

Affinity Hybrid Tree: An Indexing Technique for Content-Based Image Retrieval in Multimedia Databases

Kasturi Chatterjee and Shu-Ching Chen
Florida International University
Distributed Multimedia Information System Laboratory
School of Computing and Information Sciences
Miami, FL 33199, USA
{kchat001, chens}@cs.fiu.edu

Abstract

A novel indexing and access method, called Affinity Hybrid Tree (AH-Tree), is proposed to organize large image data sets efficiently and to support popular image access mechanisms like Content-Based Image Retrieval (CBIR) by embedding the high-level semantic image-relationship in the access mechanism as it is. AH-Tree combines Space-Based and Distance-Based indexing techniques to form a hybrid structure which is efficient in terms of computational overhead and fairly accurate in producing query results close to human perception. Algorithms for similarity (range and k-nearest neighbor) queries are implemented. Results from elaborate experiments are reported which depict a low computational overhead in terms of the number of I/O and distance computations and a high relevance of query results. The proposed index structure solves the existing problems of introducing high-level image relationships in a retrieval mechanism without going through the pain of translating the content-similarity measurement into feature-level equivalence and yet maintaining an efficient structure to organize the large sets of images.

1. Introduction

Owing to the recent advancement of hardware, storing a large amount of image, video, and audio data, or a combination of them has become quite common, which emphasizes the growing needs of developing efficient multimedia organization and retrieval systems. The huge size of multimedia data makes indexing a crucial component for fast and efficient retrieval processes. Content Based Image Retrieval (CBIR) with Relevance Feedback (RF) has been a popular method for image retrieval [4][6][9][11][15]. Hence, the need of an efficient index structure for images

supporting popular access mechanisms like CBIR arises. The main challenge of implementing such an index structure is to make it capable of handling high-level image relationships easily and efficiently during access. The existing multidimensional index structures support CBIR by translating the content-similarity measurement into feature-level equivalence, which is a very difficult job and can result in erroneous interpretation of user's perception of similarity.

There are several multidimensional indexing techniques for capturing the low-level features like feature based or distance based techniques, each of which can be further classified as a data-partitioned [1][5][8][19] or space-partitioned [10][13] based algorithm. Feature based indexing techniques project an image as a feature vector in a feature space and index the space. The basic feature based index structures are KDB-tree [13], R-tree [8], etc. The KDB-tree is a space partitioning based indexing structure; whereas the R-tree is a data partitioning based indexing structure. Later, the Hybrid Tree [3] made an attempt to combine these two structures together and overcome the drawbacks of each. It splits the node based on a single dimension, making the fan-out independent of the dimensionality. It also allows overlapping whenever a clean split makes the tree cascade down, thus solving the utilization problem. Distance based indexing structures are built based on the distances or similarities between two data objects. Some famous distance based indexing structures are SS-tree [17], M-Tree [5][19], vp-tree [18], etc. Among them, only the M-Tree guarantees a balanced structure as it is built in a bottom-up manner. In the M-Tree, the objects are partitioned on the basis of their relative distances, measured by specific distance functions (which are metric in nature), and these objects are stored in fixed sized nodes [5][19]. The leaf nodes store all the indexed objects represented by their keys or features; whereas the internal nodes store the routing objects.

However, none of the above discussed indexing struc-

tures considers the incorporation of the high level image concepts as perceived by the user into their framework efficiently without translating those concepts into their low-level equivalence like in [4][6][11]. They either ask the user to attach weights to the features during the query or generate a feature weight model by interpreting the images selected by the user as the most related ones to the user’s similarity perception. Such a technique is highly complicated as it is difficult and error-prone to attempt to translate user’s perception of content-similarity into feature-level equivalence. In this paper, a novel indexing and access technique called the Affinity-Hybrid tree (AH-Tree) is proposed which aims to overcome the above stated problem by combining the low-level feature and the high-level image relationship as it is in the index structure. AH-Tree is designed to combine the feature based and distance based indexing techniques and uses the high level image relationship to provide semantically related query results.

The novelty of the AH-Tree is in devising a way to combine Spatial and Metric Access methods to support a content-based retrieval mechanism which cannot be implemented efficiently by any one method individually. The high-level image relationship cannot be used by space based indexing structures independently and efficiently because the Spatial Access Methods [1][3][8][10][14] require the distances between objects to be strictly related to the object position in a low dimensional vector space. Hence, any high level image relationship needs to be translated across different dimensions, which is a very difficult mapping and often results in inefficient low level translation of the high level relationship. Though a distance based index can introduce the high-level image relationship, the computation overhead is very high when the pair-wise distances and high level relationships among all the data points in the database need to be computed. The proposed index structure makes an attempt to solve the above problems by combining the space based and distance based indexing structure to incorporate the high level relationship efficiently without involving high computation overheads. The distance based indexing structure is used to actually incorporate the affinity relationship, and the space based indexing technique filters the metric space, thus reducing the (dis)similarity calculation manifold. Hence, the AH-Tree efficiently improves the query result by capturing the user perception into the image index structure without incurring the overhead of high computation.

The high level image relationship used in AH-Tree is captured using the concept of affinity relationship, a parameter of the Markov Model Mediator (MMM) mechanism, that maps the low level features and high level concepts in CBIR [16] by capturing the image relationship as perceived by the user. The MMM mechanism builds an index vector for each image in the database and considers the relation-

ship between the query image and the target image. The main idea is *the more frequent two images are accessed together, the more related they are*. The relative affinity measurement ($aff_{m,n}$) between two images m and n is defined as follows:

$$aff_{m,n} = \sum_{k=1}^q use_{m,k} \times use_{n,k} \times access_k$$

Here, $use_{m,k}$ denotes the usage pattern of image m with respect to query q_k per time period, and $access_k$ denotes the access frequency of query q_k per time period. Any high level image relationship capturing mechanism similar to the affinity relationship can be used in the proposed index structure without major changes.

The remainder of this paper is organized as follows. The actual representation of the proposed AH-Tree structure is discussed in details in Section 2. Section 3 introduces the algorithms to implement the range queries as well as the k-NN queries. Section 4 presents the implementation of the AH-Tree. A brief conclusion and scope for future work are presented in Section 5.

2. The Affinity Hybrid Tree

The Affinity Hybrid Tree (AH-Tree) is an indexing technique which supports content-based image retrieval by mapping the low level features with high level concepts. This index structure has a space indexing technique to index the data spaces and form the subspaces containing the actual data points. For each subspace thus formed, a metric tree is built using the metric distance functions among the data points. For range and nearest neighbor queries, the AH-Tree uses affinity relationship, a value determining how frequently two images are accessed together during CBIR, to prune the metric tree. The AH-Tree is a balanced structure as it uses a bottom-up technique to be built.

The affinity value is promoted from the leaf nodes to the parent nodes till the root node of the metric tree to incorporate the high-level image relationship at all levels of the AH-Tree structure and to maintain the metric distance function at the same time. Thus, the routing objects at the intermediate nodes get an affinity value. The details is discusses in Section 2.3.

2.1. AH-Tree Structure

A schematic representation of the tree structure is presented in Figure 1. Instead of storing the pointer to the data objects, the Data Nodes of the space-based index structure store the pointers to the root of the distance-based index structure built with these data points. If the number of

data objects in a particular Data Node of the space indexing structure is less than some threshold value (depending upon the total number of image objects indexed), the sibling data nodes are merged together and the distance based index structure is built with the data points of the merged leaves. This is a logical merge and the actual structure of the Hybrid Tree remains the same so as to keep it balanced. The *affinity* and *maximum affinity* values are stored in the leaf nodes and promoted at the non-leaf nodes of the distance based index structure for utilization during similarity searches. The algorithm to build an AH-Tree from a set of feature vectors is described in Table 1.

Table 1. Implementation of the AH-Tree

<pre> Feed the data points in the form of Feature Vectors and build the Space Index; if (# data points in any particular Data Node $S_1 < \eta$) { //where $\eta = \text{Threshold Value}$. Merge sibling Data Nodes S_1 and S_2; Set Pointer of both S_1 and S_2 to Ptr (M_1); } else { Set Pointer of S_1 to Ptr (M_1); Build metric tree M_i for each Data Node S_i; Set Root of the AH-Tree=Root of the Space Index Tree; Set Leaf=Ground Nodes of the metric tree; Set Intermediate Nodes =[Index Nodes of Space Index Tree, Data Nodes of Space Index Tree, Routing Nodes of metric tree]; } </pre>

2.2. Incorporation of Affinity Values

AH-Tree uses a metric tree to embed the high level image relationship in the index structure. A metric tree indexes a metric space formed by points which are related to each other by a metric distance function. A distance function is called metric if it obeys the laws of *symmetry*, *positivity* and *triangular inequality*. In the proposed AH-Tree, while building the M-Tree for each subspace of the space based index structure, the *affinity relationship* could not be incorporated within the distance function in order to keep the distance function between the data points metric. If the *affinity relationships* between the data points are used as a factor to scale the distance between them (higher the affinity value, lower is the computed distance), the *triangular inequality* of the distance function demands the factor to be the same. The affinity value is a high level concept depicting the similarity between each pair of images as perceived by the user and hence cannot be equal to each other. If the affinity, or in other words the user concept of similarity, could have been

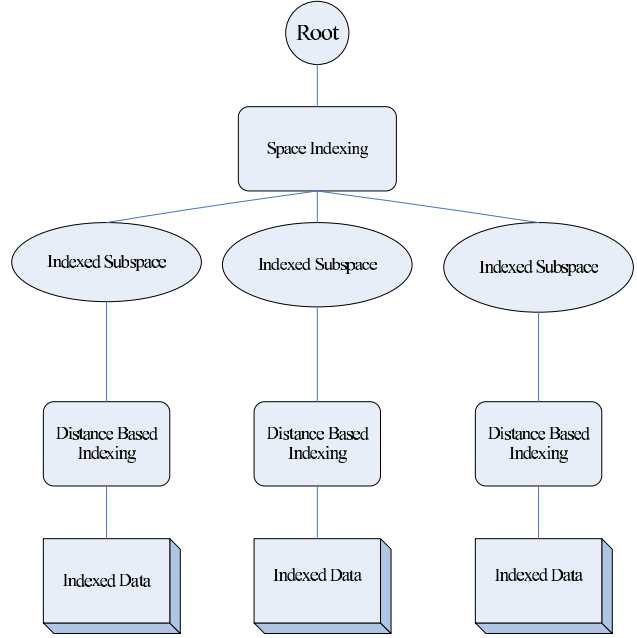


Figure 1. Structure and Component of AH-Tree

projected in the feature level, the distance function could have been scaled using them by attaching different weights to the feature values as discussed in [4]. However, the main motive of this index structure is to incorporate the user perception in its high level form and not to translate it. Hence, the affinity relationship is incorporated after the index structure is formed during each query for pruning the tree by a method called affinity promotion. The detailed derivation of the above claim is described in Lemma 1.

Lemma 1 *The affinity relationship cannot be involved while constructing the metric tree as it no longer keeps the search space metric.*

Proof: Let O_1 , O_2 and O_3 be three objects in a metric space. Let D_{13} , D_{12} , and D_{23} be the original distances computed before the introduction of the affinity values. Let K_{13} , K_{12} and K_{23} be the affinity factors used to scale down the distance and β_{13} , β_{12} and β_{23} be the newly computed distance functions. Therefore,

$$\beta_{13} = \frac{D_{13}}{K_{13}}, \beta_{23} = \frac{D_{23}}{K_{23}}, \beta_{12} = \frac{D_{12}}{K_{12}} \quad (1)$$

Thus,

$$D_{13} = K_{13}\beta_{13}, D_{23} = K_{23}\beta_{23}, D_{12} = K_{12}\beta_{12} \quad (2)$$

According to *triangular inequality*,

$$D_{13} \leq D_{12} + D_{23}, D_{12} \leq D_{13} + D_{23}, D_{23} \leq D_{12} + D_{13} \quad (3)$$

Thus,

$$\beta_{13} \leq \left(\frac{K_{12}}{K_{13}}\right)\beta_{12} + \left(\frac{K_{23}}{K_{13}}\right)\beta_{23} \quad (4)$$

$$\beta_{12} \leq \left(\frac{K_{13}}{K_{12}}\right)\beta_{13} + \left(\frac{K_{23}}{K_{12}}\right)\beta_{23} \quad (5)$$

$$\beta_{23} \leq \left(\frac{K_{12}}{K_{23}}\right)\beta_{12} + \left(\frac{K_{13}}{K_{23}}\right)\beta_{13} \quad (6)$$

Hence, to maintain the triangular inequality of the weighted distance function, we gather from equations (4, 5 and 6),

$$\frac{K_{12}}{K_{13}} = 1, \frac{K_{23}}{K_{13}} = 1 \quad (7)$$

The above proves that the affinity factor should be the same to maintain the triangular inequality.

2.3. Promotion of Affinity Values

In order to incorporate the affinity relationship in the AH-Tree, they should be promoted from the leaf level to each internal node level up to the root of the metric tree during a query. Before starting the query, for each query object (O_q), the affinity values of the data points at the leaf level with respect to O_q are promoted as discussed in Definition 1. The affinity value of a leaf node along with the affinity value of its sibling is used to compute the affinity value of their parents. The process continues till the root of the metric tree is reached. Such a promotion ensures that if any of the child nodes has an affinity value greater than or equal to the required affinity value with the query image, the corresponding parent node should be visited to avoid any false dismissal. It also ensures that if none of the children of a particular parent has an *affinity value* greater than or equal to the supplied query *affinity value*, the parent along with all its children need not be visited which saves the distance computations overhead during similarity search. The promotion of the affinity value is implemented in the Similarity Queries using the function Affinity_Promotion() which follows the technique discussed in Definition 1.

Definition 1 Let N_a and N_b be the leaf nodes of an M -Tree containing the indexed objects O_a and O_b respectively represented by their keys or features, and N_r be the parent of N_a and N_b . Let $aff_{a,q}$ and $aff_{b,q}$ be the precomputed affinity values between the query object and the objects at the leaf level. Hence, the affinity value of the parent of N_a and N_b (i.e., N_r) with respect to the query O_q is equal to $\max(aff_{a,q}, aff_{b,q})$.

Table 2. Implementation of Range Query in AH-Tree

```

AH_Range_Query(R(Q):query_region,
r(Q):search_radius, Q:query_object, aff:affinity_value,
N:node) {
  if (N is Null) {terminate;}
  else
  {
    Let page=root page;
    RNF=BR corresponding to N;
    Space_Search(R(Q), N, RNF);
    //space search sub-routine.
  } //end of space search.
  Set Nchild=Root_Metric;
  Metric_Search(Q, Nchild, r(Q), aff);
  //metric search.
} //end of AH_Range Search.

```

Table 3. Implementation of Space Search in AH-Tree

```

Space_Search(R(Q), N, RNF) { //Space Search.
  if (N ≠ Space_Data_Node)
  {
    I = RNF ∩ R(Q);
    if (I ≠ ∅)
    {
      ∀ child nodes in N {
        Compute Rchild from RNF;
        Set RNF=Rchild;
        Space_Search(R(Q), Nchild, RNF);
      }
    }
  }
} //end of Space Search Subroutine.

```

3. Similarity Queries

The proposed AH-Tree supports both the range queries and the k-NN queries. Before going into the detailed algorithms, the tree traversal and the node information processing are discussed in each case. A query is represented as a collection of features $Q(F)$, where F is the same set as the image feature vector and is extracted in the same manner as the image objects. Once the feature vector of the query image is obtained, the AH-Tree is traversed from its root to the subspaces of the feature space containing data points related to the query object. The metric trees corresponding to subspaces having the maximum number of data points are merged and the affinity relationships of all the nodes

in the metric tree are computed by affinity value promotion technique. By computing the distance and the affinity between the query object and the tree objects of the metric tree, the query result is obtained. A detailed pseudo-code of the range and k-NN queries for the AH-Tree is presented in the following subsections.

3.1. Range Queries

A range query ($Q, r(Q)$) traverses through the AH-Tree and selects all the appropriate database objects (O_i) which satisfy the following condition:

$$\forall O_i, d(O_i, Q) < r(Q).$$

The AHRange_Query as discussed in Table 2 for the AH-Tree was developed so as to implement the range query in the feature space as well as in the metric space. Since the space-based indexing technique requires a search range and a metric-based indexing technique requires a search radius to implement the range search, both the values are provided while initializing the range search algorithm for the AH-Tree. The algorithm for the range query is described in details in Table 2, 3 and 4. The AHRange_Query first performs range search on the feature space to get the feature sub-spaces within the supplied range of the query using the function Space_Search($R(Q), N, R_{NF}$) as discussed in Table 3. Once the feature subspaces are obtained, in order to increase the metric search space, neighboring feature subspaces are combined in a step-wise manner depending upon the user input starting with just the original result obtained from the space search. The metric search method includes the introduction of the affinity concept. For the router objects i.e. the intermediate objects, the similarity distance is first evaluated against the search radius. If satisfied, the affinity of the routing object with the query object is checked against the required supplied affinity value. Upon satisfying both the conditions, the metric search is iterated for the subtree of the routing object. The metric search is implemented in the function Metric_Search($Q, N_{child}, r(Q), aff$) discussed in Table 4. In many cases, providing appropriate search radius with the query is rather difficult and do not result in satisfactory query output. Such a scenario is taken care by maintaining a second parallel result set formed depending upon only the affinity requirement fulfillment even when the similarity distance do not fall within the search radius. This result set is used if the user is not satisfied with the earlier result set. This gives the high level image relationship a greater importance in determining the query result when the low level feature relationships fails to produce a satisfactory result set. For data objects residing at the leaf nodes, similar evaluation is performed except that the image objects are added to the result

set directly when evaluation is successful instead of initiating an iterative metric search. For image objects without affinity values (possible if a new image object is introduced whose affinity value is not available), simple metric search is performed depending upon the classical similarity evaluation.

Table 4. Implementation of Metric Search in AH-Tree

```

Metric_Search(Q, Nchild, r(Q), aff) { //Metric Search.
  Affinity_Promotion(); //promotion of affinity value.
  if (aff(Or, Q) ≠ 0) { //affinity value available.
    if (Or is a routing object){
      ∀ Or in Nchild do: {
        if (| d(OM, Q) - d(Or, OM) | ≤ r(Q)+r(Or)) {
          Compute d(Or, Q) and aff(Or, Q);
          if ((d(Or, Q) ≤ r(Q)+r(Or)) &&
              (aff(Or, Q) ≥ aff)) {
            Metric_Search(ptr(T(Or)), Q, aff);
            //T(Or): pointer to the subtree.
          }
        }
        elseif (aff(Or, Q) ≥ aff){
          //giving affinity relationship greater
          //priority over similarity distance.
          Metric_Search(ptr(T(Or)), Q, aff);
          //T(Or): pointer to the subtree.
        }
      } //end of search for Or
      //satisfying metric condition.
    } //end of search for all Or in Nchild.
  } //end of internal node search of the
  //metric tree.
  elseif (Or is a leaf object){
    If the object qualifies the distance function
    and the affinity, add to the result set;
  }
  } //end of search for query object with affinity.
  else {
    Metric_Search with the absence of the affinity
    comparison;
  } //end of search for query object without affinity.
} //end of Metric Search Subroutine.

```

3.2. k-NN Queries

The AH_k-NN_Search algorithm as discussed in Table 5 retrieves k nearest neighbors from the AH-Tree for a query object Q. The AH-Tree uses a branch-and-bound technique similar to the one designed for the R-Tree [8]. The algorithm proposed here to implement the k-NN query on the AH-Tree first determines the k-nearest subspaces to a given

query point. Then it merges the metric tree corresponding to each space ultimately performing k-NN search on the combined metric tree thus formed to get the k nearest neighbor objects. The search algorithm implements an ordered depth-first-search on its feature space using the function `Space_Nearest_Search(N, nearest, Q)` discussed in Table 5. During traversal, at each non-leaf node, the metric bounds are calculated between the query point and all its Minimum Bounding Regions (MBRs) and stored in an ordered list. The list is pruned depending on the similarity measure and the search iterates upon this list until it is empty. In each iteration, the next branch belonging to the particular MBR is selected. On reaching the data nodes of the feature based index structure, the value of the nearest distance is updated and the iteration continues until k feature subspaces are obtained. The metric tree corresponding to each feature subspace thus obtained is then combined. The affinity values of the combined metric tree is promoted from the leaf levels. A priority queue is maintained which points to the active sub-trees of the metric tree. The function `Metric_Nearest_Search(N, k, Q)` discussed in Table 5 implements the metric search in the metric space. The search radius and the affinity value now become dynamic in nature and are defined in Definitions 2 and 3 respectively.

Definition 2 *The search radius is defined as the distance between the query point and the current k-th nearest neighbor.*

Definition 3 *The affinity value is defined as the affinity between the query point and the current k-th nearest neighbor.*

For an iteration of the priority queue, the sub-tree with the minimum distance and maximum affinity with the query point is chosen. The minimum distance and the maximum affinity thus obtained become the search radius and the affinity value for the next iteration. The pseudo-code is described in Table 5.

4. Implementation of the AH-Tree

In this section, we provide the detailed implementation of the AH-Tree and an analysis of the experimental results. The H-Tree and M-Tree packages [2][12] were used as a framework upon which the AH-Tree application was built using C++ in an Linux environment. A node size of 4 Kbytes was used. The image database used has 10,000 color images of 72 semantic categories. The feature matrix is developed by obtaining the color information for each image from its HSV color space. Twelve color features are considered which makes the feature matrix 12-dimensional. An affinity relationship matrix of dimension 10,000 X 10,000 is used which is precomputed from a training set capturing the user perception. We performed extensive experiments

Table 5. Implementation of k-NN Search in AH-Tree

```

AH_k-NN_Search(N:node, nearest:distance,
Q:query point, k:number of nearest neighbors){
  if (N is Null) {terminate;}
  else
  {
    Space_Nearest_Search(N, nearest, Q);
    //Returns k nearest space, searching
    //the Space Indexing Tree by generating
    //Available Node List and Sorting them
    //based on the similarity measure iteratively.

    //Combine the metric trees corresponding to
    //k Spaces.

    //Search on the corresponding metric tree.
    Affinity_Promotion( );//promotion of the
    //affinity value.
    //Perform the metric search as explained in
    //range search with the difference of making
    //the search radius dynamic by making it the
    //distance between Q and the current kth
    //nearest neighbor and storing all the non
    //leaf nodes with the required similarity
    //measurement in a priority queue.
    Metric_Nearest_Search(N, k, Q);
  }
} //end of k-NN search.

```

to evaluate the performance of the AH-Tree during its construction and during queries. The implemented AH-Tree is compared with the performance of M-Tree both during construction and during queries. The AH-Tree structure is not compared with the Hybrid Tree or any other Space-Based indexing structure as it has already been discussed that the high-level image relationship introduced in the AH-Tree cannot be utilized in any Space-based indexing technique without translating it to its low level equivalence.

The experimental results imply that AH-Tree is capable of overcoming the problems discussed above by combining the space-based and distance-based indexing structure. Figure 2(a) depicts the distance computation and number of I/O vs number of objects for M-Tree and AH-Tree. It clearly indicates that by using the space based indexing structure to filter the feature space prior to building the M-Tree for each subspace, there has been a drastic reduction in computation overheads.

Experiments are carried to implement both range as well as k-NN queries using 10 query images each for both AH-Tree and M-Tree with k=10. The distance computations and

number of I/O averaged over 10 queries of the range and k-NN query is plotted in Figure 2(b) and Figure 2(c). This also demonstrates that the AH-Tree performs far better as far as overhead is concerned.

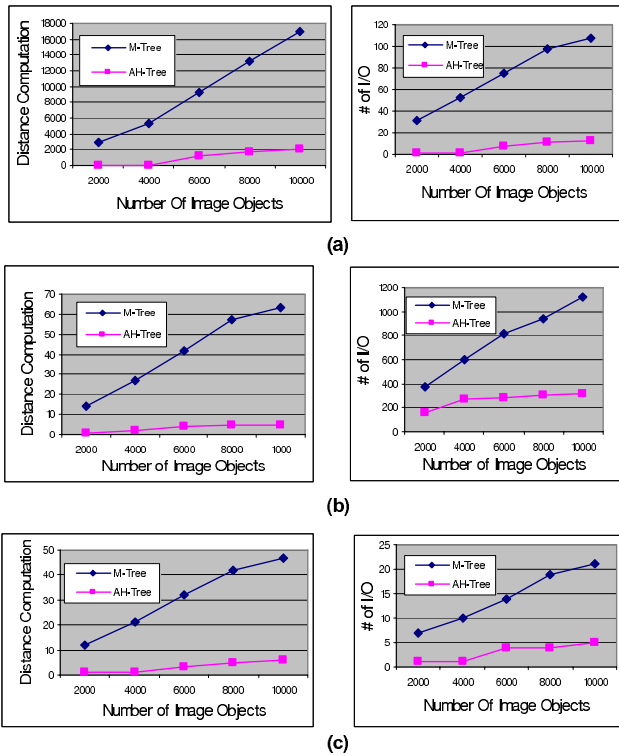


Figure 2. Distance Computation and Number of I/O during (a) building the Index Trees, (b) Range Queries, and (c) k-NN Queries

Since the novelty of the paper is the introduction of the high level image relationship in the index structure to facilitate getting semantically related query results without translating them into their low-level equivalence, hence the query results obtained from the AH-Tree as well as those obtained from M-Tree are checked for accuracy against images annotated manually. The *accuracy* is defined as the *percentage of the retrieved images that are semantically related to the query image*. It is noted that the results obtained from the M-Tree do not exhibit any regular pattern of semantic relationship and has an accuracy as low as 10% on an average. AH-Tree on the other hand has an average accuracy over 80% which is depicted in Figure 3 for an example 10-NN query where the query image is at the top left-most corner. It can be seen that about 8 among the 10 retrieved images have a close semantic relationship (animals in natural surroundings) and hence possess an accuracy of 80% for this example. The result is ranked in an order of decreasing

similarity from left to right and top to bottom. Such stark difference in the accuracy of obtained query results is clearly due to the introduction of high level image relationship.



Figure 3. Query Results for 10-NN Query

The above analysis of the experimental results help us to conclude that the proposed AH-Tree indeed performs better both in terms of computation overhead and relevance of the query results. Thus, it achieves the two essential goals of any multimedia indexing structures. First, it reduces the computation time in retrieving multimedia objects and second, it makes the retrieved result as close to human perception as possible.

5. Conclusion and Future Work

In this paper, an efficient indexing and accessing technique, called the AH-Tree, is proposed. The AH-Tree combines low level features and high level image relationship and supports CBIR by integrating feature based and distance based indexing. To the best of our knowledge, the AH-Tree is the first attempt to combine the feature based and distance based multidimensional indexing techniques to introduce high level image relationships, making the query results more semantically related in an efficient way. The experimental results demonstrate that the proposed AH-Tree is a promising indexing mechanism to bridge the gap between the low level features and the high level relationship among the images, and has a lot of potentials for future research and development. Presently, the *affinity relationship* is calculated offline. As a part of our future work, a learning technique to determine the affinity relationship in real time will be investigated to extend the capability of the indexing mechanism in capturing the high level image relationships efficiently.

Acknowledgment

For Shu-Ching Chen, this research was supported in part by NSF EIA-0220562 and HRD-0317692. We would like to thank Dr. Marco Patella, who is a researcher at DEIS, University of Bologna, Italy for his invaluable discussions and suggestions while understanding the basic framework of M-Tree.

References

- [1] S. Berchtold, D. A. Keim, and H. Kriegel. The x-tree: an index structure for high dimensional data. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 28–39, Bombay, India, September 1996.
- [2] K. Chakrabarti. Hybrid tree code. <http://www.ics.uci.edu/kaushik/research/htree.html>, 2005.
- [3] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high-dimensional feature spaces. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 440–447, Sydney, Australia, March 1999.
- [4] K. Chakrabarti, K. Porkaew, M. Ortega, and S. Mehrotra. Evaluating refined queries in top-k retrieval systems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(2):256–270, February 2004.
- [5] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, pages 426–435, Athens, Greece, August 1997.
- [6] R. Fagin. Fuzzy queries in multimedia database systems. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 1–10, Seattle, Washington, United States, June 1998.
- [7] D. Greene. An implementation and performance analysis of spatial data access methods. In *Proceedings of ICDE*, pages 606–615, Los Angeles, California, United States, February 1989.
- [8] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, Massachusetts, United States, June 1984.
- [9] R. Krishnapuram, S. Medasani, J. Hwan, C. Y. Sik, and R. Balasubramaniam. Content based image retrieval based on fuzzy approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(10):1185–1199, 2004.
- [10] D. B. Lomet and B. Salzberg. The hb-tree: A multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, 1990.
- [11] A. Motro. Vague: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, 1988.
- [12] M. Patella. M-tree code. <http://www-db.deis.unibo.it/Mtree>, 2005.
- [13] J. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, Ann Arbor, Michigan, United States, April 1981.
- [14] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of Data*, pages 71–79, San Jose, California, United States, May 1995.
- [15] Y. Rui, T. Huang, and S. Mehrotra. Content based image retrieval with image retrieval in mars. In *Proceedings of International Conference on Image Processing*, pages 815–818, Santa Barbara, California, United States, October 1997.
- [16] M.-L. Shyu, S.-C. Chen, M. Chen, C. Zhang, and C.-M. Shu. MMM: A stochastic mechanism for image database queries. In *Proceedings of the IEEE Fifth International Symposium on Multimedia Software Engineering (MSE2003)*, pages 188–195, Taichung, Taiwan, ROC, December 2003.
- [17] D. A. White and R. Jain. Similarity indexing with sstree. In *Proceedings of the 12th International Conference on Data Engineering*, pages 516–523, New Orleans, LA, United States, February 1996.
- [18] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, Philadelphia, PA, United States, January 1993.
- [19] P. Zezula, P. Ciaccia, and F. Rabitti. M-tree: A dynamic index for similarity queries in multimedia databases. In *Technical Report 7, HERMES ESPRIT LTR Projects*, 1996.