On Index Methods for an Image Database

Rui Mao[1], Wenguo Liu[1], Qasim Iqbal[2], Daniel P. Miranker[1]
[1] Department of computer sciences, University of Texas at Austin
{rmao, liuwg, miranker}@cs.utexas.edu

[2] ClearCube Technology, Austin, Texas
qiqbal@clearcube.com

Abstract

We investigate the performance of three index structures to support scalable proximity retrieval of images. The development of a general purpose tree-structured database index to support proximity search in metric space with formal O(log n) performance guarantees has eluded researchers. Consequently, there has been a proliferation of heuristic-based index methods whose performance must be evaluated with respect to individual workloads. We measure the performance and scalability of two variations of M-Trees and Multi-Vantage-Point Trees loaded isotropic and anisotropic image descriptors.

1. Introduction

Today, a great challenge in database area is to manage various nontraditional types of data, such as multimedia, video, image, voice, text and biological data types, sequences, protein structure and mass spectra [17, 18]. Among the queries of these data types, content-based retrieval plays a dominant role. Answering this kind of query requires computing the relative distances between data objects, which is typically a very costly operation. As distance measures are refined, the interest will be in large disk-resident data sets. Retrieval will have to be supported by external data structures that optimize both disk I/O and the number of distance calculations.

Traditional index structures (such as B+-tree) can handle traditional, ordinal, data types, such as numeric data, string data, etc, however, these index structures do not support spatial database search. For spatial access geographic information systems exploit R-trees [10] and k-d trees. However, the tree indexes are not suitable for high-dimensional space data search. Consequently, nontraditional data types require new forms of indexing. One approach is metric space indexing.

Definition 1: A **Metric Space** is a set of data objects together with a distance function, *d*, with the following properties [5, 8]:

(i). $d(O_x, O_y) = d(O_y, O_x)$ (symmetry)
(ii). $0 < d(O_x, O_y)$, $O_x, \neq O_y$, and $d(O_x, O_x) = 0$ (non-negativity)
(iii). $d(O_x, O_y) \leqslant d(O_x, O_z) + d(O_z, O_y)$ (triangle inequality)

There are two important types of similarity queries in Metric Space, i.e., range query and *k*-nearest neighbor query:

Definition 2: Range query range(Q,r): find all data objects O such that $d(Q, O) \leqslant r$; ***k*-Nearest Neighbor query $NN_k(Q)$:** find *k* data objects such that they are the *k* closest ones to Q [5, 8].

Since nearest-neighbor queries can be systematically implemented by range queries [5], to simplify the discussion, we only consider range queries in this paper.

The key difference between answering a range query in metric space and in a dimensional (Euclidean) space is that there does not exist a total ordering of the data that preserves the relative similarity [9]. The only way we can prune distance calculations during search of a metric space is to make use of the triangle property of the distance function [5]. The value of a metric-space approach is that it is unnecessary to find a meaning for the data with respect to the axes of a coordinate system.

Ciaccia et al.'s M-tree effort stands out as the single investigation of a fully general external metric-space index structure [7, 8]. M-tree is a radius-based structure with dynamic capabilities [8]. It is page mapped to disk to manage large data sets. However, we tried M-tree for some real data sets, and the results were not satisfying. We then developed some improvements on M-trees. Further, we have implemented a Multi-Vantage Point index tree algorithm. We compare the performance of the three methods on an image database.

In radius-based indexing methods, the data is hierarchically clustered, where each cluster is defined by a center and the maximum distance from the center to any data element in the cluster, *the radius* [5, 7, 8]. The center and the radius define a bounding sphere. The hierarchical structure is materialized as a tree, each vertex of the tree entailing a bounding sphere. If a query point is too far from the center of a cluster, by virtue of the triangle inequality, all the points in the sub-tree, (within the bounding sphere) may be removed from consideration without further distance calculations. I.e. the tree branch may be pruned.

A challenge we noted in M-trees is that a single outlier in a cluster may greatly expand the volume of a bounding sphere. Bounding spheres may overlap, and when they do search pruning is inhibited. Since the clustering is driven by the physical volume of a disk page, the introduction of outliers into clusters is common, and the performance of M-tree's sensitive to the initial clustering of the data. In the context of MoBIoS (Molecular Biology Information System) [16] we optimized the node structure, developed an improved initialization scheme for M-tree's and refined the method for calculating the radius of a cluster [17, 18].

A multi-vantage point tree, (MVP-tree), is built by first selecting some number of points, the vantage points. The distance range from each vantage point is broken into intervals. The Cartesian product of the intervals forms a set of data partitions. Each data element is allocated to a partition by calculating its distance to each vantage point [2, 3, 5]. The construction is applied recursively to form an index tree. See Bozkaya and Ozsoyoglu for details [2, 3]. Our MVP-tree implementation accounts for elements of an external data structure by organizing the leaves of the tree as data-pages.

MVP-trees have a complementary problem with M-tree's. The data partitions are guaranteed to be disjoint. For larger range queries, the bounding sphere representing the query may overlap many MVP-tree partitions, also mitigating search pruning. This paper details this trade-off with respect to an image retrieval system.

The paper is organized as follows. In Section 2, we present the image distance function and prove that it is metric. M-tree and Vantage point tree structures, together with their variations are analyzed in Section 3, followed by experimental results in Section 4. Section 5 consists of conclusions and future work.

2. Image Distance Function

To compute the distance between images, each image is represented by 3 sets of features reflecting the image properties in structure, color, and texture. Feature retrieval is based on isotropic (structure extraction and color histogram) and anisotropic mappings (texture analysis) [13]. Isotropic mappings are defined as mappings invariant to the action of the planar Euclidean group on the image space -- invariant to the translation, rotation, and reflection of image data. Anisotropic mappings, on the other hand, are defined as those mappings that are correspondingly variant. For detailed information on how to retrieve features from images, please refer to [13, 14].

The 3 sets of features, actually 3 vectors for each image, correspond to three different metric spaces (structure space, color space and texture space). To compute the distance between two image objects, two distance functions are defined.

The distance in structure space and texture space are computed with the Euclidean norm,

$$d(u,v) = \|u - v\|_2 = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}. \quad (1)$$

Meanwhile, the distance in color space are computed with the distance function,

$$d(u,v) = 1 - \sum_{i=1}^{n} \min(u_i, v_i), \quad (2)$$

where n is the dimension of the image features, u and v are vectors representing the features of two images, respectively. (1) represents a L2 norm, while (2) is a L1 norm.

After the distances are computed in each metric space, three weights are used to combine these quantities together to get the final distance between two image objects. These weights should sum to 1.0. In our current model, a weight of 1/3 is used for each of these metric spaces.

3. Index Structures

All tree-based database index structures may be generalized as follows: Consider a data set S of n objects. Arbitrarily partition S into blocks, such that the content of each block can be qualified by a predicate P. For example, if the data type is rectangles on a plane, then P may be the minimum-bounding rectangle that covers all rectangles in the block. If there are B objects per block, roughly n/B blocks are created. Consider as a new data set the n/B predicates describing these blocks, and repeat the process until a single block, the root, is formed. The result is a balanced tree of height $\log_B n$.

We study two primary methods used to define the predicates for metric-spaces: radius-based and vantage points. In radius-based structure [5, 7, 8], each partition satisfies a predicate P(C, r), i.e., C is the center of the partition and the r is the covering radius. This predicate means that any data object in the partition is within the distance r to C. In

vantage point methods [2, 3, 5], each partition satisfies a predicate P([V_i, $r_{min,\,i}$, $r_{max,\,i}$]), i.e., each distance from the vantage point V_i to each point in this partition is within the range ($r_{min,\,i}$, $r_{max,\,i}$). Details of the two structures are analyzed in the following.

3.1 Radius-based structure

We first discuss the node structure of a radius-based metric-space index, and introduce initialization algorithms. Both variations of the M-tree use the same search rules as the original M-tree algorithm [8]. In radius-based index tree, internal nodes and leaf nodes have different structures, which are shown in Figure 1.

C	R	$child_1$			...			$child_n$		
		c_1	r_1	d_1	...	...	…	c_n	r_n	d_n

(a) Internal node

C	R	$child_1$		...		$child_n$	
		c_1	d_1	...	...	c_n	d_n

(b) Leaf node

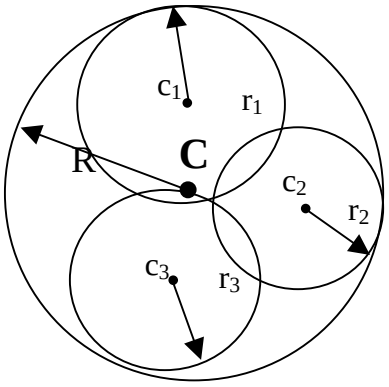

Figure 1 Radius-based index tree structure

Both internal nodes and leaf nodes have a center, C, and a covering radius, R, within which any data object in the sub-tree of the node has a distance to the center. The data fields of an internal node are shown in table (a) of Figure 1, where $child_1$, …, $child_n$ are pointers to all children nodes, c_1, …, c_n are centers of all children nodes, r_1, …, r_n are covering radii of all children nodes, and d_1, …, d_n are the distances from the center of each child to the center of the node. In leaf nodes, their children are data objects. As shown in table (b) of Figure 1, $child_1$, …, $child_n$ are pointers to all data objects, c_1, …, c_n are the index keys of all data objects, and d_1, …, d_n are the distances from the center of the node to the index keys of every data objects.

Both internal nodes and leaf nodes are sized as disk pages, so that to read or write a node needs exactly one I/O operation. Furthermore, the number of children of an index node is also a function of a page occupancy parameter.

Initialization of a radius-based tree comprises a hierarchical clustering of the data. We consider the original M-tree initialization of repeated inserts. This is equivalent to a greedy, bottom-up clustering of the data. Since an insert has amortized time complexity O(log n), the total initialization time complexity is O(nlog n). We have evaluated M-tree's explicit top-down bulk-loading algorithm. The bottom-up method performed better, so we do not detail those results.

Our improvement on M-tree comprises a bi-directional initialization algorithm. Briefly, bi-directional is a bottom-up method. For each level of the tree, starting at the bottom we first run a simple top-down hierarchical clustering algorithm. The clusters returned by simple top-down first form the leaves of the tree. Then, the predicates for the set of leaves

are treated as data, and reapplying the top-down clustering algorithm we construct the clusters that form the first level of interior nodes. The procedure repeats recursively, terminating when the predicate set fits on one disk page. The algorithm to cluster a dataset to sub-clusters is farthest-first-traversal [12]. It is a greedy k-center algorithm, and is guaranteed to produce clustering result within a constant factor of two of optimal.

Further, we refine M-tree's determination of the radius by setting it as the maximum distance between the center and each data objects in the sub-tree. In M-tree, the radius is the maximum sum of the radius of a child and the distance between the centers of the child and the parent, i.e., $r = \max \{r_i + d(C, c_i)\}$.

3.2 Vantage point tree and variation

The precise form of the bounding predicates in an MVP-tree is: $P([V_i, r_{min,\ i}, r_{max,\ i}])$. Specifically, the predicate comprises, for each vantage point V_i the distance from V_i to any data object in the sub-tree of the node is within the range $[r_{min,\ i}, r_{max,\ i}]$. We refer the reader to Bozkaya and Ozsoyoglu for detail of the search process [2, 3]. Although our implementation remains in main-memory we organized our node structures and initialization algorithm in a manner consistent with disk-mapped storage structures.

	child$_1$	**...**	**child$_n$**
VP$_1$	$d_{1,1}$ [min,max]	…	$d_{1,n}$ [min,max]
…	…	…	…
VP$_n$	$d_{n,1}$ [min,max]	…	$d_{n,n}$ [min,max]

(a) Internal node

	child$_1$	**...**	**child$_n$**
VP$_1$	$d_{1,1}$	…	$d_{1,n}$
…	…	…	…
VP$_n$	$d_{n,1}$	…	$d_{n,n}$

(b) Leaf node

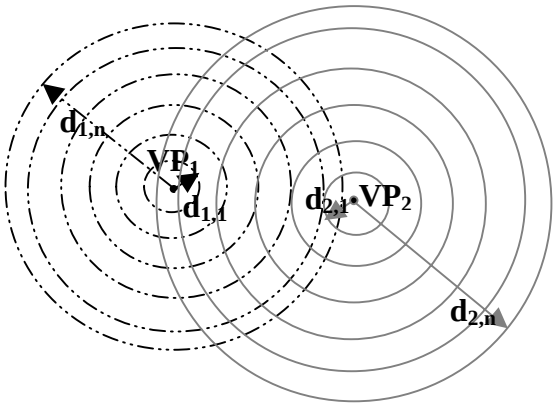

Figure 2 Vantage point index tree structure

The node structure of an MVP-tree is shown in Figure 2. There are two key differences between our node and node structures in [2, 3]. First, to accommodate the large data set that needs to use secondary storage, the sizes of internal nodes and leaf nodes are equal to the disk page size so that to read or write a node needs exactly one disk I/O. Second, in leaf nodes in [2, 3], each data object stores a list of distances between it and the vantage points of the nodes on the path from the leaf to the root of the tree. This list of distances is used to save some distance calculations when the leaf node is searched. However, more space is needed to accommodate a data object in the leaf node, and thus reduce the fanout for a fixed node size. Therefore, we don't use the distance list in the leaf node.

To initialize the MVP-tree we use a simple top-down hierarchical clustering algorithm. For the first level we first select some data objects as vantage points, and the remaining data objects are split evenly into partitions based on their distances to the vantage points. The process is applied recursively until disk-page size cluster are achieved. The

difficulties are how to select vantage points, and how to partition the remaining data objects.

We want to select m vantage points, and for each vantage point, we partition the data set into n equal sized clusters. Therefore, we have altogether n^m sub-clusters. The values of m and n are decided by the disk page size, index key size and node occupancy constraint. To select vantage points, we first run farthest-first-traversal algorithm with a randomly selected first center to select k, k>=m, centers, and then compute the pair-wise distances of all the k centers. We determine center, C, with the maximum distance to a remaining point. Then using C as the first center we run farthest-first-traversal algorithm again to find m centers. These m points are used as the vantage points.

We partition the data sequentially for each vantage point. For the first vantage point, we first compute the distance from all remaining points to it, and then computing n-median we determine intervals that partition the data set into n equal size sets. This is repeated for the second vantages point and so on. This yields a fanout of $n^{m.}$ Since the subsets are of equal size, recursive application of the procedure yields a balanced tree.

4. Experimental results

Our test image database has contains features for 10221 images. Each image has features for the three different metrics detailed above. All of our testing comprises a linear combination of the three metrics, each weighted by 1/3. Our experimental results comprise data to compare the query performance and the scalability of the 3 index structures. We use the M-tree open source release v0.91, which is written in C++ [15]. Our own implementation of radius-based tree (RBT) and multi-vantage point tree (MVP) are written in JAVA. Therefore, we report on implementation independent measurements such as counting the number of distance computations and number of pages accesses rather than actual execution times. For RBT and MVP, since the node sizes are equal to disk page size, we measure the number of nodes visited and present that as the number of disk I/O operations. The same source data and parameters, such as node utilization, dataset size, are used for both the original M-tree release and our Java implementation.

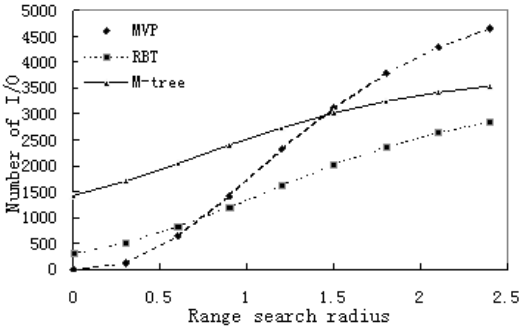

Figure 3 Number of I/Os VS radius

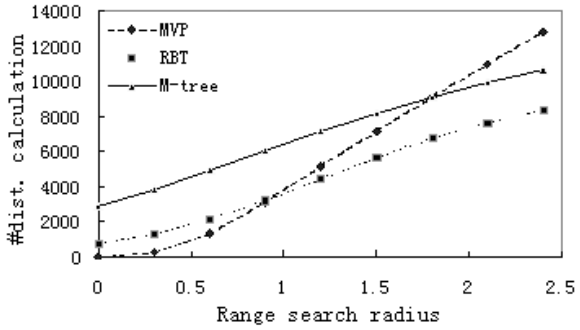

Figure 4 Number of dist. calculations VS radius

We first compare the query performance of M-tree, RBT and MVP. The number of I/O and the number distance calculation are shown in Figure 3 and Figure 4 respectively. Specifically, a number of data objects are randomly selected from the database as query

objects. For each query objects, we run range query and compute the average value of the number of distance calculations.

From the figures we can see that RBT always outperforms M-tree, which means the bi-directional initialization algorithm works well. For small radii, MVP yields the best performance. There is a cross-over such that for larger ranges RBT is the best. The reason is that only a small number of data objects satisfy the query when the radius is small. These data objects distribute in a small number of clusters. Since partitions in vantage point tree do not overlap while those of radius-based tree do, fewer index nodes are accessed in MVP. As the radius increases, due to the curse of dimensionality [6], more and more data objects satisfy the query. The answer set distributes in a larger number of clusters. Ultimately the performance curves of the MVP-trees crossover the curves for the radius-based methods. For large radii the queries overlap too many MVP partitions and pruning becomes impossible. The crossover with M-trees is later than for RBT since the raw performance of RBT is better.

Next, we show the scalability of the index structure. The scalability can be represented by the relationship between query performance and database size. In Figure 5, we show the relationship between number of distance calculation and database size. The data is the average value of range queries with small radii of the 3 index structures. When the radius increases, more data objects, and ultimately all the data objects will be accessed. Therefore, the best scalability that could possibly be obtained is linear for large radii. The corresponding figure for number of I/Os very similar to Figure 5 and is not presented..

Figure 5 reveals that MVP is a very scalable index structure to support proximity query with small radii. Recall, partitions in vantage points tree don't overlap. When the radius is small, only a small number of nodes are to be accessed. Therefore, when searching an internal index node, only a small portion of its children is visited next. Consequently, the scalability is nearly to be logarithmic. From Figure 5, we can also see that RBT also scales, although not as well as MVP. For M-tree, it shows little scalability, and it's weakness for small radii queries magnified as the size of the database is increased.

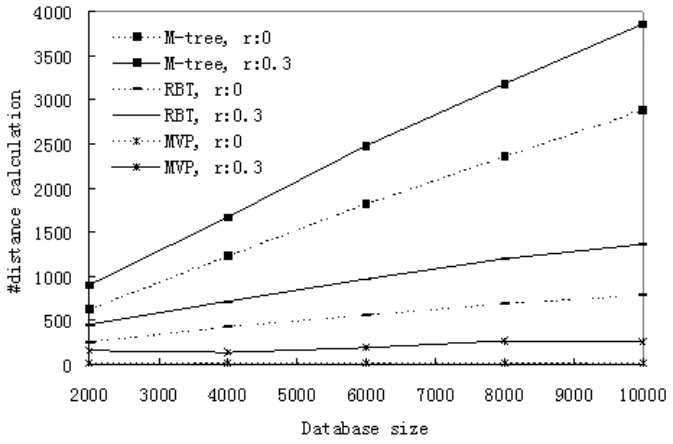

Figure 5 Scalability: average number of distance calculations VS database size of range queries with small radii for 3 index structures

Conclusions and Future Work

In this paper, we explore the application of metric-space index to support proximity query in image databases. Since no domain-related meaning of data is needed, these index techniques can be possibly used in all multimedia databases to support proximity query. Our study reveals a crossover in performance with respect to the radius of the search. One might anticipate for small nearest-neighbor searches, small radii will always be

sufficient and MVP-trees the method of choice, e.g. find the 10 closest matching images. But for range queries, the correct choice becomes domain specific. E.g. find all images that match within tolerance t. For each method of feature extraction and concomitant metric, a value of t that corresponds to meaningful similarity will be different. Developers will have to assess for their particular workload if their value of t is to the left or to the right of the crossover.

In ongoing work we are exploring index structures and clustering algorithms in hopes of finding a single generally applicable solution. There is considerable flexibility in the choice of vantage points and partitioning in MVP-trees. Just as we were able to improve the initial clustering of M-trees and improve performance, we anticipate that MVP-trees will admit similar tuning. Besides absolute performance, we will seek to push the crossover point in the performance to much larger radii queries. Hopefully sufficiently large that MVP-trees may support a broad-class of applications. In addition to radius-based methods and vantage-point methods, taxonomically there is a third family of metric-space index structures, generalized-hyperplane techniques. We are developing overlayable results from this family as well [19,4].

References

[1] Böhm, C., Berchtold, S., Keim, D. A. 2001. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys 33(3):322-373.

[2] Bozkaya, T. and Ozsoyoglu, M.. Distance-based indexing for high dimensional metric spaces. In Proceedings of the 1997 ACM SIGMOD, pages 357--368. ACM, 1997.

[3] Bozkaya, T. and Ozsoyoglu, M.. Indexing Large Metric Spaces for Similarity Search Queries. Association for Computing Machinery Transactions on Database System, pages 1--34, 1999.

[4] Brin, S. Near Neighbor Search in Large Metric Spaces. In Proc. 21st. Int. Conf. Very Large Data Bases (VLDB), pp. 574-584, 1995.

[5] Chavez, E., Navarro, G., Baeza-Yatesand R., Marroquin, J. L. Searching in metric spaces. ACM Computing Surveys 33(3): 273-321, 2001

[6] Chazelle, B.. Computational geometry: a retrospective. In Proc. ACM STOC'94, pages 75--94, 1994.

[7] Ciaccia, P. and Patella, M.. Bulk loading the M-tree. In Proceedings of the 9th Australasian Database Conference (ADC'98), pages 15--26, Perth, Australia, February 1998.

[8] Ciaccia, P., Patella, M. and Zezula, P. M-tree: an efficient access method for similarity search in metric spaces. Proc. 23rd Int. Conf. Very Large Databases (VLDB), 1997.

[9] Gaede, V. and Gunther, O.. Multidimensional Access Methods. ACM Computing Surveys, 1997.

[10] Guttman, A.. R-trees: A Dynamic Index Struc- ture for Spatial Searching. Proc. of SIGMOD, 1984.

[11] Hellerstein, J. M,. Naughton, J. F. and Pfeffer. A., Generalized Search Trees for Database Systems. Proc. 21st Int'l Conf. on Very Large Data Bases, Zürich, September 1995, 562-573.

[12] Hochbaum, D. S. and Shmoys, D. B.. A best possible heuristic for the k-center problem. Mathematics of Operational Research, 10(2):180-184, 1985.

[13] Iqbal, Q. and Aggarwal, J. K.. Perceptual Grouping for Image Retrieval and Classification. 3rd IEEE Computer Society Workshop on Perceptual Organization in Computer Vision, July 8, 2001, Vancouver, Canada, pp. 19.1-19.4.

[14] Iqbal, Q. and Aggarwal, J. K.. Image Retrieval via Isotropic and Anisotropic Mappings. Pattern Recognition Journal. Vol. 35, no. 12, pp. 2673-2686, December 2002.

[15] M-tree project home page. http://www-db.deis.unibo.it/Mtree/index.html

[16] Mao, R., Miranker, D. P., Sarvela, J. N. and Xu, W.. Clustering Sequences in a Metric Space-the MoBIoS project. Poster of the 10th International Conference on Intelligent Systems for Molecular Biology, August 3-7, 2002, Edmonton, Canada.

[17] Mao, R., Xu, W., Singh, N. and Miranker, D. P.. An Assessment of a Metric Space Database Index to Support Sequence Homology. In the proceeding of the 3rd IEEE Symposium on Bioinformatics and Bioengineering, March 10-12, 2003, Washington D.C

[18] Miranker, D. P., Xu, W., and Mao, R.. Architecture and Application of MoBIoS, a Metric-Space DBMS to Support Biological Discovery. To appear as a poster in the 15th International Conference on Scientific and Statistical Database Management.

[19] Uhlmann, J. K.. Satisfying General Proximity/Similarity Queries with Metric Trees. Information Processing Letter, 40(4):175-179, November 25, 1991.