DICTA2002: Digital Image Computing Techniques and Application, 21--22 January 2002, Melbourne, Australia.

An Investigation of Using K-d Tree to Improve Image Retrieval Efficiency

Yunshuang He, Guojun Lu and ShyhWei Teng
Gippsland School of Computing and Information Technology
Monash University, Churchill, Vic 3844
yunshuang.he@infotech.monash.edu.au

Abstract

Content-based image retrieval problem is in essence to determine distance or similarity between multi-dimensional image feature vectors. Linear comparison will be too slow when vector dimensions and image database are large. We investigate the use of K-d tree in image retrieval based on color histograms and found that K-d trees improve retrieval efficiency significantly. We evaluate the impact of leaf node size and the number of images required to retrieval on the retrieval performance. Finally, we discuss the relationship between vector dimensions and retrieval efficiency.

1. Introduction

In multimedia database system, content-based retrieval of similar objects such as images, audios and videos, have been required in many applications [1], [2], [3], [4]. However, in contrast to classical search in a relational database, similarity search is often required for content-based retrieval. For example, Image database can be queried to find and retrieve images in the database that are similar to the query image.

Similarity is typically measured not on objects directly, but rather on feature vectors of objects. In many cases, feature vector, which are extracted from the important properties of objects, are often high-dimensional vectors. Examples of features are color histograms [4], shape descriptors [2], [6] and Fourier vectors [5] etc. Then similarity of two objects is determined as the distance between two feature vectors and similarity search corresponds to a nearest-neighbor search within the vector space of the features.

The most straightforward way to solve the nearest neighbor search problem in high dimensional space is to sequentially/linearly compare all feature vectors, using the given distance measure. It will calculate all the distance between query and points in data set, which is very expensive for large database.

The other approach to this problem is to use a multidimensional index structure. One technique is space-partitioning method, which recursively partitions space into mutually disjoint subspaces, like k-d tree [7], quadtree [8], and LSD tree [9]. Another technique is Data-partitioning index tree which divide the data space according to the distribution of data, such as R-tree [10], R+ tree [11], R* tree [12], X-tree [13] and SR Tree [14]. The main idea behind this approach is to efficiently prune searching space during searching process so that the access cost becomes logarithmic in the number of objects in the tree.

Data partition structures like R-tree [10], R*-tree [12], and X-tree [13], consist of bounding boxes which are hierarchically arranged in a d-dimensional Space. This structure uses all the d dimensions to represent space partitioning which leads to nodes with low fanout at high dimensionalities. Furthermore, the overlap in the directory increases very rapidly with growing dimensionality of the data. Overlap in the directory directly affects the query performance; more overlap means more paths have to follow when processing a query.

Space partition structures like k-d tree [7], quadtree [8], and LSD-tree [9], are more suitable for high dimensional feature spaces. Firstly, the fanout of node is independent of dimensionality since the size of these structures will not increase with dimensionality. Secondly, the space is always partitioned into non-overlap subspaces, unlike data partition structure, which may follow multiple paths even for simple point queries.

In this paper, we study colour based image retrieval efficiency using K-d trees. We evaluate the impact of leaf node size and the number of images required to retrieval on the retrieval performance. Finally, we discuss the relationship between vector dimensions and retrieval efficiency.

2. VAM K-D Tree

The k-d tree [7] is a binary tree in which each node represents a hyper-rectangle and a hyper-plane orthogonal to one of the coordinate axis, which splits the hyper-rectangle into two parts. These two parts are then associated with the two child nodes.

Sproull [17] provided taxonomy of k-d tree variants. He listed what we call the three dimensions of k-d tree variations: (1) split (partition plane) orientation, (2) split (partition plane) position, and (3) distance representation.

VAM K-D tree [18] choose the split orientation as the dimension with the largest variance. They found that compared with the standard maximum spread dimension, used in [19], this provided equivalent or slightly better search performance and required less CPU time to build the tree. Furthermore they chose the partition value of a node to be the approximate median of the data points in the hyper-rectangle represented by that node along the split dimension, making the k-d tree node split equally to their child nodes.

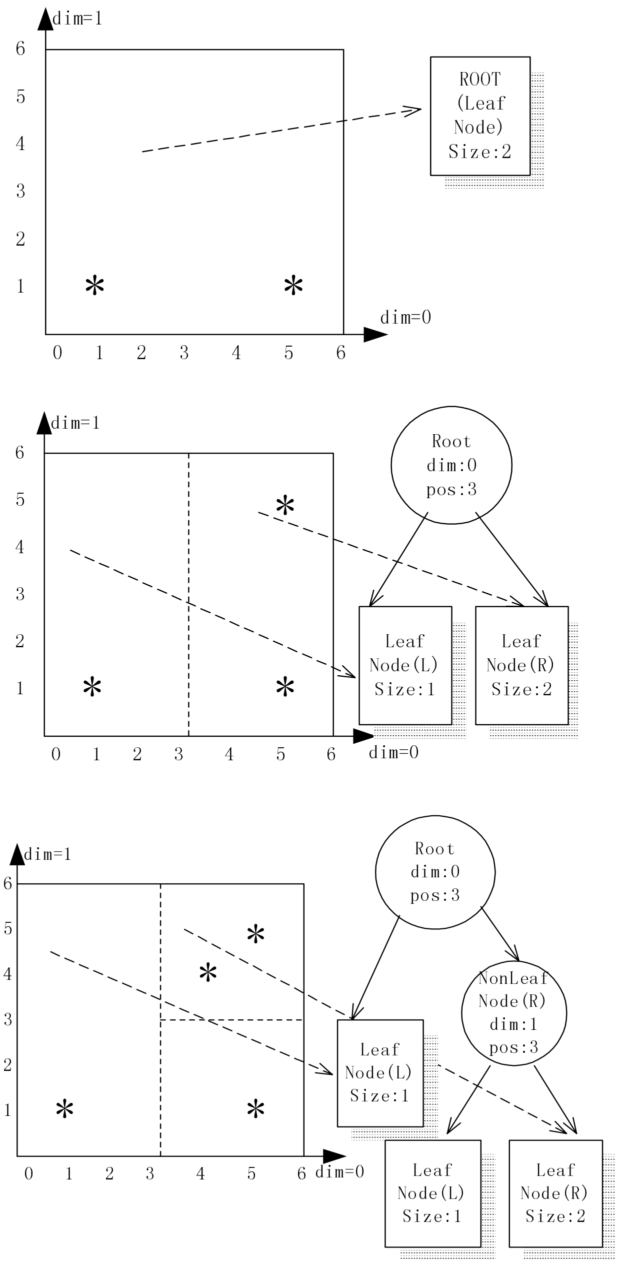

Figure 1. Creation of VAM KD Tree

Figure 1. shows the creation of VAM KD Tree, with dimension of 2 and bucket size of 2. Initially, the whole data space corresponds to one node as the root of tree. After insertion of another data, the root node is overflowed and a split occurs. The split dimension is 0 and position is 3, which are determined by largest variance and approximate median value. Then the data are moved to child nodes of root corresponds to their position in split dimension. With the next insertion, the right node of root is overflowed. Again this node is split into two child nodes. This process is repeated each time the capacity of a leaf node is exceeded.

3. Nearest Neighbor Search Algorithm for K-D Tree

At each leaf node visited the distance between the query point and each data point in the bucket is computed, and the nearest neighbor is updated if this is the closest point seen so far. At each internal node the subtree whose corresponding hyper-rectangle is closer to the query point is visited. Later, the farther subtree is searched if the distance between the query point and the closest point visited so far exceeds the distance between the query point and the hyper-rectangle of the farther subtree.

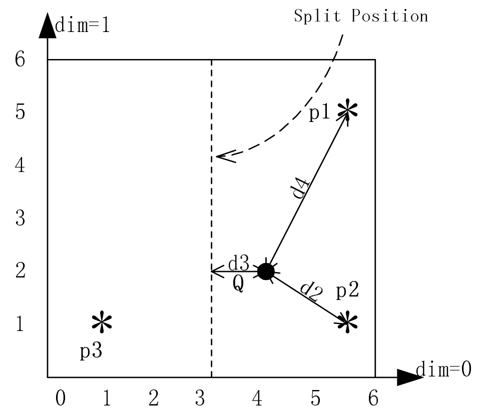

Figure 2. NN search in K-d tree

For example, in Fig 2, we first calculate the distance between Q and p1 as d4, and the distance between Q and p2 as d2. Then we can get the nearest distance in this leaf node as d2. Secondly we compare the nearest distance we have got so far with the distance between Q and split boundary as d3 to see if we need to visit to the sibling node. Obviously d3 is less than d2, so we have to visit left node to check whether the nearest neighbor is in left node.

4. Application of VAM -D Tree to Image Retrieval

One of the most common content-based image retrieval techniques is based on colour histograms. A histogram can be considered as an n-dimensional vector, n being the number of bins of the histogram. To test retrieval efficiency using K-D tree, three histograms with bin number 64, 512 and 4096 are created for each image. A total of 10,115 images are used in the text database. For each group of histograms (corresponding to three different bin number), a K-d tree is created for the test image database.

4.1. Optimal bucket size

Bucket size (the maximum number of vectors in each leaf nodes) is an important parameter we have to decide before building a VAM KD Tree. The K-D Tree with the best bucket size is the k-d tree, which finds the best K-NN (K – nearest neighbor) matches in the least time.

The time cost in K-NN search mainly includes distance calculation time and node visiting time. When bucket size grows large, the distance calculation time increases but the nodes visiting time decrease.

We study the CPU Time cost when bucket size grows up for 64, 512, 4096 dimension histogram data respectively.

Our experimental results show that when bucket size is too small, the time cost by nodes visiting is dominant. On the other hand when bucket size is too large, the time cost on distance calculation is very expensive. The results shows, there is a compromise between nodes visiting time and distance calculation time, the bucket size from 20 to 40 can achieve the least CPU time cost for all the three dimensions.

4.2. Retrieval efficiency

Table 1 shows the percentage of nodes visited by using the VAM k-d tree (bucket size 20) to find the 1-NN match for the three dimensions (assume the traditional linear search will visit all (100%) nodes). The results show that the average leaf nodes visited for 64 dimension data of 100 queries is about 17%. And even dimension grows as high as 4096, there are still about 40% of nodes not visited by 1-NN search algorithm.

Table 1. Average Percentage of Nodes visited by 1- NN Algorithms

Dimension	64	512	4096
VAM k-d tree	12%	45.22%	59.44%

4.3. K-NN Search efficiency

In many cases of image retrieval, we wish to find a ranked list of K nearest neighbor. A priority queue is created to keep the K nearest neighbor found so far in the searching process. We use the bucket size of 20 to perform the K-NN test. Table 2 summarizes the K-NN search performance.

Table 2. Average Percentage of Nodes visited by K- NN Algorithms

Number of NN	1	10	20	40	80
64	12%	30.14%	34.48%	39.61%	45.96%
512	45.2%	73.23%	75.47%	76.92%	78.46%
4096	59.4%	79.15%	79.61%	80.13%	80.58%

The results show that the number of visited nodes increases quickly when K varies from 1 to 10. The number of visited nodes increases steadily when K varies from 10 to 160.

4.4. Discussion

Other researchers found that when the vector dimension is higher than 30 for uniformly distributed data set, the use of data structures will not be able to improve search efficiency over the linear search [13][14][15]. To test this finding, we randomly generated 10000 data (ie, they are uniformly distributed in the multi-dimensional space). By applying the K-d tree, we found that the percentage of node visited grows exponentially as dimension increases for NN search (Figure 3). Before the dimension reaches 30, all the nodes of the VAM K-D tree have been visited, which means that there are no improvement of VAM K-D tree on high dimension (>30) uniform distributed data sets. Then why did we observe significant performance by using the K-d tree even when the dimension is equal to 512

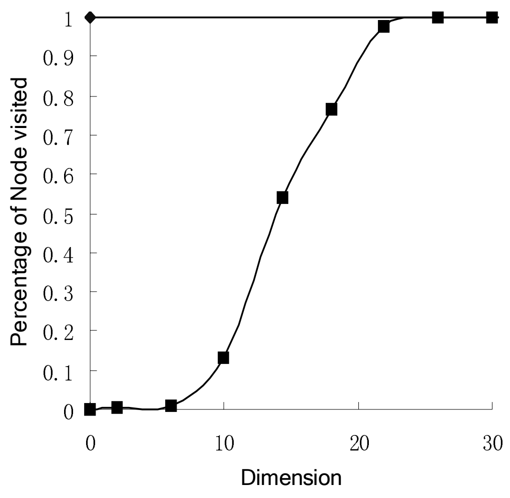

Figure 3. Percentage of Nodes visited grows exponentially as dimension increase

The answer lies in the type of data distribution and effective (intrinsic) dimension. Image feature vectors (eg colour histograms) are often correlated and clustered and often have dependencies between attributes. Therefore there effective dimension is low. For example if we divide the 64, 512, 4096 dimension space by splitting in middle of every side of space. We can get 264, 2512, 24096 hypercube respectively. But when we put the 64, 512, 4096 dimension histogram data of 10115 photos into these hypercube, they only occupy 40, 74, 31 of them,

The intrinsic dimensionality of a real data set is significantly lower than its embedding dimension. Pagel et al [16] show both analytically and experimentally that the fractal dimension, rather than the dimension of space containing the data set, is the real determinant of performance.

The Hausdorff fractal dimension is defined as below:

For a point-set that has the self-similarity property in the range of scales (r1, r2), its Hausdorff fractal dimension D0 for this range is measured as:

$$D_0 = -\frac{\partial \log(N(r))}{\partial \log(r)} = c \quad \text{for} \quad r \in (r_1, r_2)$$

We plot log (N (r)) vs. log (r). If the point –set is self-similar for $r \in (r_1, r_2)$ then its plot will be a straight line for this range. The slope of this line is the Hausdorff fractal dimension D0 of the point-set for the range of scales (r1, r2).

We calculated fractal dimensions of the 10,115 vectors for the three dimensions of 64, 512 and 4096. We found that their corresponding fractal dimensions are 5.35, 6.23, and 4.97 respectively. (One possible reason that the fractal dimension of the data set of 4096-dimension is lower than the other two is that the calculation of the fractal dimension is an approximation.)

From the above, we can conclude that the intrinsic dimension for color histogram is much lower than the bin numbers. This explains why the K-d tree can improve the K-NN search efficiency.

5. Conclusions

Solution of the find the similar image problem requires solving the nearest neighbor problem in vector space. VAM K-D Tree is an efficient method for finding nearest neighbors in high dimensional search problem. In this paper, we have adapted the VAM K-D trees to the problem of image retrieval and found the suitable bucket size regarding minimizing the access time. Tests on 64, 512, 4096 dimension histogram data of 10115 image data sets showed that though the dimension is very high, we still achieve some improvement compared with linear search. For dimension is as high as 64, which shows no improvement for uniform distribution data, the VAM K-D tree still gives an improvement for more than 3 times for color histogram data compared with linear search because of the correlation in the real data sets.

References

[1] C. Faloutsos et al., “Efficient and Effective Querying by Image Content,” J. Intelligent Information Systems, vol. 3, pp. 231-262, 1994.

[2] H.V. Jagadish, “A Retrieval Technique for Similar Shapes,” Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 208-217, 1991.

[3] R. Mehrotra and J.E. Gary, “Feature-Based Retrieval of Similar Shapes,” Proc. Ninth Int'l Conf. Data Eng., pp. 108-115, 1993.

[4] H. Sawhney and J. Hafner, “Efficient Color Histogram Indexing,” Proc. Int'l Conf. Image Processing, pp. 66-70, 1994.

[5] Wallace T., Wintz P. “An Efficient Three-Dimensional Aircraft Recognition Algorithm Using Normalized Fourier Descriptors”, Computer Graphics and Image Processing, Vol. 13, pp. 99-126,

1980

[6] Mehrotra R., Gary J. "Feature-Index-Based Similar Shape retrieval", Proc. of the 3rd Working Conf. On Visual Database Systems, March 1995

[7] BENTLEY,J.L.AND FRIEDMAN, J. H.. Data structures for range searching. ACM Comput. Surv. 11, 4, 397–409, 1979

[8] SAMET, H. "The quadtree and related hierarchical data structure". ACM Comput. Surv.16, 2, 187–260, 1984

[9] HENRICH, A., SIX, H., AND WIDMAYER, P. "The LSD tree: Spatial access to multidimensional point and non-point objects". In Proceedings of the Fifteenth International Conference on Very Large Data Bases, 45–53, 1989.

[10] A. Guttman. "R-trees: A dynamic index structure for spatial searching". In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 47-57, Boston, MA, June 1984.

[11] T. Sellis, N. Roussopoulos, and C. Faloustos. "The R+-tree: A dynamic index for multi-dimensional objects". In Proc. Of the Int. Conference on Very Large Databases, pages 507-518, Brighton, England, 1987.

[12] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. "The R*-tree: An efficient and robust access method for points and rectangles". In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, pages 322-331, Atlantic City, NJ, 23-25 May 1990.

[13] S. Berchtold, D. Keim, and H.-P. Kriegel. "The X-tree: An index structure for high-dimensional data". In Proc. Of the Int. Conference on Very Large Databases, pages 28-39 1996.

[14] N. Katayama and S. Satoh. "The SR-tree: An index structure for high-dimensional nearest neighbor queries". In Proc. of the ACM SIGMOD Int. Conj. on Management of Data, pages 369-380, Tucson, Arizona USA, 1997.

[15] S. Berchtold, C. BBhm, B. Braunmiiller, D. Keim, and H.-P. Kriegel. "Fast parallel similarity search in multimedia databases". In Proc. of the ACM SIGMOD Int. Conf. On Management of Data, pages 1-12, Tucson, USA, 1997.

[16] A. Belussi and C. Faloutsos. "Estimating the selectivity of spatial queries using the 'correlation' fractal dimension". In Proc. 21st Intl. Conf. on VLDB, pages 299-310, Zurich, 1995

[17] R. Sproull. "Refinements to nearest- neighbor searching in k-dimensional trees". Algorithmica, 6(4): 579-589, 1991

[18] D.A White and R. Jain, "Similarity Indexing: Algorithms and Performance", Visual Computing Laboratory, University of California, San Diego, 1997

[19] J.H. Friedman, J.L. Bentley, and R.A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time", ACM Fransactions on mathematical Software, 3(3), p.209-226, Sep. 1977