Efficient Search with Changing Similarity Measures on Large Multimedia Datasets

Nataraj Jammalamadaka, Vikram Pudi, and C.V. Jawahar

Center for Visual Information Technology
International Institute of Information Technology
Hyderabad 500032, India
natraj@students.@iiit.ac.in, {vikram,jawahar}@iiit.ac.in

Abstract. In this paper, we consider the problem of finding the k most similar objects given a query object, in large multimedia datasets. We focus on scenarios where the similarity measure itself is not fixed, but is continuously being refined with user feedback. Conventional database techniques for efficient similarity search are not effective in this environment as they take a specific similarity/distance measure as input and build index structures tuned for that measure. Our approach works effectively in this environment as validated by the experimental study where we evaluate it over a wide range of datasets. The experiments show it to be efficient and scalable. In fact, on all our datasets, the response times were within a few seconds, making our approach suitable for interactive applications.

1 Introduction

Information retrieval schemes from multimedia collection often need to compare two multimedia objects and effectively measure their similarity [1,2,3]. A large class of such algorithms represent the multimedia content using an appropriate feature vector and refine the similarity measure using relevance feedback techniques. In this paper, we consider the problem of similarity search when the query object is matched against a large database. In the context of similarity search on large datasets, we specially focus on scenarios where the similarity measure itself is not fixed, but is continuously being refined. User gives direct or indirect feedback regarding whether each retrieved object is indeed similar to the query object or not.

In order to handle large datasets, we employ an index structure that helps narrow down the multimedia objects that actually need to be verified for similarity. The problem of exactly finding the k most similar objects from a large dataset is known to be time consuming. Our algorithm therefore, only attempts to retrieve the k most similar objects, approximately. Our experiments show that most of the objects retrieved are among the true k nearest neighbors.

Existing approaches [4,5,6] for efficient similarity search take a specific similarity measure as input and build index structures tuned for that measure. A few approaches were designed for changing similarity measures [7,8,9,10]. Even they do not perform satisfactorily for multimedia data and associated similarity measures. Realistic multimedia data is characterized by clusters in the data rather than random uniformly distributed data. Associated similarity measures are characterized by a large number of dimensions where a few are actually dominant – *i.e.,* there is high variance in the weights/relevance of dimensions.

In contrast, our approach works effectively in this environment as validated by our experimental study where we evaluate it over a wide range of situations. The experiments show it to be accurate, efficient and scalable. While existing techniques degrade in the presence of clustered data and widely varying relevances of dimensions, our algorithm actually thrives in their presence. On all our datasets, the response times were within a few seconds, making our approach suitable for interactive applications related to multimedia retrieval.

1.1 Problem Statement

In many multimedia object retrieval systems, user presents the system with a query object (say an image or a video clip) and the system retrieves the k most *similar* objects from a database. Multimedia objects are typically represented by a vector of numeric features $X_1, X_2, \ldots, X_D$ which form a multidimensional space. Several approaches exist to measure similarity between multimedia objects [11,12]. Most of these approaches popularly utilize a weighted Euclidean distance to measure the (dis)similarity between points.

Formally, given a point $X = [X_1, X_2, \ldots, X_D]^T$ in a D-dimensional space and a query point $X' = [X'_1, X'_2, \ldots, X'_D]$, the weighted Euclidean distance between X and X' is given by:

$$distance^2(X, X') = \sum_{i=1}^{D} w_i (X_i - X'_i)^2, \qquad (1)$$

where the vector $w_1, w_2, \ldots, w_D$ constitutes the weights along each dimension.

As the multimedia data set grow, scalability is becoming an important issue. When similarity function becomes dynamic, standard data structures become insufficient for the efficient search. We propose a scalable, efficient solution to this problem. We describe our simple and effective solution in Section 2. Performance of the algorithm is comprehensively analyzed in Section 3.

2 Our Approach

The natural approach to design systems for multimedia object retrieval from large databases is to build an *index* for similarity search. Index structures for similarity search is a well-studied field. Unfortunately, most of the available index structures in the literature require a *full* specification of the similarity measure as input. In our context, this is not possible because the similarity measure is continuously being refined by the user during the retrieval session.

Our approach is to build a simple and flexible index structure that can be used for similarity search based on the weighted Euclidean distance measure. It does not require a full specification of the similarity measure as input. Note that the proposed scheme is also applicable to many other distance measures directly or with minimal modifications.

2.1 Index Structure

The index structure is simple: For each of the D dimensions, a list is maintained that contains all the data points sorted along that dimension. In actual implementation, to prevent redundancy, these lists could contain only the pointers or ids of points and the actual points could be stored elsewhere. The lists can be stored on disk and be implemented as $B+$ *trees*.

Insertion and deletion of points from the index structure is simple: it only involves inserting and deleting the projections of those points from the lists of each dimension. These operations can therefore be accomplished in $O(DlogN)$ time for each point, where D is the number of dimensions.

2.2 Retrieval

The retrieval operation is designed to efficiently (but approximately) retrieve the k nearest neighbors of a query point. The pseudo-code of this operation is shown in Figure 1 and is explained below.

```
Retrieve($k, t, X', M$):
  1 $neighbors = \{\}$
  2 for each dimension $d$ (in non-increasing order of weights):
  3     $C$ = $t$ nearest neighbors in dimension $d$
  4     $neighbors$ = $k$ nearest neighbors of $X'$ among ($neighbors \cup C$)
  5 return $neighbors$
```

Fig. 1. Approximate k-Nearest Neighbor based Retrieval

The algorithm takes as input k: the number of desired nearest neighbors, t: the number of candidate neighbors to consider along each dimension, X': the query point and M: the index structure. The output consists of the k nearest neighbors (approximately).

The neighbors of the query point is initialized to the empty set (in line 1 of Figure 1). Next, the dimensions are enumerated in decreasing order of their weights (line 2) and the nearest t neighbors along each dimension d are retrieved (line 3). This is done by searching for the query point in the list for dimension d in the index structure. This search will retrieve the point closest to the query point. Then, a linear traversal along the list from that point in both directions will retrieve the closest t points.

The t points obtained along each dimension are *candidate* points to be considered for being among the k nearest neighbors of X'. These points are compared with the nearest neighbors so far obtained to determine whether they are to be retained in the k nearest neighbor set, or to be discarded (line 4). Finally, the nearest neighbors obtained after enumerating points along all dimensions are output (in line 5).

2.3 Complexity Analysis

Consider N points in the database. Each point is D dimensional. As mentioned earlier, insertion is an offline process and can be done efficiently.

In the search operation Step 3 takes order complexity of $O(log(N) + D)$ and Step 4 takes $O(D)$. Thus the search operation takes $O(D*logN+k*D^2)$. Since $log(N) >> D$ we have the complexity of the search operation to be $O(D * logN)$. For the methods [7,10] order complexity cannot be arrived at and as the weight vectors improve the performance degrades resulting in looking at most of the disk blocks. We experimentally compare the performance in the next section.

2.4 Rationale Behind Design

Our approach has been to build a simple and flexible index structure that can be used for similarity search based on the weighted Euclidean distance measure. Rather than attempting to modify the index structure as the similarity measure is refined during a user-session, we use the same index structure for any combination of weights.

The key operation at the time of retrieval is to obtain the neighbors of the query point along each dimension. This is easily achieved in our approach since the index structure can access the nearest point in $O(logN)$ time and then merely traverse a linked list to enumerate its neighbors. Our approach works effectively because the neighbors along a particular dimension *do not change* when the weights of dimensions are modified.

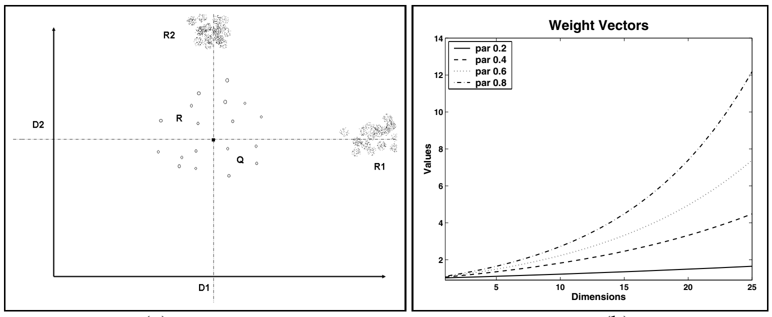
The retrieve operation enumerates dimensions in non-increasing order of their weights. This means that the most important dimensions are enumerated first. This makes it likely that most of the true nearest neighbors are retrieved very early during the execution of the algorithm. This can be advantageous in situations where the user is interested in any neighbors that are within a specified threshold distance from the query point.

Finally, it should be noted that the retrieval algorithm only retrieves *approximately*, the k nearest neighbors. This is due to the fact that there may be nearest neighbors that are not among the nearest neighbors along any dimension. To compensate for this, the algorithm actually retrieves t nearest neighbors along each dimension, where $t \geq k$. Our experiments show that good accuracy is obtained for reasonable values of t.

2.5 Shortcomings

Consider the situation in Fig. 2(a). Here, Q is the query point and the circles marked around it are its true nearest neighbors. Let this region be R. Let R_1 be the region of points which are nearer to the query point along the dimension D_1. And let R_2 be the region of points which are nearer to the query point along the dimension $D2$. When the nearest neighbors to the point Q are desired then the points in regions $R1$ and $R2$ may interfere, because they are the closest points along their corresponding dimensions. Thus the algorithm may fail to retrieve the nearest neighbors. On further analysis we can classify this situation into three cases.

1. Consider the case when region R is a dense region. When the dimension D_1 is considered, along with the points in the region R, points in the region R_1 are also likely to be among the nearest neighbors. When computing the nearest neighbors along the dimension D_2, the points in the region R are preferred over those selected from R_1. Likewise, points from R_2 will also get eventually rejected leaving points only from R. Our algorithm is designed to take care of this.



(a) (b)

Fig. 2. (a) Suspected case of failure (b) Skew of the weight vector is increasing with the value of parameter

2. In the case when region R is sparsely populated, the nearest neighbors along each dimension are not necessarily from region R. Subsequently, many false positives are output by the algorithm. However this situation can be identified by seeing the actual number of intersections during the merge operation. This is then overcome by increasing the t/k ratio.
3. When the query point falls in a region which is extremely sparse, any point retrieved will be irrelevant. If there are no relevant objects in the database, errors in the retrieval process will not affect the overall performance.

3 Performance Study

In this section we evaluate the performance of our approach on both synthetic datasets and real datasets. The synthetic datasets consisted of uniformly distributed, clustered and mixed datasets. The real data set that we employed is from the Corel image collection. All the experiments are performed on a 3 GHz Intel Xeon PC with 4 Gigabytes of main memory, running RedHat Linux 2.6.5-1.

3.1 Accuracy

Since our algorithm only retrieves the k nearest neighbors *approximately*, we have performed detailed experiments to measure its accuracy. We measure accuracy in terms of the number of points retrieved by our algorithm that are actually among the k nearest neighbors. For comparison purposes, the actual k nearest neighbors were found using an exhaustive search algorithm. Accuracy could depend on different factors like the weight distribution and number of points retrieved.

Number of Points Retrieved. Our algorithm takes a parameter t that represents the number of neighboring points to be retrieved. Obviously, accuracy will improve as t is increased. We have therefore experimented with various values of the ratio t/k to determine suitable values for different datasets.

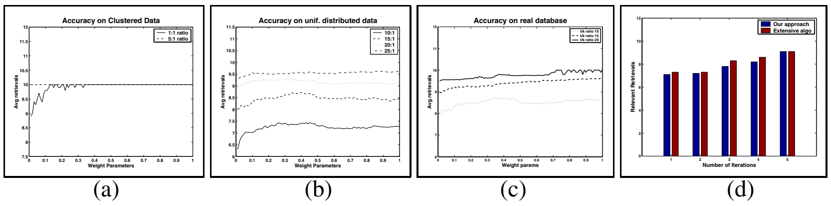


Fig. 3. Performance on various types of data sets. (a) Clustered data (b) Uniform data (c) Real data (Corel dataset) (d) Results of user testing. Our algorithm prefers data sets with concepts. With variances of relevance scores increasing, accuracy increases. Also for most of the weight vectors, our algorithm provides acceptable approximation.

The two curves in the graph of Figure 3 (a) show the accuracy when the values of the t/k ratio is equal to 1 and 5, respectively. The x-axis in this graph represents a parameter that is used to modify the weight vectors (described in Section 3.1) and the y-axis represents the accuracy. The value of k is set to 10. We observed in this graph that as the t/k ratio was increased, the accuracy also increased and reached 100% at 5:1 for all values along the x-axis.

Figure 3 (b) shows the results for the same experiment on the uniform dataset. Here also we observe that with increase in t/k ratio the accuracy goes high. Our algorithm performs better on data sets with one or more clusters (strong concepts). We note that most real-world datasets are likely to contain clusters, rather than being uniformly distributed and hence our approach suits them. This is especially true when the extracted features are relevant to the problem.

Weight Vectors. In order to study the effect of weight vectors on accuracy, we tried various weight vectors for the weighted Euclidean distance metric. Weight vectors denote the importance given to each component (dimension) of the feature vector.

The graph shown in Figure 2(b) shows the weight vectors used. Increase in the values of the component across weight vectors denote the user feedback. A similar plot from the previous experiment given in Figures 3(a) and 3(b) also show the variation of accuracy for different weight vectors. We observe that when the variance of weights is high (i.e, when some dimensions are much more relevant than other dimensions), accuracy is high. The reason for this is that the algorithm takes advantage of dimensions with high weights by enumerating them first. We note that in most multimedia applications, there would be some dimensions that dominate in importance.

Results on Real Database. We have tested the algorithm on the two real world scenarios. In the first scenario we have used a real image database and evaluated our algorithm against it. This experimental results indicate that our algorithm can efficiently perform on real world databases. In the second scenario we subjected our algorithm to user testing. We implemented our algorithm on a standard statistical relevance feedback based image retrieval system.

Figure 3 (c) presents the results of the experiment. We notice that the graph is in accordance the above mentioned results. As the weight vector gets more skewed along the x-axis the accuracy improves. Similarly, as the t/k ratio is increased the accuracy

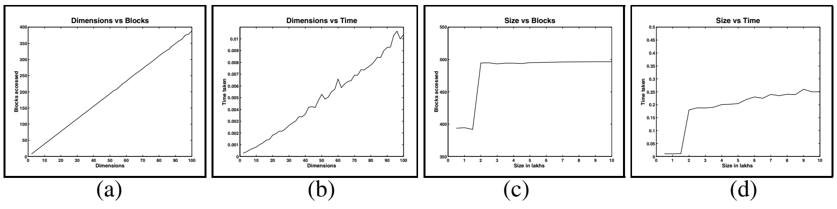


Fig. 4. (a) and (b) suggests that the algorithm has a linear complexity in dimensions and (c) and (d) suggests that the algorithm is logarithmic in size which is desired

improves. Figure 3 (d) gives the total number of relevant retrievals obtained by using our algorithm and the extensive search algorithm as validated by the users . In this experiment the number of desired images are 10 and the result is averaged over 10 users. Note that the accuracy reported in Figure 3 (c) is benchmarked against extensive search procedure. Though the accuracy reported is not 100% Figure 3(c) suggests that the images retrieved by our approach are relevant as validated by the user.

Response Time. We evaluated the response time of our algorithm for different datasets and parameters. We see that flat-files perform better than SQL and R-trees because the overhead of system calls in order to scan all the points is smaller for flat files. For the same reason, SQL performs better than R-trees. Our algorithm performs significantly better than any of these approaches because it does not need to scan through all points of the database – its index can help narrow down the search to only a few points. It may be argued that the R-tree index also could be used to narrow down the search for the k nearest neighbors. Unfortunately, this is not possible because the weight vector indicated in the user-session may be different from what was used in the R-tree construction.

3.2 Scalability

We studied the scalability of the approach in terms of the variation in response time and number of disk block accesses with respect to the database size (number of points) and the dimensionality. These studies were made on the clustered dataset.

Number of Dimensions. Figure 4(a) shows the effect of increasing number of dimensions on the response time of the algorithm. It is seen that the relation is linear. This is expected as per the complexity analysis done in Section 2.3, when $N >> D$. Figure 4(b) shows the effect of increasing the number of dimensions on the number of disk block accesses made by the algorithm. It is again seen that the relation is linear, further explaining the linear nature of the response times. In these graphs, each point is the average over all the 10 weight vectors used in earlier experiments and also over 10 different queries.

Database Size. Figure 4(c) shows the effect of increasing number of database points on the response time of the algorithm. It was seen that the relation is logarithmic. This is expected as per the complexity analysis done in Section 2.3. Figure 4(d) shows the effect of increasing number of database points on the number of disk block accesses made by the algorithm. It was again seen that the relation is logarithmic, further explaining the nature of the response times. Again, note that in these graphs, each point is the average over all the 10 weight vectors used in earlier experiments and also over 10 different queries.

The relationship between the number of blocks accessed and the dimension is given by $Blocks = (Height) * (D) + D * \sum_{i=1}^{D}(LNS(i))$ where Height is the present height of the tree, and $LNS(i)$ is the total leaf nodes accessed in the dimension i. This relationship explains the sudden transition in the Figure 4(d). We have used B+ trees to maintain all the points along each dimensions. Increase in the height of the tree as a result of increase in the size is responsible for the sudden transition. However, it is important to note that after the transition, the number of blocks accessed remained constant in-spite of huge increase in the size of the database (from 0.2 million to 1 million), thus indicating that the approach is efficiently handling *large* databases.

3.3 Comparison with Existing Approaches

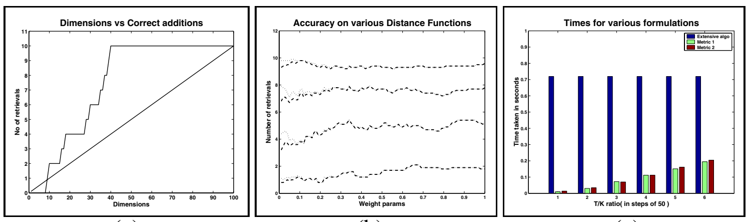
Table 1 shows the blocks accessed of our algorithm as compared against the approach in Kurniawati *et al.* [10]. In this table, we show for datasets with varying number of dimensions, the following statistics: (1) LFO: the leaf fan out (2) IFO: inner node fan out (3) ltouch: the number of leaf nodes of the index structures that have been visited by the two approaches, (4) lused: the number of leaf nodes that actually contained some of the k nearest neighbors, and (5) the number of internal nodes accessed (inode). On observing the total number of block accesses($ltouch + lused + inode$) it is clear that our approach is almost consistently better than the approach in Kurniawati *et al.* [10]. In fact, it performs an order of magnitude better for high dimensions, which is often the case in multimedia datasets.

3.4 Heuristics for Performance Enhancement

Different distance functions. We have experimented with two variants of the distance functions with various weight vectors and t/k ratios to choose a better performing distance function both in-terms of accuracy and speed.

The first one is given by the $distance = \sum_{i=1}^{D} weight(i) * (query(i) - db(i))^2$ where D is the number of dimensions of the data. The second is $distance = \sum_{i=1}^{d}(weight(i) * (query(i) - db(i))^2)$ where d is the dimension that the algorithm is looking at. We observed that the second formulation is doing as good as the first function Figure 5(b) in terms of accuracy but in terms of time Figure 5(c) , it outperforms the traditional distance measures.

Stop criteria. Many of the applications have an acceptable accuracy and provides a scope for improvement in speed. At the end of each iteration we have a total number of t candidates(k being desired). Of these few of them(lets say $t1$) will match with result of exhaustive search algorithm. We call these $t1$ entries as correct additions. Observing the correct additions in Figure 5(a), we find that initially there are large number of correct additions, and later this rate decreases. We can safely conclude that at any given time



(a) (b) (c)

Fig. 5. (a) suggests a possible scenario of how accuracy increases with dimensions. (b) and (c) compare the accuracy and time taken for different distance metrics.

Table 1. Comparison of our approach with an existing one. Ours is an order times better than this, particularly for high dimensions.

	Our Approach					Kurniawati et al				
Dimension	LFO	IFO	ltouch	lused	inode	LFO	IFO	ltouch	lused	inode
2	682	1024	2.1	2.1	4	682	225	1.4	1.4	1
4	409	1024	4.3	4.3	8	409	146	4.5	2.7	2.2
8	227	1024	8.5	8.5	16	227	78	89.5	8.9	4.8
16	120	1024	18.5	18.5	32	120	40	740.6	25.9	20.4
32	62	1024	40.9	40.9	64	62	20	1613.0	41.0	87.0

s ($< T$), the accuracy of the algorithm is approximately $\frac{s}{T}\%$ of the total accuracy of the algorithm. Thus the above observation could be used as stopping criteria, given the acceptable accuracy.

Comments on Implementation. The optimal t/k for a given session may depend on the query scenario. This could be done by estimating the t/k ratio dynamically. The dynamic estimation could be done based on the following observations: (a) With the increase in dimensions the rate of correct additions is going down. (b)With each dimension, the distance(dis) of the ith nearest neighbor decreases. (c) As the t/k ratio increases, the distance of the ith nearest neighbor is decreasing. (d) The amount at which such distance would fall decreases with the t/k ratio.

4 Related Work

Content-based multimedia retrieval has been an active area of research [1,2,3]. Scalability of these approaches for large datasets of images has not received the due attention. A large body of work exists on the study of index structures for similarity search. These algorithms involve building a spatial access tree, such as an R-tree [5], k-d tree [4], SS-tree [6] or their variants. The index structures presented in these papers were novel, elegant, and useful. However they are not applicable in our study as they take a specific similarity measure as input and build index structures tuned for that measure. Recent attempts [7,8,9,10] on this problem focused on scenarios where the similarity measure itself is not fixed, but continuously being refined. These algorithms have taken a branch and bound approach, which may degrade to searching most of the tree structure. In this work, we focus on efficiently retrieving the data, with bounds on the time taken, when similarity measure is varying continuously.

5 Conclusion

In this paper, we addressed the problem of finding the k most similar objects in large datasets given a query object, when similarity measure is continuously being refined with user feedback. Our approach builds a simple index structure that can short-list the points to be considered for nearest neighbor search effectively even without a complete specification of the similarity measure. Experimental study over a wide range of datasets showed our approach to be accurate, efficient and scalable. In fact, on all our datasets, the response times were well within a few seconds, making our approach suitable for interactive applications.

References

1. Huang, T., Zhou, X.: Image retrieval with relevance feedback: From heuristic weight adjustment to optimal learning methods. In: International Conference on Image Processing. (2001) III: 2–5
2. Rui, Y., Thomas, S.H., Chang, S.F.: Image retrieval: Past, present, and future. In: International Symposium on Multimedia Information Processing. (1997)
3. Yang, J., Li, Q., Zhuang, Y.: Towards data-adaptive and user-adaptive image retrieval by peer indexing. International Journal of Computer Vision **56** (2004) 47–63
4. Bentley, J.L.: Multidimensional binary search trees used for associative searcing. Communications of the ACM **18** (1975) 509–517
5. Guttman, A.: R-trees: A dynamic index structure for spatial searching. ACM SIGMOD International Conference on Management of Data (1984) 47–57
6. White, D.A., Jain, R.: Similarity indexing with ss-tree. In Proc 12th International conference on Data Engineering, New Orleans, Louisiana (1996)
7. Faloutsos, C.: Searching multimedia databases by content. Advances in Database Systems. Kluwer Academic Publishers, Boston (1996)
8. Faloutsos, C., Equitz, W., Flickner, M., Niblack, W., Petkovic, D., Barber, R.: Efficient and effective querying in image content. J of Intelligent Information Systems (1994a) 231–262
9. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. ACM SIGMOD International Conference on Management of Data (1994b) 419–429
10. Kurniawati, R., Jin, J., Shepherd, J.A.: Efficient nearest-neighbor searches using weighted euclidean metrics. Technical report, Information Engineering Department, School of Computer science and Engineering, University of New South Wales (1998)
11. Rui, Y., Huang, T.S., Mehrotra, S.: Relevance feedback techniques in interactive content-based image retrieval. IS&T and SPIE Storage and Retrieval of Image and Video Databases VI, San Jose, CA, USA, Jan. (1998)
12. Tian, Q., Hong, P., Huang, T.: Update relevant image weights for content-based image retrieval using support vector machines. IEEE Inter. Conf. on Multimedia & Expo **18** (2000) 1199–1202