マルチメディア通信と分散処理 82－18
（1997．4．25）

Similarity Indexing in High Dimensional Image Space

K. Curtis, J. Nakagawa, N. Taniguchi and M. Yamamuro
{kate, junichi, nota, masashi}@dq.isl.ntt.co.jp

Database Systems Laboratory
NTT Information and Communication Systems Laboratories

Any large scale image database must be coupled with fast and accurate image retrieval methods so that users can effectively utilise its resources. Similarity retrieval focuses on the use of high dimensional feature vectors to represent image content. Several indexing methods have been proposed for such vectors including many variants of the R-tree structure. This paper proposes a method of index construction that combines elements of non-hierarchical clustering methods and R-tree construction methods to produce an optimised tree structure index. This new structure, known as the C-tree, has been implemented. Experimental results show that, where construction speed is not of primary importance, the C-tree can provide superior retrieval performance.

画像類似検索のための多次元空間インデクス

キャサリン　カーティス　　中川　純一　　谷口　展郎　　山室　雅司
ＮＴＴ情報通信研究所

大量画像をデータベース化し，画像の特徴によって検索する画像類似検索が注目されている．その検索の基本は多次元の画像特徴ベクトルによる近傍検索であり，データベースが大規模になると検索性能が問題になる．本論文では，多次元ベクトル空間での検索高速化のための新しいインデクスC-treeを提案する．本インデクスを実装し，従来法の中で高速だといわれているVAMSplit R-treeとの比較評価実験を行った．その結果，VAMSplit R-treeに比べてC-treeは，検索ではかなりの性能向上があり，次元数やデータ数に対するスケーラビリティも確認された．これにより，検索速度が要求されるような大規模画像データベースの検索で有効であることが分かった．

1 Introduction

Advances in multimedia technologies have led to a high demand for image database systems. An important component of such systems should be an indexing system that enables fast searching for similar images based on image content. This procedure satisfies the need of the database user to find images that are alike without the need to verbalise the similarities between images. Images can be represented by a set of feature vectors and advances in image processing have meant that these vectors which can be automatically extracted from the image data and stored together with the original image in the database. For example, red, green, blue, hue, saturation, intensity and shape are all possible features. Similarity retrieval is the process of identifying similar images by measuring the similarity of their feature vector sets. Although Euclidean distance is most commonly used as a similarity measure, it can be advantageous to allow for other measures such as Manhattan distance or data specific formulas. Automatic extraction of feature vectors gives an important advantage over more traditional key word retrieval methods which tend to rely on time consuming and error prone manual input of key words into the database.

It is anticipated that each vector will be of high dimension (sample image databases used in our research thus far have had feature vectors that range in dimension from 16 to 256), rendering most common spatial indexing systems inefficient. The most promising advances have been achieved though modifications to the k-d tree and R-tree structures [White 1996a]. This paper looks at another alternative way of building the tree index structure. Although the indexing method described in this paper has only been applied to the indexing of image databases so far, the system is equally applicable to the indexing of the high-dimensional data that occurs in other fields, for example computer aided design, molecular biology and time-series data collections of market data.

2 Background

The validity of measuring the distance between image feature vectors to establish the level of similarity between images is a basic assumption of this research. Vectors created from the colour histograms of images were shown to be effective in [Swain 1990]. More recently hue, saturation and intensity vectors have been included and algorithms proposed for the extraction of shape vectors [Akama 1997]. While most research has detailed only the use of Euclidean distance as the measure of similarity between vectors, [Vinod 1996] proposed a colour intersection method that took advantage of the computational efficiency of measuring the Manhattan distance between vectors.

An R-tree is an extension of the B-tree indexing method for multidimensional objects. A complex data element is represented by its minimum bounding

hyper-rectangle (MBR). The tree is constructed so that non-leaf nodes contain a pointer to a set of child nodes and the value of the MBR of the MBRs of all the child nodes. Leaf nodes contain a pointer to a data element and the MBR of that element. When dealing with the simplified case of data elements from a feature vector set, leaf nodes contain a pointer to the image object and the value of the feature vector for that image. Non-leaf nodes can also be referred to as clusters as they group a set of nodes together.

Many different flavours of R-tree, have been proposed since the original R-tree of [Guttman, 1984]. However, most varieties of R-tree have not been able to efficiently search data of high dimension. As pointed out in [Berchtold 1996] as the dimension of the data increases, the overlap between the node MBRs at a single level in an R-tree increases dramatically leading to a very high number of nodes that must be searched at each level of the tree. At this point a linear search often becomes more effective. Most R-tree structures remain dependent on the order in which the data is inserted into the tree and fail to take advantage of the fact that all the data is available at the start of the construction process. Some of the more successful refinements to the tree construction algorithm have introduced forced reinsertion of nodes and the deferred splitting of nodes [Berchtold 1996, White 1996b]. These methods enable the tree to see a greater percentage of the data elements before making a final decision as to the position of a data element in the tree. [Gavrila 1994] ably points out this disadvantage and suggests a construction algorithm that makes use of an optimisation function over the entire data set. The major disadvantage of this algorithm is the lengthy processing time (on the order of several hours for data sets of 100,000 elements) necessary to construct the tree.

Currently, one of the most successful implementations of the R-tree for indexing high dimensional data appears to be the VAMSplit R-tree proposed by [White 1996a]. This tree is created by recursively choosing splits of the data set using the maximum variance dimension and then choosing a split that is the median. The tree has a fast construction time and does take advantage of examining the entire data set before choosing the first split.

Once the index tree has been built, a search algorithm must be implemented to find the k-nearest neighbours to a given data element. The branch and bound search algorithm for the nearest neighbour searching technique in [Roussopoulos 1995] gives the basis for the search algorithm for many of the R-tree variations. They detail the concept of using minimum and maximum distance metrics between the key data element and the minimum bounding hyper-rectangles of a set of non-leaf nodes to establish a prioritised search order among the those nodes.

Approximate searching has also been proposed as a way of improving retrieval performance in situations where a user may not absolutely require an the most accurate result. This is particularly suitable to image data queries. Since the comparison of feature vectors is only an approximation to the comparison a human eye would make, a further level of approximation does not usually significantly affect the search results and can lead to significantly increased search times.

3 The C-tree

The purpose behind design of the C-tree is to provide a faster retrieval indexing system for very large image repositories in situations where the construction time of the index is of secondary importance to the retrieval performance time. Since each image may have five to ten feature vectors associated with it, it would be foolish to completely ignore either construction time or memory storage requirements. It must still be possible to construct all the indices for a database in a reasonable time (minutes rather than hours) from a database administrator's point of view.

3.1 Construction

Overview

The C-tree is largely based on Wishart's [Anderberg, 1973] method for non-hierarchical clustering and is named for this use of clustering methods. Non-hierarchical clustering methods are designed to cluster data units into a single classification of k clusters, where k is either specified *a priori* or is determined as part of the clustering method. Wishart's method was chosen as it allows the number of clusters to be determined as part of the clustering process and its use of a residue list appears to give better clustering results for use in tree construction than MacQueen's k-mean method. Clusters can themselves be considered data elements and clusters of clusters created in order to build up a hierarchical structure.

The C-tree is constructed from the bottom (the leaves) up, unlike most other previous construction methods which have worked from the root down. The first step of the method uses Wishart's algorithm to divide the data between an appropriate number of clusters. Data units that do not fit well in any cluster (i.e. outlying single data units) are put into a residue list. These clusters make up the deepest layer of non-leaf nodes, with the data elements contained inside the clusters forming the deepest leaf nodes of the tree. The cluster centroids, together with the data units in the residue list, are then treated as a new data set, and the clustering algorithm reapplied to find the nodes of the tree at the next level. This procedure is continued until only one cluster is needed to enclose all nodes at the previous level in the tree. This cluster is the root of the tree. It will immediately be noticed that this procedure creates an unbalanced tree, with leaf nodes (actual data elements) possibly appearing at any level in the tree.

Each node in the tree records various bookkeeping information about itself which is used to compile statistics about the tree and for use in the search algorithm. For non-leaf nodes this information includes at least the minimum bounding hyper-rectangle and

centroid for the child nodes emanating from that node.

Details of Wishart's Method

Wishart's method begins by defining three parameters

thresh: the maximum distance between a data unit and the center of the cluster it belongs to

minsiz: the minimum number of data units in a cluster

maxit: the maximum number of iterations of the procedure

The first step of the method is to arrive at an initial distribution of data units in clusters. This is achieved using the VAMSplit R-tree and is described in the next sub-section.

Each cluster is then processed one by one. For each data unit in the current cluster, the cluster which minimises the distance between the cluster centroid and the data unit is identified. If this minimising cluster is not the current cluster, the data unit is moved to this closest cluster. However, if the distance between the data unit and the closest cluster is greater than *thresh*, then the node is instead assigned to the residue list. After an assignment or deletion of a node to a cluster, the cluster centroid is recalculated. When all clusters have been processed, the clusters with less than *minsiz* data units are elimated and these units assigned to the residue list. Finally all the data units in the residue list are examined and if the minimum distance from a unit to its closest cluster is less than *thresh*, the data unit is moved to that cluster. This cycle is repeated until either no changes are made during an iteration or the number of iterations has reached *maxit*.

An important modification was made to Wishart's methods in order to significantly speed up its execution. At the beginning of each iteration through the cluster list, the complete set of inter-cluster distances are calculated. By doing this, only nearby clusters need be considered in the search for a cluster that is closer to a data element than the cluster it currently belongs to. The list of nearby clusters is only updated once every iteration, which may result in some inaccuracies in the list as data elements are moved between clusters. However, these will be resolved on the next iteration through the cluster list and the possible extra iteration required is more efficient than exhaustively comparing every data element with every possible cluster.

Finding the Initial Cluster Distribution

There are many ways of arriving at an initial cluster configuration for the data units. It is possible to start by choosing a minimum distance, *mindist*, for cluster separation, or by choosing a desired number of clusters and letting the program automatically arrive at a suitable distance. Both these methods consume significant processing time since they involve several iterations through the complete data set.

One parameter over which we would like to be able exert some control, that is not considered by Wishart's method, is the maximum number of data elements assigned to a cluster. If data elements are not artificially prevented from joining an already full cluster, the maximum number of data elements in any cluster is initially determined by *mindist*.

A solution to this dilemma was found by choosing the VAMSplit R-tree as the best way to establish an initial cluster configuration for a set of data elements. The R-tree has strict limits on the maximum number of data elements allowed in any one internal node and so adjusting the tree construction parameters is an easy way of controlling the number and size of the initial clusters. The R-tree is balanced so that a set of clusters containing all the data units can always be easily extracted from the bottom layer of the tree. Most important, the fast construction speed of the VAMSplit R-tree, together with its ability to create clusters with small overlap, aids greatly in the overall performance rating of the C-tree construction method.

As successive levels of the tree are created there are no problems associated with applying the VAMSplit R-tree algortihm to a set of cluster centroids, or a mixed set of cluster centroids and data elements that may result from the merging of a cluster set and a residue list.

Parameter Optimisation

The parameter *thresh* plays perhaps the most important role in determining the final cluster configuration at a given level in the tree. It can be thought of as the maximum radius of a cluster and directly impacts the number of elements that can be stored in a cluster. If *thresh* is too large, the number of elements in a cluster may become too many to allow effective searching. Alternatively if *thresh* is too small, too many data elements will be assigned to the residue and the number of clusters will be small and each surviving cluster will have close to the minimum number of elements allowed.

It is intuitive to see that each new level in the tree must be created with a larger value of *thresh* than the level below. In this way the cluster radius increases from the bottom of the tree towards the top as the clusters at each level include successively greater numbers of data elements. The radius of the root node/cluster must be large enough so that all data elements can be included in the root node cluster.

Although the *thresh* parameter is extremely important to the success of the clustering algorithm, there is no obvious method for determining its value. In order to avoid lengthening the execution time of the algorithm, iterating over several values of *thresh* to find the best distance is not a feasible option. Instead, a suitable value for thresh can be found by examining the initial cluster configuration derived from the VAMSplit R-tree. The value of *thresh* is derived from the values of the maxium distances from a cluster centroid to a data element in that cluster. Currently, *thresh* is chosen equal to seventy per cent of the average maximum distance but further experimentation might prove useful in determining a more effective choice.

It would also be possible to add a step to the clustering algorithm that examines each cluster and, if

the number of data elements were greater than a set limit, to split the cluster into two or more sub-clusters, increasing the depth of the tree at that point. However, initial investigations revealed no performance enhancement when this was tried and the control of the parameter *thresh* appears to be sufficient to achieve good clustering results.

The value of *maxit* should be high enough to allow the clusters to achieve stability. In practice a value of 20 seems to be more than adequate. The most important cluster rearrangements generally occur on the first two or three cycle and so where a faster index construction time is more important, the number of iterations can be reduced.

The value of *minc* is more difficult to determine. The minimum possible value for *minc* is 2. A value of 1 would render the use of a residue list largely unnecessary. By increasing the value for *minc*, the size of the residue list at each level is increased significantly. From practical observation, a value of 5 or 10 leads to acceptable results.

Construction Speed

The C-tree index construction method was designed to achieve good clustering results in order to improve data retrieval speeds. It was assumed that tree construction would be carried out infrequently, as images are usually inserted into a database in a block at the same time. Therefore, a fast construction time is not very important. The tree structure does allow for the fast dynamic insertion of a number of image data units after the initial tree construction. Dynamic insertion could be as simple as searching for the most similar data element already in the tree and inserting the new element in the same cluster. Since there is no set maximum number of elements in any cluster no problems with cluster overflow need be considered. However, after a given number of such insertions, the entire tree should be deleted and reconstructed so that search performance is not substantially degraded.

Since the construction algorithm does require the computation of the similarity between a significant percentage of data element pairs, the time needed will still grow exponentially with the number of data elements, albeit more slowly than previous algorithms that have attempted such optimisations. A limit to the number of data elements that can easily be included in a tree, without further modification to the algorithm, exists and depends on the dimensionality of the data elements and the speed of the processor executing the construction. However, it is possible to significantly alleviate this problem by taking advantage of the initial execution of the VAMSplit R-tree alogorithm to split the tree into a small number of sub-trees and then execute the C-tree construction method on each sub-tree before merging the resulting trees back together to form one complete tree.

3.2 Searching the C-Tree

The algorithm for searching the tree to find similar images is straightforward. The algorithm aims to find the k nearest data elements to a supplied key element, with an allowable approximation of a. The approximation factor a represents the percentage error that is allowed when removing clusters of data elements from consideration during the search process.

The basic algorithm is to begin at the top level of the tree, by ranking all the nodes at that level according to the distance from MBR of that node to the key data element. Then, the highest ranked node (i.e. the node closest to the key element), that is not a leaf node (i.e. is not an actual data element) is expanded. The node is removed from the ranking list and instead, all the child nodes of this node are added to the ranking list according to their distance from the reference image. The highest ranked non-leaf node in the ranking list is repeatedly examined in this way. The process terminates when the ranking list contains only leaf nodes.

In order to expedite this search process, nodes can be removed from the ranking list when it becomes impossible for that node to contain data elements that are closer to the reference image than the top k data elements found thus far. Therefore if the distance between the theoretically closest possible data element contained in the node and the reference image is greater than the distance between the current k^{th} closest data unit and the reference image, that node can be deleted from the ranking list. The distance calculation can include the approximation parameter a, to allow for the fact that the closest data unit contained in a node is often significantly further away than the theoretical calculations can allow for.

A straightforward variation on this algorithm allows for finding the furthest k images instead of the closest.

4 Experimental Results

The search performance is perhaps most usefully measured in terms of the number of distance calculations made in order to find the k closest data elements to a specified key element. This is equivalent to finding the number of nodes (both internal and leaf nodes) in a tree that are touched during a particular search process. The measure gives a good indication of the comparative speeds of different search methods, yet is independent of any particular execution environment.

All the results in this section are bases on a 21-nearest neighbour search, where the key images is a member of the indexed data set. This simulates a 20-nearest neighbour search where the key images is not a member of the data set. The data set is searched for each data element in turn and the number of nodes searched, as shown in the figures, represents the average number of nodes searched for the index. The VAMSplit R-tree provides results that are largely independent of dimension and sub-linear in proportion to data set size. Therefore the C-tree results also display these qualities as shown in Figures 1 to 3.

4.1 VAMSplit R-tree vs. C-tree

In order to determine if the use of the C-tree can

improve search performance, several data sets are indexed using both the C-tree and the VAMSplir R-tree and the results compared in Figure 1. The data sets are composed of various feature vector sets for a collection of Japanese drawings which were made available. The results shown in the figure compare the percentage of nodes than must be searched in each tree during the 21-nearest neighbour search. The number of nodes searched in the VAMSplit R-tree is taken to be 100% in each case.

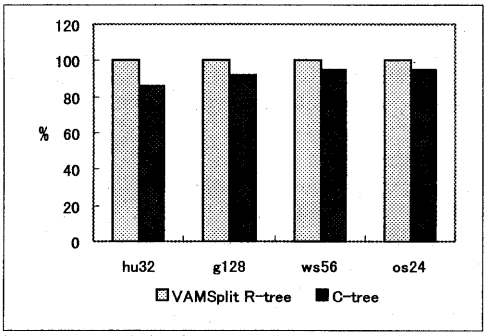

Figure 1. Search Performance Comparison between the VAMSplit R-tree and the C-tree.

hu32: A data set of 11620 elements in 32 dimensions representing the hue feature vector.
g128: A data set of 11620 elements in 128 dimensions representing the green colour feature vector.
ws56: A data set of 4132 elements in 56 dimensions representing the shape feature vector as extracted using wavelet analysis.
os24: A data set of 11690 elements in 24 dimensions representing the outer shape feature vector.

The C-tree requires fewer nodes in the tree to be visited in all cases, but the difference (ranging from eighty-five to ninety percent) is not as pronounced as had been hoped for. However, since a wide variety of data sets were not yet available for testing it is premature to draw decisive conclusions regarding the general relative performance of the trees.

The C-tree is an unbalanced tree, and therefore comparisons were also made between the minimum and maximum numbers of nodes searched in both the C-tree and the VAMSplit R-tree. It was feared that the unbalanced C-tree would show a wider variation in the minimum and maximum values, perhaps leading to worse search performance for some elements than the balanced R-tree. However, this was not proved to be the case. In approximately ninety per cent of all data sets tested the maximum number of nodes searched in the C-tree was less, often significantly less, than in the VAMSplit R-tree. In the remaining data sets the maximum never increased by more than two per cent. The minimum number of nodes searched is always significantly less in the C-tree. Further tests are planned to examine the variance of the search results in greater detail.

4.2 Data Set Size and Dimensionality

In order to obtain a sense of how the C-tree search performance is dependent on the data set size and dimension, a series of tests were carried out on the image feature vector sets as well as manufactured data sets. Figure 2 shows the change in the percentage fewer nodes searched for a change data dimension on the search performance. Again the percentage is measured relative to the performance of the VAMSplit R-tree. The initial feature vector is a 256 dimension vector representing the colour green. This was reduced to three alternative vector sets of dimensions 128, 64 and 32. It appears that the optimisation becomes less effective as the dimension of the data increases.

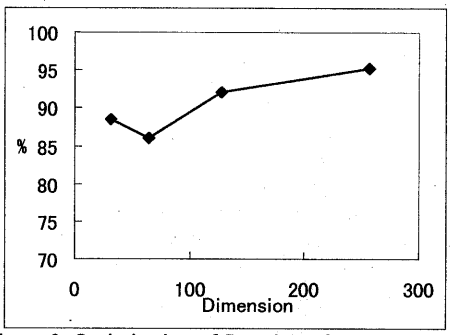

Figure 2. Optimisation of Search Performance as the Data Set Dimension Changes.

Figure 3 shows the effect of changing the size of the data set while the dimension of the data set is fixed at 4. The data used in this series of tests is artificially manufactured random data.

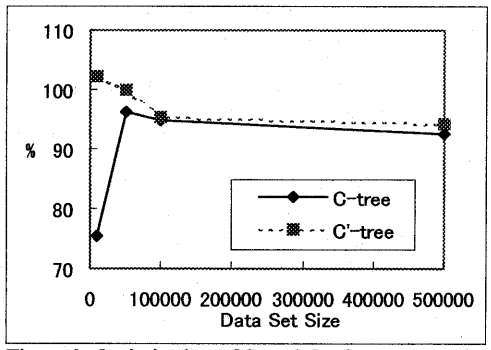

Figure 2. Optimisation of Search Performance as the Data Set Size Changes.

The optimisation of the search performance increases slightly as the size of the data set increases. This is important for databases that wish to index very large amounts of data in a single structure. However, further tests are needed to show that this is also true for data sets derived from actual images.

4.3 C-tree Construction Speed

A C-tree containing 10,000 data elements can be created

in less than 1 minute. However, a tree containing 100,000 elements typically takes up to 40 minutes to construct. (These tests were carried out on a Sun Sparc workstation) This is still significantly faster than the optimisation method suggested in [Gavrila 1994] and may be justified by the increased search performance. Alternatively, the larger tree can be subdivided into a set of sub-trees during the initial application of the VAMSplit R-tree algorithm and each subtree subjected to the rigorous C-tree construction method. In this case the construction time is reduced significantly to less than 10 minutes and the search results are still improved over the VAMSplit R-tree. The dashed line in Figure 3 represents the performance of the C'-tree index built in this way.

4.4 Other Observations

Preliminary trials showed that approximate searching could be used to great effect in both the C-tree and the VAMSplit R-tree with the number of nodes searched typically being cut by about 30% with negligible error when an approximation factor of 10% is allowed. It was also noted that the parameter that appeared to exert most effect on search performance was the node size chosen for the initial cluster identification using the VAMSplit R-tree algorithm.

5 Conclusions

The results of the similarity retrieval tests using the C-tree shows that it is possible to further optimise tree index construction using quasi-exhaustive iterative techniques when construction speed is not of primary importance. Previously such optimisation had been dismissed as too time consuming, but the C-tree provides an efficient construction algorithm that can achieve further optimisation in an acceptable time. By allowing the C-tree to become unbalanced it is thought that the volume of some minimum bounding hyper-rectangles can be significantly reduced as isolated nodes are not forced into a cluster. This leads to improved performance.

The C-tree provides increasingly improved performance as the size of the data set is increased, even when the optimisation process is only carried out on individual sub-trees in order to speed up construction time. However, initial results would indicate that the optimisation is reduced in effectiveness as the dimension of the data set increases. It is possible that this trend could be reversed if closer attention is paid to adjusting the clustering algorithm parameters in a dimension dependent manner. The optimisation would still appear worthwhile on small and medium dimension data sets.

There is much work that still needs to be done to optimise the automatic selection of construction parameters for both initial tree and optimisation phases of the C-tree construction process. The maximum node size allowed in tree construction, the minimum number of child nodes allowed in an internal node, the threshold distance for a data element's removal to the residue list and the number of iterations allowed in the optimisation process can all dramatically affect the search performance of the tree. It may well be that further improvements in performance will best be gained from looking at these parameters than by finding new ways to optimise the methodology of tree construction.

It is also important to look at the overall search strategy for an image database retrieval system. It is usually necessary for a single search to combine the results from several different indices. These indices may be of the same type, simply representing different types of feature vector, or they may be completely different indexing systems. The procedure to combine these different search results must also be optimised to provided an overall efficient search performance.

References

[Akama 1997] Akama H., Mii K., Konya S. and Kushima K. "ExSight - Image Retrieval System based on Automatic Object Extraction," *Proc. of IEICE DEWS 97*, March 1997.

[Anderberg 1973] Anderberg M.R., *Cluster Analysis for Applications*, Academic Press, 1973.

[Berchtold 1996] Berchtold S., Keim D.A. and Kriegel H., "The X-tree: An Index Structure for High-Dimensional Data," *Proc. of the 22nd VLDB Conference*, 1996.

[Gavrila 1994] Gavrila D.M., "R-tree Index Optimization," CAR-TR-718, University of Maryland Technical Report, June 1994.

[Guttman 1984] Guttman A., "R-trees: a Dynamic Index Structure for Spatial Searching," *Proc. of ACM SIGMOD*, 1984.

[Kamel 1994] Kamel I. and Faloutsos C., "Hilbert R-tree: An Improved R-tree Using Fractals," *Proc. of the 20th VLDB Conference*, 1994.

[Petrakis 1994] Petrakis E. and Faloutsos C., "Similarity Searching in Large Image Databases," *Technical Report CS-TR-3388*, University of Maryland, College Park.

[Roussopoulos 1995] Roussopoulos N., Kelley S. and Vincent F., "Nearest Neighbor Queries," *Proc. ACM SIGMOD*, 1995.

[Swain 1990] Swain M. and Ballard D., "Indexing Via Color Histograms," *Proc. Image Understanding Workshop*, pp. 623-630, 1990.

[Vinod 1996] Vinod V., Murase H. and Hashizume C., "Focussed Color Intersection with Efficient Searching for Object Detection and Image Retrieval," *Proc. IEEE Multimedia 96* , 1996.

[White 1996a] White D.A. and Jain R., "Algorithms and Strategies for Similarity Retrieval," *Technical Report VCL-96-101*, University of California at San Diego, 1996.

[White 1996b] White D.A. and Jain R., "Similarity Indexing with the SS-tree," *Proc. 12th IEEE International Conference on Data Engineering, New Orleans*, 1996.