the approximate NN search algorithm allows to return any point within the distance of $(1 + \epsilon)$ times the distance from q to p. Recent studies have shown that by adopting the approximation, the high-dimensional NN search problem can be efficiently resolved by reducing the dependence on the dimensionality from exponential to polynomial complexity. Several recent studies, such as Locality Sensitive Hashing (LSH) [10, 8], have successfully applied the random projection idea for approximate NN search over high-dimensional data.

Motivated by the above results, in this papper, we propose a new method for approximate NN search in high-dimensional space using the random projection principle, called the Randomly Projected kd-trees (RP-kd-Trees). The basic idea is to project the high-dimensional data points into a lower dimensional space and integrate multiple kd-trees by utilizing the advantage of kd-trees for low-dimensional NN search. Besides, to further improve the performance, we also present a machine learning approach to enhancing RP-kd-Trees by applying distance metric learning techniques.

The rest of this paper is organized as follows. Section 2 presents the proposed RP-kd-Trees method, which integrates the Random Projection technique and kd-tree structures in the unified framework to efficiently resolve approximate nearest neighbor search problem. Section 3 discusses the enhanced RP-kd-Trees scheme by applying distance metric learning techniques. Section 4 discusses the experimental results of applying the proposed RP-kd-Trees technique for content-based image retrieval application. Section 5 sets out the conclusion of this work.

2 Randomly Projected KD-Trees

We now present a framework of Randomly Projected KD-trees (RP-kd-Trees) for approximate NN search on high-dimensional data. We will first introduce some relevant techniques, including the basic concept of Random Projection, and the kd-tree data structure, followed by presenting the proposed indexing algorithms.

2.1 Random Projection

Random Projection is a technique to reduce the curse of dimensionality with little lost information of distances between pairs of points in a high-dimensional vector space. In order to project the given points onto a lower dimensional space, we multiple X by a random matrix $M \in \mathbb{R}^{d' \times d}$, where M often consists of multiple elements in normal distribution $N(0, 1)$. By doing this, we speed up the computation and make it possible to use existing data structures to handle the image similarity search problem. We expect that random projection approximately preserves pair-wise distances. We will describe the technique to overcome the accuracy loss caused by random projections in later part of this paper.

Achlioptas [1] proposed sparse random projections by not using the $\mathcal{N}(0, 1)$ elements in M but elements in $\{+1, 0, -1\}$ with probabilities $\{\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\}$, attaining

a threefold speedup in projection processing time. It shows the following theorem for performance assurance [1].

Theorem 1. *Let X be an arbitrary set of n data points in $\mathbb{R}^d$, represented as a matrix $X \in \mathbb{R}^{d \times n}$. Given $\epsilon, \beta > 0$ let $d_0 = \frac{4+2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$, for integer $d' \geq d_0$, let M be a $d' \times d$ random matrix with $M(i,j) = M_{ij}$, where $\{M_{ij}\}$ are independent random variables from the following probability distribution:*

$$M_{ij} = \sqrt{3} \begin{cases} +1, & \textit{with probability } 1/6 \\ 0, & \textit{with probability } 2/3 \\ -1, & \textit{with probability } 1/6 \end{cases}$$

Let us define $E = \frac{1}{\sqrt{d'}} MX$, and define $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ that maps the i^{th} column of X to the i^{th} column of E. With probability at least $(1 - n^{-\beta})$, for all $u, v \in X$, we then have

$$(1-\epsilon)||u-v||^2 \leq ||f(u) - f(v)||^2 \leq (1+\epsilon)||u-v||^2 \tag{1}$$

Remark. In [12], the authors recommended the use of probabilities $\{\frac{1}{2\sqrt{d}}, 1 - \frac{1}{\sqrt{d}}, \frac{1}{2\sqrt{d}}\}$ for a significant $\sqrt{d}$-fold speedup, with slight loss of accuracy.

2.2 The kd-Tree Data Structure

Kd-tree [14] is a binary tree structure for storing a finite set of points in k-dimensional space. Every internal node of kd-tree has a splitting hyper-plane that divides the space into two subspaces. The points left to the hyper-plane are represented by the left sub-tree of that node, while the points right to the hyper-plane are represented by the right sub-tree. Thus, each node contains information about all its descendants in a hyper-rectangle. The details of building a kd-tree structure can be found in [14].

The NN search process for kd-tree is conducted in a recursive manner. It starts from the root node, and moves down the tree recursively. The idea is trying to prune the candidate hyper-rectangles that definitely do not contain nearest neighbors of the query point. A candidate hyper-rectangle is inspected only if there are some parts of it within the current best distance to the query point. The number of points inspected appears to be reasonable in low-dimensional space, but usually grows rapidly when the dimensionality of the data points increases. This is the reason that prohibits the use of traditional kd-tree for indexing high-dimensional data.

2.3 Algorithm

We now proceed to introduce our algorithm, Randomly Projected (RP) kd-trees (RP-kd-Trees) for solving the approximate k-NN problem, which takes advantage of random projection for efficient dimension reduction and kd-tree data structure for efficient low-dimensional data indexing. First, we generate m projection matrices using Achlioptas's technique [1]. These projection matrices are used to generate m different copies of projected data set in lower dimensional space d'. These projected data sets are then stored in m corresponding d'-dimensional

kd-trees. By performing projections, we make it possible for kd-tree to handle our data points.

For each projection process, we choose matrix $M \in \mathbb{R}^{d' \times d}$ where M_{ij} is:

$$M_{ij} = \sqrt{3} \begin{cases} +1, & \text{with probability } 1/6 \\ 0, & \text{with probability } 2/3 \\ -1, & \text{with probability } 1/6 \end{cases}$$

The matrix M represents some random projection from $\mathbb{R}^d$ to $\mathbb{R}^{d'}$. Then, multiplying X consisting of n vectors in d-dimensions with matrix M leads to the set of projected points $X' \in \mathbb{R}^{d' \times n}$. Note that the scale factor $\frac{1}{\sqrt{d'}}$ in projection from Theorem 1 could be ignored because we only need to compare among pairwise distances.

Algorithm 1 Preprocessing and Indexing of RP-kd-Trees

```
Input: X - A set of data points, m - number of kd-trees used
Output: RP-kd-Trees T_u, u = 1, ..., m
procedure PREPROCESSING_INDEXING
    for u ← 1, m do
        Initialize kd-tree T_u
        Generate projection matrix M_u
    end for
    for u ← 1, m do
        for i ← 1, n do
            Compute the projected point of x_i with matrix M_u
            Store it into kd-tree T_u
        end for
    end for
end procedure
```

To find k nearest neighbors of a given query point $\mathrm{q} \in \mathbb{R}^d$, we iterate each of m structures and process as follows: First the query point is projected into d' dimensional subspace corresponding to the kd-tree. We use this projected query point and standard nearest neighbors search in kd-tree to find k nearest neighbors. The answer provided by each kd-tree is only an approximate result to the NN problem, and alone may not be very accurate. To improve the accuracy, we try to integrate answers from all the kd-trees by ranking the union set by the distance to the query point $q \in \mathbb{R}^d$, and return the top k nearest neighbors.

The preprocessing, indexing and querying algorithms are summarized in Algorithm 1 and 2. To attain the final result, one priority queue is maintained. It keeps k current best candidate neighbors and will be updated whenever a nearer candidate is found. The insert and update operations in the priority queue are very fast, with complexity of $\mathrm{O}(\log k)$. The querying operation over kd-tree in low dimensional space is very efficient, with complexity of logarithm of the number of points. Besides, our method is also easy to be parallelized by querying multiple kd-trees simultaneously using emerging parallel computing techniques.

By performing random projection, we will lose some information of the data set. However, with m projection matrices and kd-trees we expect the accuracy will be highly boosted. From Theorem 1, the Achlioptas's random projection

Algorithm 2 Approximate Nearest Neighbor Query in RP-kd-Trees

Input: q - a query point, k - number of nearest neighbors
Access: RP-kd-Trees T_u, $u = 1, ..., m$
Output: k (or less) approximate nearest neighbors
procedure ANN-QUERY
  $S \leftarrow \varnothing$
  for $u \leftarrow 1, m$ **do**
    Let $q' \leftarrow M_u q$ the projection of the point q onto the d'-dimensional subspace given by M_u
    Let $S \leftarrow S \cup$ {k neighbors returned from T_u with query q'}
  end for
  Rank points in S by the distance to the query point q
  Return the top k nearest neighbors.
end procedure

method preserves pairwise distances approximately at $(1 \pm \epsilon)$ with probability at least $\gamma = 1 - n^{-\beta}$. That means the failure probability of each structure is $(1 - \gamma) = n^{-\beta}$. Then by integrating results from m structures, we lower this probability to $n^{-\beta m}$. Hence, to achieve the desired probability $(1 - \delta)$, the following inequality must hold: $1 - n^{-\beta m} \geq 1 - \delta$. In other words, one can choose β by: $\beta \geq \frac{-\log \delta}{m \log n}$. Therefore, choosing the value of d' as: $d' \geq \frac{4+2\frac{-\log \delta}{m \log n}}{\epsilon^2/2 - \epsilon^3/3} \log n$, should suffice to provide quality guarantee.

2.4 Complexity Analysis

Empirically RP-kd-Trees could provide its best performance with relatively small value of projected dimension d' (d'=10 when original dimension $d = 297$ in our experiments). The projection time complexity $O(dd')$ is not significant because both d and d' are not very large. Similarly ranking objects in the result sets is very quick because there are usually a small number of candidates. Thus time consumed mostly falls into the process of querying kd-trees, and is expected as $O(d' \log n)$ for one kd-tree. Besides, it needs space complexity $O(nd')$ to store one kd-tree.

The RP-kd-Trees method makes use of m trees, so it has the expected time complexity of $O(md' \log n)$ and space complexity of $O(mnd')$. For the enhanced RP-kd-Trees by distance metric learning to be discussed in the subsequent section, the distance metric learning process (when used, see below for details) usually can be performed quite efficiently because the number of items in training data set is often not large. The only additional computation for the enhanced RP-kd-tress with distance metric learning approach would be projecting the original data set only once by the linear transformation $W \in \mathbb{R}^{d \times d}$ learnt from training data set, which is $O(nd^2)$.

3 Enhancing RP-kd-Trees by Distance Metric Learning

In this section, we consider a machine learning approach to enhancing the indexing performance of RP-kd-Trees for CBIR. In particular, given a training

set with side information (pairwise constraints indicate if image pairs are similar/dissimilar), a well-known technique to improve the distance measure is to explore Distance Metric Learning (DML) techniques, which can improve the performance of RP-kd-Trees by finding more effective distance metrics.

Considering a DML task, we are given a set of n data points in a d-dimensional vector space $\mathcal{C} = \{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^d$, and some side information which is typically provided in the forms of two sets of pairwise constraints among the data points. Each pairwise constraint $(\mathbf{x}_i, \mathbf{x}_j)$ indicates if two images $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar ("must-link") or dissimilar ("cannot-link") judged by users. For image retrieval, such information can be easily collected from real-world systems, such as users' relevance feedback logs in CBIR systems.

One key issue of CBIR is to define appropriate distance measure $f(\mathbf{x}_i, \mathbf{x}_j)$ to calculate distance/dissimilarity between any two images $\mathbf{x}_i$ and $\mathbf{x}_j$. Specifically, assume images are represented in a vector space, by specifying a distance metric $A \in \mathbb{R}^{d \times d}$, we can express the formula of general Mahalanobis distance below:

$$f_A(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_A^2 = (\mathbf{x}_i - \mathbf{x}_j)^\top A(\mathbf{x}_i - \mathbf{x}_j) = \mathbf{tr}(A(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^\top) \quad (2)$$

where A is a symmetric matrix of size $m \times m$, and $\mathbf{tr}$ stands for the *trace* operator. In general, A is a valid metric if and only if it satisfies the non-negativity and triangle inequality properties. In other words, matrix A must be positive semi-definite (PSD), i.e., $A \succeq 0$. In general, A parameterizes a family of Mahalanobis distances on the vector space $\mathbb{R}^d$. As a special case, setting A to an identity matrix $\mathbf{I}_{d \times d}$, Eqn. (2) reduces to regular Euclidean distance.

Despite its simplicity, Euclidean distance has some critical limitations. By Euclidean, all variables are assumed independent, the variance across all dimensions is 1, and the covariances among all variables are 0. Such a scenario is seldom satisfied in practice. Instead of using Euclidean, it is more desirable to learn an optimal metric from real data. This motivates us to study DML to optimize the matrix/metric A for distance measure in real applications.

In this paper, our goal is to apply DML techniques to improve the performance of RP-kd-Trees. The structure of the proposed RP-kd-Trees enables the feasibility of exploiting DML techniques in a simple and effective way. In particular, different distance metrics can be learned separately from different projected data sets in d' dimensional space. The learned metrics can be applied to the RP-kd-Trees by a simple projection. Specifically, each Mahalanobis matrix A can be decomposed as $A = W^\top W$. As a result, the distance $d(\mathbf{x}_1, \mathbf{x}_2)$ is computed:

$$\begin{aligned} f(\mathbf{x}_1, \mathbf{x}_2) &= (\mathbf{x}_1 - \mathbf{x}_2)^\top A(\mathbf{x}_1 - \mathbf{x}_2) = (\mathbf{x}_1 - \mathbf{x}_2)^\top W^\top W(\mathbf{x}_1 - \mathbf{x}_2) \\ &= (W(\mathbf{x}_1 - \mathbf{x}_2))^\top (W(\mathbf{x}_1 - \mathbf{x}_2)) \end{aligned} \quad (3)$$

Thus, applying a metric A to RP-kd-Trees is equivalent to conducting a projection with the matrix W. It is important to note that the above approach does not increase the time cost of online query or any additional memory cost.

In this paper, we apply several popular DML algorithms, including relevant component analysis [3], discriminative component analysis [9], neighbourhood components analysis [11], and large margin nearest neighbor metric learning [16]. For limited space, we skip the discussions on their details.

4 EXPERIMENTS

This section evaluates the performance of the proposed RP-kd-Trees to identify the advantages and limitations of the proposed method from different aspects.

4.1 Data Sets and Experimental Settings

We experimented with real-world image data sets: (1) The COREL data set consists of 5,000 images, which are classified into 50 categories based on their semantic concepts; each category has 100 images; and (2) the Flickr data set contains 500,000 photos, which were crawled from www.Flickr.com.

In the experiments, we split 5,000 COREL images into 2 sets: 2,000 and 3,000 images (i.e. 20 and 30 categories). The 2000-image set was used to test the performance of the methods without DML (RP-kd-Trees, LSH [6], Multi-probe LSH [13]), while the 3000-image set was only used as the training set for the enhanced RP-kd-Trees with DML. Finally, the query set was created by randomly choosing 100 query images from the 2,000 classified images. The image data set to be queried was the combined set of 500k Flickr images and 2000 COREL images, total of 502,000 images, referred as FlickrCOREL.

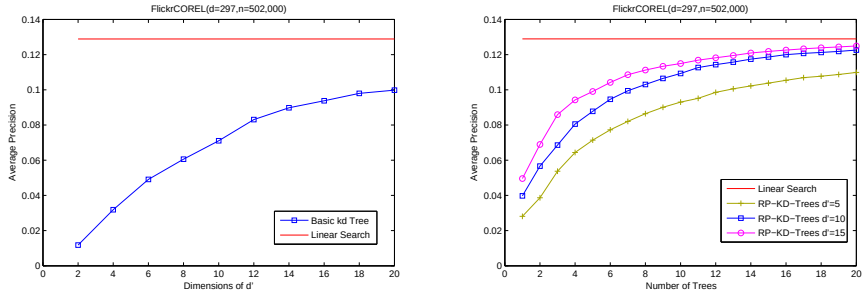
Low-level features were extracted from images, including grid color moment, local binary pattern, Gabor wavelets texture, and edge direction histogram features. All together, a 297-dimensional feature vector was used to represent an image. Practically, storing the FlickrCOREL data set in this way took about 1,137 MB (using double-precision floating point data type or 8 bytes for one coordinate). One trick was exploited to speed up RP-kd-Trees algorthm with little loss on the accuracy that we terminated search process after performing distance checking for a certain number of points. Finally, all experiments were conducted in a Linux machine with 2.8GHz CPU and 16GB memory.

For performance assessment, we adopted a fairly standard metric widely used in multimedia retrieval, i.e., Average Precision metric on top returned images, defined as: $AveragePrecision(t) = \sum_{i=1}^{n} precision(i)\Delta recall(i)$, where $precision(i)$ is the *precision* of the first i returned images and $\Delta recall(i)$ is the change in the *recall* from $i-1$ to i returned images. A returned image was considered a hit if it belonged to the same category as the query image. All methods were required to retrieve 100 relevant images in later experiments.

4.2 Performance Evaluation of RP-kd-Trees

RP-kd-Trees is expected to provide high accuracy search result by using multiple kd-trees. This experiment is to understand the behaviors of RP-kd-Trees with different parameters. Through this experiment, we also want to find out what value the projected dimension should be for RP-kd-Trees. We varied the number of trees m from 1 to 20 and reported the Average Precision, querying time (s) and memory consumed (MB) when projected dimension $d' = 5$, 10, and 15. The experiments were done against FlickrCOREL data set ($d = 297$, $N = 502{,}000$).

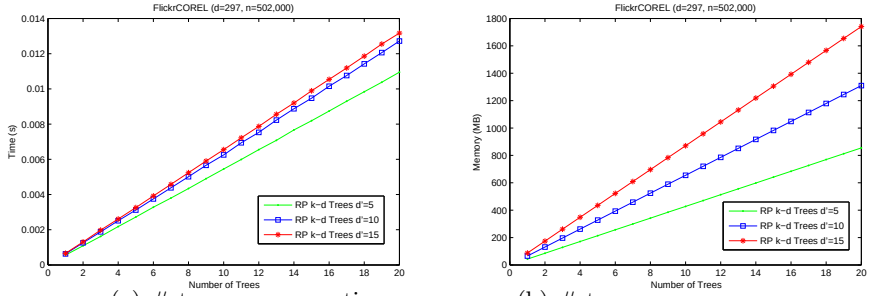
Figure 1 shows a comparison between basic kd-tree and RP-kd-trees. Specifically, Figure 1(b) illustrates how accurate the returned images are for different d'. As reflected from the figure, Average Precision increased significantly when the number of trees m increased. When $m = 20$, the returned average precision of RP-kd-Trees with $d' = 10$ was 0.123, close to the optimal result by the exact



(a) Basic kd-tree (b) # RP-kd-Trees

Fig. 1. Evaluation of Average Precision: Basic kd-tree vs. RP-kd-Trees

linear search. In contrast, the accuracy in Figure 1(a) by basic kd-tree was much lower than the exact linear search. This result indicates that by employing multiple kd-trees the accuracy is highly boosted even though the trick introduced in the preceding section decreases the accuracy slightly.



(a) # trees vs. query time (b) # trees vs. memory

Fig. 2. Evaluation of RP-kd-trees: number of trees vs. query time/meomry

Figure 2 shows the time and memory evaluation of RP-kd-trees. We found that the querying time and memory costs increased linearly with m. When $m = 10$ and $d' = 10$, RP-kd-Trees needed 0.006 seconds on average to answer one query, which was 67 times faster than linear search. At this configuration, it also needed 655 MB and achieved Average Precision of 0.11. To provide higher accuracy, e.g. 0.123, the structures required about 1300 MB and answered a query in about 0.013 second. Thus, if available memory is large enough, this method will be able to deliver the desired accuracy for the application.

Besides, we notice that Average Precision did not improve much when the projected dimension d' increased from 10 to 15. Meanwhile, the querying time and required memory increased considerably. Thus, for this data set, RP-kd-Trees performed well when the projected dimension d' was 10. In later experiments, d' was set to 10 when RP-kd-Trees was compared with other methods.

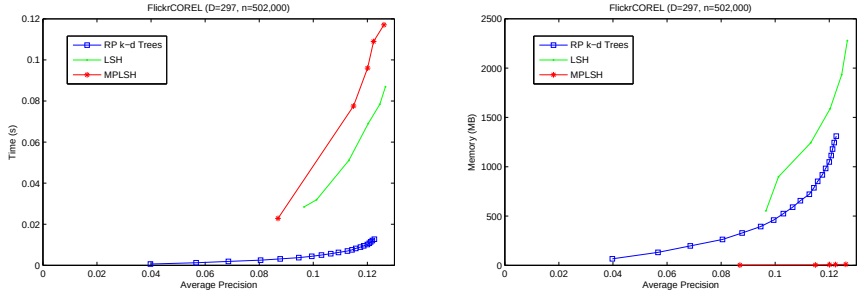
4.3 Comparison against Other Methods

We compared RP-kd-Trees with two state-of-the-art methods, i.e., LSH and Multi-Probe LSH. All compared methods were required to return top $k = 100$ relevant images from dataset FlickrCOREL with respect to each query image.

Parameters were selected to reflect the best performance of each method on the training set. The compared methods are listed below:

- RP-kd-Trees: we set the projected dimension $d' = 10$ and varied the number of kd-trees (m) from 1 to 20.
- LSH [6]: We adopted the E^{2}LSH package[1]. It has two key parameters: L - number of hash tables and k - number of elements in LSH functions. In this experiment, k was set as its typical value 10 while L ranges from 2 to 12.
- Multi-probe LSH [13]: We adopted the LSHKIT library [7]. In the library, the following parameters must be specified: L - number of hash tables, T - number of bins probed in each hash table. When using $L = 10$ and $T = 10$, the search accuracy is very good - nearly close to the exact search result, but the query time is intensive, more than 0.4 second for a query. Thus, in this experiment we used lower values, i.e., $T = 2$ and varied L from 1 to 5.

We ran each of the compared methods on the data set 10 times, and reported their average performance. Figure 3 shows the comparison of different methods in terms of accuracy, querying time, and memory cost.



(a) AP vs. query time (b) # AP vs. memory

Fig. 3. Comparison of different approximate NN search methods

From Figure 3(a), in terms of processing time, multi-probe LSH ran almost as fast as LSH on FlickrCOREL, however, it needed a much smaller number of hash tables, i.e., only 10 MB to store the indexing structure. Meanwhile, LSH needed a lot of memory in order to provide good result, i.e. to have average precision of 0.120, it needed 1588 MB (40% more than the data set itself).

From Figure 3(b), we clearly found that RP-kd-Trees method consistently outperformed other methods on this FlickrCOREL data set. At the average precision of 0.114, RP-kd-Trees achieved up to 7 times faster than LSH method and 10 times faster than multi-probe LSH. The method was still faster when the average precision was higher, says 0.123. Thus, RP-kd-Trees method requires less space while returning nearest neighbors much faster than LSH.

In summary, RP-kd-Trees is an efficient approximate NN search method for high dimensional data, which can return highly accurate results very efficiently

[1] http://www.mit.edu/∼andoni/LSH/. The package solves the R-near neighbor problem (to find the neighbors within a radius R of the query), to find k nearest neighbors, we follow the suggestion of E^{2}LSH's manual, i.e., we solve R-near neighbor problem for several increasing values of R.

when memory is sufficiently large. The results show that RP-kd-Trees is promising and more effective than the competing LSH techniques for this challenge.

4.4 Evaluation of Enhanced RP-kd-Trees with DML

This experiment is to evaluate the performance of the enhanced RP-kd-Trees by applying DML techniques. In particular, we formed a collection of 3,000 COREL images as the training data set for learning distance metrics. In our experiments, we implemented the enhanced RP-kd-Trees by applying four different DML algorithms, including RCA, DCA, NCA, and LMNN. Figure 4 shows the performance of the enhanced RP-kd-Trees by the four DML algorithms.

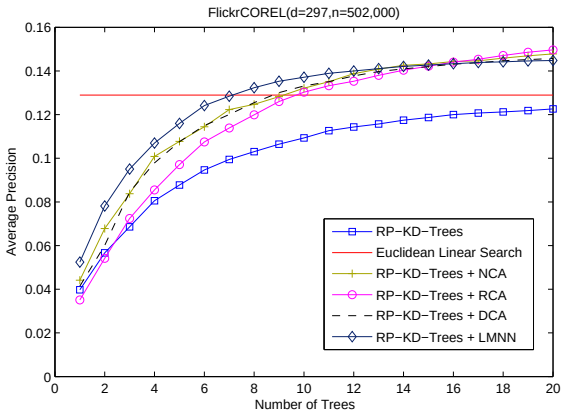


Fig. 4. Evaluation of the enhanced RP-kd-Trees by distance metric learning methods

From Figure 4, we can draw several observations. First, we found that all the enhanced RP-kd-Trees methods by DML achieved consistently better retrieval accuracy than the original RP-kd-trees without DML. Second, we found that the performance of the enhanced RP-kd-Trees monotonically improved when the number of kd-trees increased. In particular, we found that when the number of kd-trees m was greater than 10, all the enhanced RP-kd-Trees algorithms achieved a significant improvement, outperforming the exhaustive Euclidean linear search. All these results show that it is effective and promising for applying DML techniques for boosting the performance of the RP-kd-Trees technique.

Moreover, by examining different DML techniques, we found that when the number of kd-trees m is small ($m \leq 10$), LMNN tends to achieve consistently better than the others, both NCA and DCA perform quite comparably, while RCA seems to be the worst. Further, when m increases, we found that most algorithms tend to converge to the similar performance. Despite their slight differences, we can clearly observe the consistent improvements by the enhanced RP-id-Trees with DML, validating the efficacy of our technique.

5 Conclusions

This paper presented a new approximate NN search method for high dimensional data, called Randomly Projected kd-Trees (RP-KD-Tree), with application to

CBIR. Our results showed that the proposed method requires less memory and can achieve up to 7 times faster than the original LSH. Further, we showed that our method can be easily extended by applying distance metric learning techniques. By employing a variety of distance metric learning algorithms, we showed the extended method can provide consistent improvements of retrieval accuracy, even exceeding the accuracy of the exhaustive Euclidean based linear search, with no additional querying time or memory cost. For the merits of efficacy and being easy to implement, we believe RP-kd-Trees could be a practically effective technique for high-dimensional indexing in multimedia applications. Future work will further reduce the space complexity and speed up by parallel techniques.

Acknowledgement

The work was supported by Singapore National Research Foundation Interactive Digital Media R&D Program under research grant NRF2008IDM-IDM004-006.

References

1. Achlioptas, D.: Database-friendly random projections. In: ACM Symp. on the Principles of Database Systems. pp. 274–281 (2001)
2. Arya, S., Malamatos, T., Mount, D.M.: Space-time tradeoffs for approximate nearest neighbor searching. J. ACM 57(1), 1–54 (2009)
3. Bar-Hillel, A., Hertz, T., Shental, N., Weinshall, D.: Learning a mahalanobis metric from equivalence constraints. JMLR 6, 937–965 (2005)
4. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM 18(9), 509–517 (1975)
5. Clarkson, K.L.: A randomized algorithm for closest-point queries. SIAM J. Comput. 17(4), 830–847 (1988)
6. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proc. 20th annual symposium on Computational geometry (SCG'04). pp. 253–262. New York, NY (2004)
7. Dong, W., Wang, Z., Josephson, W., Charikar, M., Li, K.: Modeling lsh for performance tuning. In: ACM CIKM Conference. USA (October 2008)
8. Gionis, A., Indyk, P., Motwani., R.: Similarity search in high dimensions via hashing. In: VLDB (1999)
9. Hoi, S.C., Liu, W., Lyu, M.R., Ma, W.Y.: Learning distance metrics with contextual constraints for image retrieval. In: CVPR (Jun 17–22 2006)
10. Indyk, P., Motwani, R.: Approximate nearest neighbor: Towards removing the curse of dimensionality. In: STOC. pp. 604–613 (1998)
11. J. Goldberger, S. Roweis, G.H., Salakhutdinov, R.: Neighbourhood components analysis. In: NIPS17 (2005)
12. Li, P., J., H.T., W., C.K.: Very sparse random projections. In: ACM International Conference on Knowledge Discovery and Data Mining (KDD) (2006)
13. Lv, Q., Josephson, W., Wang, Z., Charikar, M.S., Li, K.: Multi-probe lsh: efficient indexing for high-dimensional similarity search. In: VLDB. Vienna, Austria (2007)
14. Robinson, J.T.: The k-d-b-tree: A search structure for large multi-dimensional dynamic indexes. SIGMOD pp. 10–18 (1981)
15. Shamos, M., Hoey, D.: Closest-point problems. In: Proc. 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS). pp. 151–162 (1975)
16. Weinberger, K.Q., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. JMLR 10, 207–244 (2009)