Approximate Nearest Neighbor Searching in Multimedia Databases

Hakan Ferhatosmanoglu
Department of Computer Science

Ertem Tuncel
Department of Electrical and Computer Engineering

Divyakant Agrawal
Department of Computer Science

Amr El Abbadi
Department of Computer Science

University of California
Santa Barbara, CA 93106
{hakan,agrawal,amr}@cs.ucsb.edu, ertem@ece.ucsb.edu

Abstract

In this paper, we develop a general framework for approximate nearest neighbor queries. We categorize the current approaches for nearest neighbor query processing based on either their ability to reduce the data set that needs to be examined, or their ability to reduce the representation size of each data object. We first propose modifications to well-known techniques to support the progressive processing of approximate nearest neighbor queries. A user may therefore stop the retrieval process once enough information has been returned. We then develop a new technique based on clustering that merges the benefits of the two general classes of approaches. Our cluster-based approach allows a user to progressively explore the approximate results with increasing accuracy. We propose a new metric for evaluation of approximate nearest neighbor searching techniques. Using both the proposed and the traditional metrics, we analyze and compare several techniques with a detailed performance evaluation. We demonstrate the feasibility and efficiency of approximate nearest neighbor searching. We perform experiments on several real data sets and establish the superiority of the proposed cluster-based technique over the existing techniques for approximate nearest neighbor searching.

1 Introduction

With the rapid deployment of different types of information in large-scale applications, both the dimensionality and the amount of data that needs to be processed are increasing rapidly. The general approach in most modern applications is to generate feature vectors that represent the original data objects. A similarity query is defined as finding the most similar data objects in the data set to a given query object. For example, in image databases a possible similarity query is to find the images most similar to a given image. The images are represented as d dimensional feature vectors and the similarity between the images is defined by a distance function, e.g., Euclidean distance, between the corresponding feature vectors. The k-nearest neighbor, $k-$NN, problem is defined as finding the k most similar feature vectors to a query point q. A closely related query is the ϵ-range query where all feature vectors that are within ϵ neighborhood of the query point q are retrieved.

In typical applications, the amount of data is very large. Commercial data warehouses are doubling their sizes every 9-12 months and satellite data repositories will soon add one to two terabytes of data each day [1]. If current trends continue, large organizations will have petabytes of storage [9]. Traditional techniques until recently focused on getting exact results for queries, where exactness is defined in terms of the feature vectors and a distance function between them. However, exact results are very difficult to obtain in typical multimedia applications and leads to significant inefficiencies in the system. Besides, in these applications, the meaning of 'exact' is highly subjective. Because of the nature of multimedia applications it is usually not very meaningful to pursue exact answers in such applications.

In several modern applications, e.g., internet, both the queries and the information in the database is either imprecise or not defined with 100% accuracy. It is typically the case that the data itself is an approximate representation of real world entities. With the usage of the internet, information may be obtained from several incomplete and inaccurate data resources and searching over these resources may give useful but sometimes imprecise information. The generation of feature vectors from the original objects may itself be based on different heuristics. Besides, the semantics of queries are also not as strict as the exact queries used in relational databases. For example, a query asking the closest 5 Italian restaurant to a moving car may easily miss the 3rd closest and include the 6th closest, and still satisfy the driver. The definition of 'exact' is subjective and depends on the way the feature vectors are created and the distance function defined between the feature vectors. For example, the QBIC project at IBM provides the ability to run queries based on colors, shapes, and sketches [35, 18]. Similarly, the Alexandria Project at UC Santa Barbara provides similarity queries for texture data [34]. It is not possible to come up with a feature vector extraction method and a distance criterion which is universally accepted. This is not the case even with human perception, i.e., the definition of the similarity may differ depending on the viewers' expectations. One user may choose the third best result over the first or the second one, based on his/her similarity expectation. In several cases, close approximations may be good enough for human perception. Obviously, this does not mean that the feature vectors are useless, it tells us that we need to be careful when using feature vectors. They model the real data, but not always with 100% accuracy.

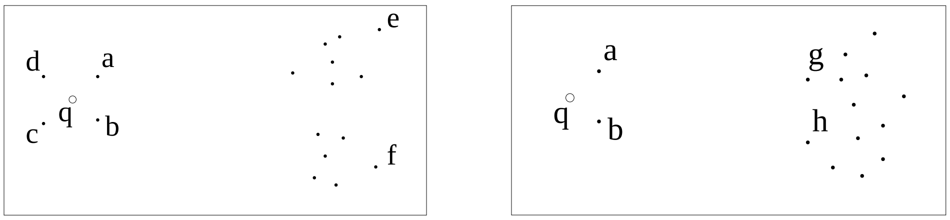
In this paper, we develop a general framework for approximate nearest neighbor queries. We first categorize the current approaches based on either their ability to reduce the data set that needs to be examined, or their ability to reduce the representation size of each data object. In many high dimensional search applications, a user can be satisfied with approximate, or incomplete results (a particular query is leading to an uninteresting data set). We therefore propose modifications to well-known techniques to support the progressive processing of approximate nearest neighbor queries. A user may therefore stop the retrieval process once enough information has been returned. This can be quite beneficial since obtaining exact results may take a long time, and in fact may be unnecessary. We then develop a new technique based on clustering that merges the benefits of the two general classes of approaches. Our cluster-based approach allows a user to progressively retrieve the approximate results with increasing accuracy.

In the next section, we discuss the main metrics for evaluating approximate nearest neighbor queries. Section 3 discusses the categorization of the current approaches. In Section 4, we develop simple progressive approximation techniques. In Section 5, we propose a cluster-based integrated approach. Section 6 includes a detailed performance analysis that demonstrates the advantages of the proposed cluster-based integrated technique. Section 7 concludes the paper with a discussion.

2 Approximation Quality Metrics

For similarity queries, the quality of the result set is traditionally measured by a combination of two important quality metrics: *recall* and *precision* [40]. They can be described as completeness of retrieval and purity of retrieval, respectively. Recall is a measure of how well the retrieval finds all relevant objects even to the extent that it includes some irrelevant objects. Precision is a measure of how well such a system finds only relevant objects even to the extent that it skips some relevant objects. The irrelevant objects in the result set are called *false hits* and the relevant objects that are not in the result set are *false dismissals*. A traditional approach which uses these two metrics is dimensionality reduction on multi-dimensional data. In general, high dimensional data is first reduced to lower dimensions and the search is conducted in the lower dimensional space [2]. Most of the dimensionality reduction techniques focus on allowing some false hits but no false dismissals.

For approximate k-nearest neighbor searching, the number of false hits and the number of false dismissals become the same value. Since the result set size is always k, if the approximation causes some false dismissals, these dismissals are replaced by false hits. Computing the number of false hits, or dismissals, is enough to capture the traditional error metric, which we will refer to as F. We use the metric F as one of our metrics in the evaluation of the proposed techniques.



(a) Example 1

(b) Example 2

Figure 1: 2-NN query on q

However, the number of false hits and dismissals does not capture important information about the quality of the approximations. In Figure 1(a), the two nearest neighbors of the query point q are asked.

The points a and b are the two closest points according to the Euclidean distance. However, the points c and d are also very close to the query point and the user may potentially be interested or satisfied with them being the answer. On the other hand, the points e and f are far away from q and there is much less chance that the user is interested in those points. Suppose there are two approximation techniques, one returns (c, d) and the other returns (e, f) as the result of 2-nearest neighbor query on q. If the traditional error metric of the number of false hits is used, both techniques have the same number of false hits, i.e., 2, which is the worst possible error. Therefore, with this metric both techniques have the same error and it is not possible to differentiate the qualities of these two different answer sets.

The points c and d in Figure 1(a) are not only the third and the fourth closest points to the query, they are also spatially close enough to be of interest to the user. It is also important to note that the rankings of the points in terms of closeness to the query does not capture any information regarding the quality of approximation. For example, points g and h in Figure 1(b) are not of that much interest even though in this case, they are also the third and the fourth closest points to q. The quality/error metric should give a higher error to (g, h) in Figure 1(b) than it gives to (c, d) in Figure 1(a). Especially when dealing with approximate similarity-based retrieval, it is important to develop a new metric which also takes into account the quality of the answers with respect to closeness to the query object.

We now introduce an alternative error metric which is particularly useful for approximate k-NN searching. Suppose the approximate $k-$NN algorithm returns the result set $(a_1, a_2, \ldots, a_k)$ and the real result set computed by the underlying distance function d_f is $(r_1, r_2, \ldots, r_k)$. We define the error metric D as the relative approximation error, given by $\frac{\sum_{i=1}^{k} d_f(q,a_i)}{\sum_{i=1}^{k} d_f(q,r_i)}$. For instance, for squared Euclidean distance D is defined as follows:

$$D = \frac{\sum_{i=1}^{k} \|q - a_i\|^2}{\sum_{i=1}^{k} \|q - r_i\|^2}.$$

3 Approaches for Approximate Nearest Neighbor Queries

In this section, we classify the general approaches that can be used to solve the approximate searching problem. We then develop various techniques as examples of these general classes.

Consider a data set with n data objects each represented by a d dimensional feature vector. In typical applications, both the dimensionality d and the number of data objects n are large. Problems such as the curse of dimensionality in high dimensions and scalability problems for very large data sets can partially be offset by structures that support approximate searching. An approximate searching technique should maximize the accuracy and minimize the cost of processing queries, which are usually dominated by the number of I/O operations. Thus, an effective approximate searching technique should organize the data such that acceptable accuracy is achieved and at the same time the number of pages retrieved as a result of a query is minimized.

Naively, a nearest neighbor query could be answered by examining the *entire* representative of *every* data object. Two general classes of possible approaches for the approximate searching problem naturally arise:

- Retrieved set reduction: the data is organized such that only a small subset of the data objects are examined to answer a nearest neighbor query.
- Representative size reduction: the representatives of the data set are organized such that only a partial representation (e.g., 2 out of d dimensions) of each object is examined.

The *retrieved set reduction* approach is important especially for scalability in large data sets. An approximate NN search technique which retrieves the feature vectors related to a subset of the data set clearly outperforms a technique that retrieves the feature vectors for the entire data set. Similarly, the *representative size reduction* approach is crucial for high dimensional data. Considering all the dimensions at the same time dramatically degrades the efficiency of high dimensional query processing. We briefly discuss some possible techniques that are based on these two basic ideas.

3.1 Retrieved Set Reduction

Instead of accessing/retrieving the entire data set, the search technique focuses on a portion of the data set, ideally on the most relevant data to the given query. One effective and well-known solution is to index the multi-dimensional data. The goal of all these index structures is to reduce the size of the data set which is retrieved as a result of a query. Instead of n objects, the query retrieves a set of s objects, which is a subset of these n objects. The better the index structure, the smaller the size of s. There have been several approaches to organize and partition a multi-dimensional data set for indexing including kdb-trees [37], hB-tree [31], R-tree [23], R*-tree [5], SS-tree [43], TV-tree [28], X-tree [8], Pyramid Technique [7], Hybrid Tree [12]. There are also techniques that are proposed to reduce the size of the retrieved set in multiple disk architectures [16, 19, 20]. All these index structures focus mainly on finding the exact result for a query. Therefore, a retrieved set of size s includes the query result set and some irrelevant objects. The optimal set of objects to be retrieved is the result set, i.e., no objects are retrieved if they are not in the result set. A branch-and-bound technique for $k-$NN queries on index tree structures, such as R-tree, is proposed in [38]. The tree, which consists of minimum bounding rectangles (MBR), is traversed and some of the MBRs are pruned if they are guaranteed not to have the closest data point(s). For example, if the shortest possible distance between the query point and any point in an MBR is larger than the current computed NN distance, then there is no need to check the data points within that MBR. The amount of data retrieved as the result of a query is reduced by pruning some of the MBRs using proximity comparisons of points within MBRs.

In the context of approximate searching mechanism a natural question arises: Is it possible to achieve faster response time if we allow some false dismissals? Recently, various approaches in different domains have developed effective algorithms for approximate searching [4, 3, 22, 14, 41, 44, 17]. Most of these techniques specifically focus on ϵ-nearest neighbor queries. The ϵ-NN is defined as finding a neighbor of the query point within a factor of $(1+\epsilon)$ of the distance to the true nearest neighbor. In [14], an algorithm was proposed in which the error bound ϵ can be exceeded with a certain probability δ using some prior information on the distance distribution of the query point. This technique can be classified as a retrieved set reduction approach, which reduces the number of retrieved data by pruning a larger number of index nodes. The retrieved data set size is first reduced by the underlying index structure. Besides the irrelevant objects pruned by the index, more objects are pruned by shrinking the NN query sphere at the expense of allowing some false dismissals. The technique also adds a second level of approximation, where more objects can be pruned by allowing to exceed ϵ with the probability δ. In [22], a locality sensitive hashing structure is created by a randomized procedure and the $(1+\epsilon)$-approximate NN point is found with a constant probability. Hashing is used to reduce the retrieved set size, therefore improving query time, again at the expense of false dismissals. This approach is also based on the retrieved set reduction idea, where hashing is used instead of a multi-dimensional index tree to identify the retrieved data set. The disadvantage of this approach, however, is that ϵ needs to be known in advance, and some preprocessing, which is exponential in $1/\epsilon$, is needed.

3.2 Representative Size Reduction

Multi-dimensional index trees are very effective in low dimensions. Therefore, they are widely used in low dimensional applications such as Geographical Information Systems (GIS) [13]. However, there are several applications, e.g., multimedia databases [18, 39], that need high dimensional support. As the dimensionality of the feature vectors increases, query performance of the multi-dimensional index tree structures degrades significantly [8, 6, 42, 10]. Therefore, considering all the dimensions at the same time dramatically degrades the efficiency of high dimensional query processing. Working on the reduced feature-vector set is an effective approach to this problem.

A typical example for representative size reduction is the dimensionality reduction approach. The most common approaches found in the literature for dimensionality reduction are linear-algebraic methods such as the Karhunen-Loeve Transformation (KLT) [27, 30], or applications of mathematical transforms such as the Discrete Fourier Transform (DFT) [36], Discrete Cosine Transform (DCT) [25], or Wavelet Transform (DWT) [11]. As the transformations are known to be distance preserving, the general approach is to transform the high dimensional feature vectors and lower dimensional vectors are created by taking the first few leading coefficients of the transformed vectors [2]. The general idea of these techniques depends on the observation that by using these transformations, a small subset of dimensions keeps a high portion of the information about the feature vectors. For dynamic data sets, approximate KLT has been shown to be an effective technique compared to the techniques based on exact transformations [26].

VA-files [42] are an effective representative size reduction approach for high dimensional nearest neighbor searching. The VA-file is basically a sequential list of approximations of feature vectors, based on quantization of the original feature vectors. Therefore, the total size of the feature-vector set is reduced by a significant amount. Exact nearest neighbor searching in a VA-file has two major steps. In the first step, the set of all vector approximations is scanned sequentially and lower and upper bounds on the distance of each vector to the query vector are computed. The vectors with the smallest bounds are determined to be the candidates for the nearest neighbor of the query. Subsequently, in the second step the real feature vectors of the candidates are visited to determine the actual nearest neighbor. Recently, Weber and Bohm [41] have stated that the performance of previous approaches for approximate NN-search, with reasonable approximation errors, is close to linear. They propose an approximate $k-$NN searching technique based on VA-files. To overcome the I/O bottleneck which is important for large databases, they proposed a VA-file based technique which omits the second step of the exact NN search algorithm. The similarity distances are estimated from the lower and upper bounds in the first step, and the result set is created using the results in the first step. By allowing some errors in the result set, approximate NN searching in VA-file achieves an order of a magnitude speedups over exact NN search in VA-file. Recently, we proposed extensions to the original VA-file to handle non-uniform and clustered data sets. In this approach, the creation of a VA-file is improved by first transforming the data using KLT into a more suitable domain. Available bits are non-uniformly allocated to the different dimensions and a quantizater is used that makes use of the data statistics. These steps result in improved performance, especially for non-uniform data sets. We will refer to this technique as VA$^+$-file and a detailed analysis of this technique can be found in [21].

4 Simple Progressive Approximation Approaches

In this section, we develop two simple approximate k-NN techniques that are instances of retrieved set reduction and representative size reduction approaches. These approaches modify the well-known techniques and adapt them to support the progressive retrieval of approximate information based on NN queries.

4.1 Sequential Scan with Retrieved Set Reduction

An obvious and simple way of implementing the retrieved set reduction is to sequentially scan a portion of the data set. The basic idea is to access only a portion of the data set and answer the query based on the portion that is read. Since sequential scan is known to be an effective alternative to several indexing techniques for high dimensional data sets [6, 42, 10], it is natural to start the discussion of approximating $k-$NN queries with sequential scan. This simple idea can be summarized as follows. The data set is stored in the storage without any particular index structure. When a query is issued, the data is scanned sequentially starting from the first page. The data pages are read in the order they are stored and k nearest neighbors can be computed within the portion of the data set that is read so far. The search can be interactive, and it can be stopped any time, and the answer is based on the portion of the retrieved data set.

4.2 Representative Size Reduction: Sub-vectors based on KLT

We now illustrate how distance preserving transformation and dimensionality reduction techniques can be adapted for progressive approximate k-nearest neighbor searching. Distance preserving transformations, e.g., KLT, are especially suitable for interactive approximate searching. However, instead of retrieving each of the feature vectors at once with all its dimensions, a better strategy is to split the feature vectors into blocks and perform multiple passes on each feature vector by retrieving the blocks in the order of importance. This helps to gather more useful information, i.e., the important dimensions of each feature vector, in earlier steps and gather less important information later in the search, if necessary. Instead of waiting for a whole pass on all feature vectors, the system can provide the user an accurate approximate result that is obtained from the first set of dimensions which have a high amount of energy.

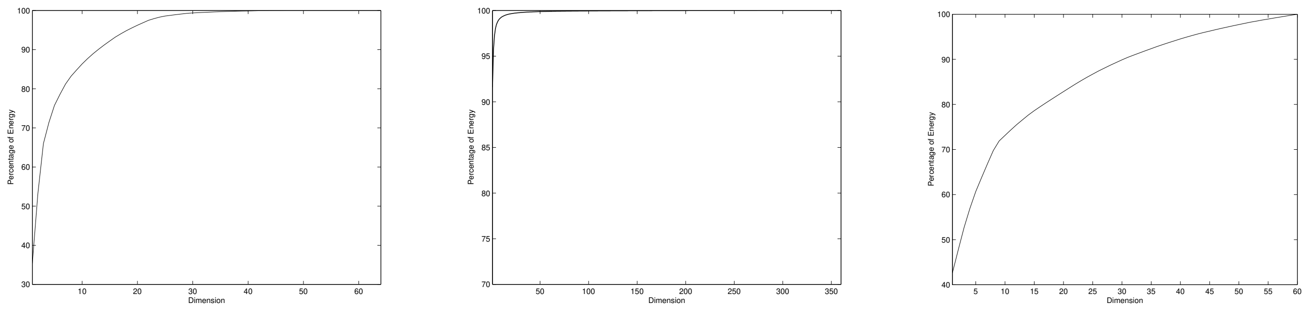
Our approach for feature-vector set organization based on distance preserving transformations works as follows. The d dimensional feature vectors, where d is usually high in typical applications, are transformed using the Karhunen-Loeve transformation (KLT) and a new set of d dimensional vectors is created. This set is used as the new set of feature vectors. We use KLT, because it is known to have the maximum energy compaction property for any given data set. That means, among all the transform-based dimensionality reduction techniques, KLT is the best in accumulating the data energy into a fixed number of dimensions. Each feature vector is divided into s fixed sub-vectors, each with r_i number of dimensions, where $\sum_{i=1}^{s} r_i = d$. The corresponding sub-vectors of all feature vectors are stored consecutively and therefore the sub-vectors of each feature vector are stored separately. Without loss of generality let $r_i = 1$, for $0 \leq i \leq d$, and therefore $s = d$ for simplicity. In this case, the values of first dimension of each vector are stored together consecutively on secondary storage, and second dimension values are stored together, and so on.

When a query is issued, in the first step, only the pages that contain the first dimensions of all vectors are read from storage and the first dimensions are considered for similarity. Then, the other dimensions are read and the similarity computation is made based on all dimensions that have been read. Partial results are stored in memory to be used for the consecutive steps. If the user stops the search anytime, the query is answered based on the results that are found up to that point. Later in the paper, we will evaluate the performance of this simple approach and compare it with the other approaches.

5 An Integrated Approach for Approximate Nearest Neighbor Searching

Both retrieved data set and representative size reductions have strong motivations and need to be considered. Therefore, it is important to develop techniques that combine the advantages of both retrieved set and representative size reductions. In this section, we propose a new technique that effectively combines the two class of approaches into a single framework, i.e., reducing the size of the retrieved set and the feature vectors for efficient approximate searching. The retrieved portion of data is reduced by the help of a clustering technique, which is an adaptation of K-means clustering [32, 15], and the feature vectors within a cluster are organized to support interactive approximate searching.

First, a dimensionality reduction, from d to r, is performed on each data point in the data set. The dimensionality reduction is done as described earlier, i.e., by taking the first r dimensions in the KLT domain. The value of r can be determined based on a statistical analysis of the data. For all dimensions i, the average energy that is stored in the first i dimensions is computed. Let r be the minimum number of dimensions that keeps energy that is above a certain threshold. Analysis of the energy stored in KLT-domain dimensions can be used for high dimensional data characterization. The r value that stores an amount of energy that is greater than a certain threshold, say 85%, is a simple characterization. We refer to the r dimensions which store more energy than the threshold, the dominant KLT-dimensionality of the data. In our analysis, we establish that the first few dimensions, usually 5 to 10 dimensions, in the KLT domain store a high amount of the energy. Figure 2 illustrates the percentage of cumulative energy accumulated into reduced number of dimensions for three different real data sets, which we will describe and use later in the performance evaluation, i.e., image color histogram, stock time-series, and satellite image textures data sets.



(a) Color Histogram (b) Stock Market Time-Series (c) Satellite Image Texture

Figure 2: Percentage of cumulative energy accumulated into reduced number of dimensions

After dimensionality reduction, a modified K-means clustering algorithm is used in the low dimensional domain. The original K-means algorithm [32, 15] iteratively constructs a number of clusters with a representative for each cluster such that the error in representation is minimized. The details of the algorithm is in Figure 3.

Start with a given set of cluster centers (or centroids) c_i for $i = 1, \ldots, K$. Set $\Delta = \infty$, and fix $\epsilon > 0$. Denote by $f(n)$ the cluster to which the data point t_n is assigned.

1. For all n, assign data point t_n to cluster i (i.e., set $f(n) = i$) if

$$\|t_n - c_i\|^2 \leq \|t_n - c_j\|^2 \quad \text{for all } j = 1, \ldots, K.$$

2. For $i = 1, \ldots, K$, compute the new centroid c_i as the center of mass of all data points assigned to cluster i, i.e.,

$$c_i = \frac{1}{N_i} \sum_{n:f(n)=i} t_n \ ,$$

   where N_i is the total number of data points assigned to cluster i.

3. Compute the total within-cluster scatter as

$$\Delta' = \sum_n \|t_n - c_{f(n)}\|^2 \ .$$

   If $\frac{\Delta - \Delta'}{\Delta} < \epsilon$, then STOP. Otherwise set $\Delta = \Delta'$, and go to step 1.

Figure 3: K-means algorithm

By close inspection, it can be observed that the total within-cluster scatter monotonically decreases, and hence the algorithm always converges. This K-means algorithm has found applications in many different disciplines: unsupervised learning in pattern recognition [15], unsupervised image segmentation [24], and vector quantizer design in signal compression [29].

The modification we introduce in the above algorithm is that we limit the size of each cluster from both above and below. If the size goes above the upper threshold, the cluster is split into two. If the size goes down below the lower threshold, then the cluster centroid is erased from the list of centroids. To split a cluster, we first duplicate the cluster centroid, and then perturb the exact copies randomly. It is known that K-means algorithm is sensitive to initialization. Since we have this splitting mechanism, instead of starting from cluster centroids chosen by some pre-processing scheme, we start by a single cluster, and the algorithm automatically creates new clusters until the population of each cluster is below the threshold. As we will demonstrate later, by having a lower threshold for cluster size, several queries can be answered by retrieving only a very small number of clusters. Also, by limiting the cluster sizes from above, we avoid extremely unbalanced distribution of data over the clusters. Although the minimum and maximum cluster sizes are not dominant factors in the performance of our technique, reasonable values need to be set for the design purposes. For k-nearest neighbor queries, k is expected to be less than a given number, say 500, because of the expectations and readability of a query result. In typical applications, k may be taken to be between 1 and 100. Practically, minimum cluster size can have a value comparable to the expected values of k in the application, e.g., 10, 50, etc. The maximum cluster size can be picked as a function of the minimum cluster size, e.g., 10 to 20 times the minimum cluster size. One can also pick an initially huge maximum cluster size, and decrease it gradually "on the fly", i.e., while the algorithm is executed, so as to keep the number of clusters in reasonable bounds. We adopted the latter approach throughout this work.

The proposed integrated algorithm for approximate k-nearest neighbor searching works as follows: Given a query, transform it using KLT and use the r most significant dimensions, and find which cluster the point falls into. Then read from disk the first r-KLT-dimensions of all the points that fall into that cluster. Sort these points in r-dimensional domain based on their distance from the query point. The first k points after the sorting would be the first level approximation of the answer set. There are two directions for progressive refinement of the query here. In the first direction, more dimensions of the points in the cluster could be read from disk. This would certainly improve the distance calculations, and hence the accuracy. The second orthogonal direction is to read neighboring points that fall in other clusters. The best clusters to read are heuristically those whose centroids are closest to the query point in the r-dimensional design domain. The closest one is already read in the base layer. Reading the next best cluster also improves the accuracy, since now there are more candidate points to choose from.

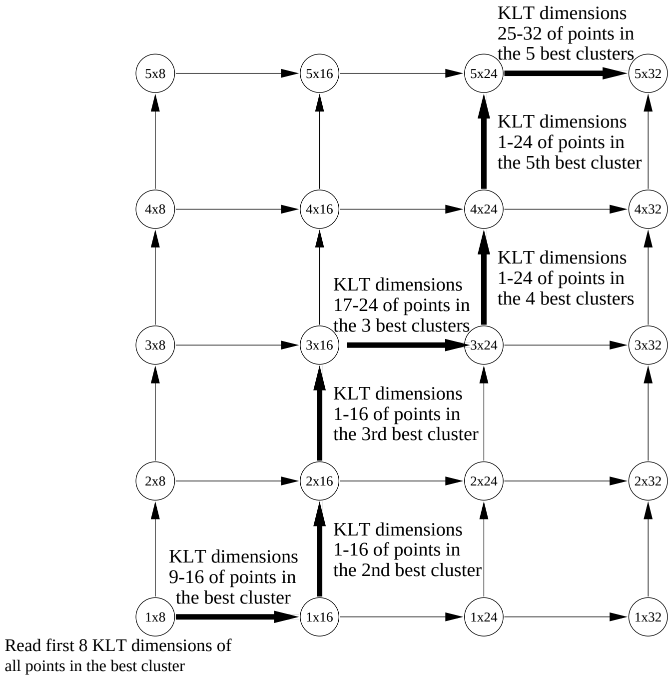


Figure 4: A path chosen in the progressive refinement domain

One can follow any path in this progressive refinement framework. Figure 4 illustrates an example of steps of the algorithm for a high dimensional data set, where r is taken to be 8. The 5×4 progressive refinement structure can be summarized as follows: We read 8, 16, 24, or 32 dimensions in the first refinement direction, and we read 1, 2, 3, 4, or 5 best clusters in the second refinement direction. An important detail is, however, that we have to read new dimensions of every point read so far, and when reading new clusters, you have to read up to the dimensionality reached so far. A node in the graph represents the stage where the corresponding clusters and dimensions are considered for approximate result. Since the design of the pages is not changed during the search, reaching a node by any path leads to the same portion of the data to be considered. Therefore, any path to a certain node results in the same number of page accesses and the same approximate result set. Any path in this framework can be used as an approximate k-NN algorithm. However, the chosen path may be important because the intermediate steps of various paths through the same node have different average time-accuracy tradeoffs.

It is possible to choose the best path by a pre-analysis of the performance of each possible path as we will discuss in the performance evaluation section. Once a path is chosen at design time, the same path can be used for all k-NN queries. The bottom-left node in Figure 4 represents a possible initial step of the algorithm. The top-right node illustrates the case where 32 dimensions of 5 best clusters are taken into consideration for computing the answer to the k-NN query.

6 Analysis and Performance Evaluation of Approximate Nearest Neighbor Techniques

In this section, we perform a detailed performance analysis of the proposed and current state-of-the-art techniques. We first describe the performance evaluation setup and the data sets. We then briefly evaluate the performance of some of the basic techniques, i.e., partial sequential scan, KLT, VA-file, and VA$^+$-file. The feasibility and efficiency of *approximate* k-nearest neighbor searching can be observed even by these simple techniques, such as KLT-based approximate k-NN searching. Finally, we analyze the performance of the proposed cluster-based integrated approach and compare it with the current best approach for several data sets. We demonstrate how the proposed clustering based approach leads to very significant improvements over the existing techniques and maintains its superiority as data set size increases.

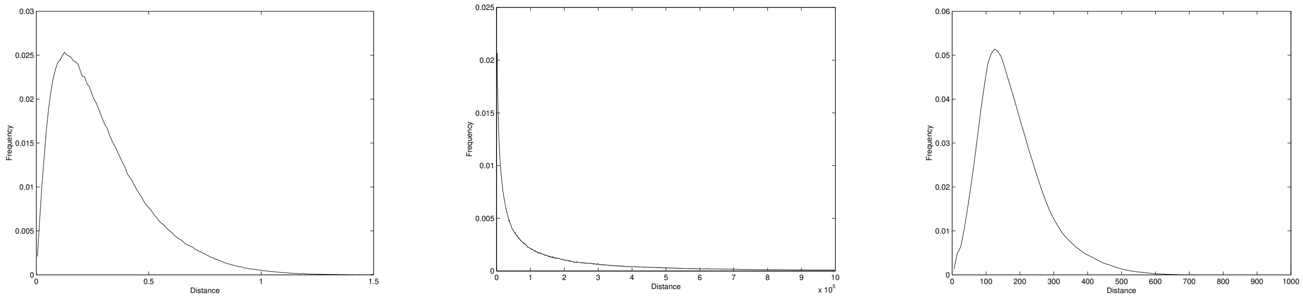
6.1 Performance Evaluation Setup

For performance evaluation purposes, we used various real-life data sets from different application domains. The first data set, *Color Histogram*, is a 64-dimensional color image histogram data set of size 12,000. The vectors represent color histograms computed from a commercial CD-ROM. The second data set, *Stock Time-series*, is a time-series data set which contains 360 days stock price movements of 2000 companies, i.e., 2000 data points with dimensionality 360. The third data set, *Satellite Image Texture (Landsat)*, is of size 100,000 with 60-dimensional vectors representing texture features of Landsat images [34]. Texture information of blocks of large aerial photographs are computed using Gabor filtering. The Gabor filter bank consists of 5 scales and 6 orientations of filters, therefore the total number of filters is $5 \times 6 = 30$. The mean and standard deviation of each filtered output are used to create the feature vector. Therefore the dimensionality of each vector becomes $30 \times 2 = 60$. This data set provides challenging problems in high dimensional indexing and is widely used in high dimensional indexing and similarity searching research [33, 22, 14]. In order to characterize each of these data sets, the frequencies of the distances between each pair of vectors for each of them are given in Figure 5.

For Color Histogram and Time-series data sets the page capacity is assumed to be 240 floating-point numbers, i.e., approximate page size=1K. Since the size of the Landsat image data set size is much larger, the page size is taken as 8KBytes in that case. We illustrate the results for 10-nearest-neighbor queries for all data points in the data set, i.e., query points=data points. The two types of metrics discussed before are used for performance evaluation:

1. Average number of pages accessed versus average number of false hits (= false dismissals in this case).

2. Average number of pages accessed versus average ratio of total distances. Let x be the query point, and let x_i and $\tilde{x}_i$ be the true and the approximate ith nearest neighbors, respectively. Then the



(a) Color Histogram (b) Stock Market Time-Series (c) Satellite Image Texture

Figure 5: Distance distribution of the data sets.

ratio of total distances is given by $D = \frac{\sum_{i=1}^{10} \|x-\tilde{x}_i\|^2}{\sum_{i=1}^{10} \|x-x_i\|^2}$. Now, it is easy to see that $D \geq 1$, with equality if and only if $\|x - \tilde{x}_i\| = \|x - x_i\|$ for $i = 1, \ldots, 10$, which means a set of exact 10-nearest neighbors is retrieved.

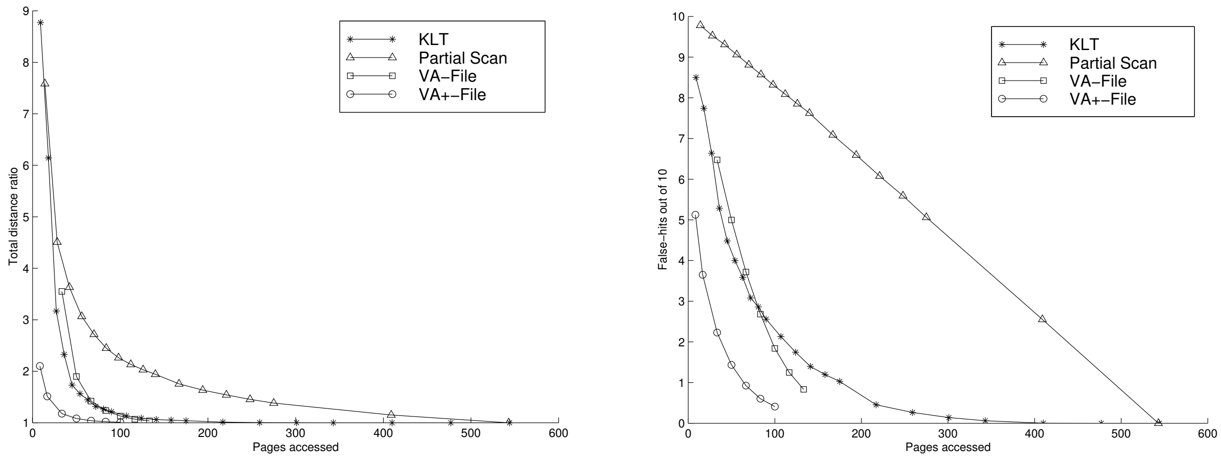
We note that with sequential search (of course with zero false hits and $D = 1$), one has to read around 534 pages, each 1K size, for the image color histogram data set and 3000 pages, each 1K size, for time-series data set. For Landsat image data set it is 2930 pages, each 8K size.

6.2 Performance Evaluation of Simple Progressive Techniques

In this section, we evaluate the performance of the four simple progressive approaches discussed earlier: partial sequential scan, KLT, VA-file, and VA$^+$-file based approximate k-NN searching. Using the two metrics discussed before, we also demonstrate the feasibility and efficiency of *approximate* k-nearest neighbor searching. The results presented in this section also establish that even simple techniques, such as KLT, can be extended and adapted for approximate searching in a very effective way. We performed experiments on various data sets and here illustrate the representative performance results for the color histogram data set, consisting of 64-dimensional image color histogram of 2000 data points. The performance behavior of the techniques is similar for all data sets.

The techniques discussed in this paper are interactive, i.e., the user is able to stop the query anytime he/she thinks it is appropriate and still gets a result with a certain error. Figure 6 illustrates a clear demonstration of the time accuracy trade-off achieved by all approximate NN approaches. As the retrieved number of pages increases, so does the quality of the approximations. It is important to adapt the approximation quality using the user as the reference point. If the user wants to get more accurate results, he/she can keep the query running until the desired result is obtained or no more data is available. Without waiting for the end of the query, which typically takes a long time, the user may also want to stop the search once he/she understands that the data set most likely does not have the desired answers. For each step of an interactive search, the expected quality can be estimated by a pre-analysis of the data set, as is done in this section.

The performance of the original VA-file approach is very close to that of the simple KLT-based technique. The VA$^+$-file [21] based approximation outperforms all the methods described so far. As an



(a) Metric: D

(b) Metric: Number of False Hits

Figure 6: Time-accuracy trade-off achieved by approximate search techniques

example, to reach $D = 1.1$, the partial sequential scan technique retrieves approximately 4 times as many pages as the KLT-based technique retrieves, and the KLT-based technique retrieves 2.5 times as many pages as the VA$^+$-file technique does. Similarly, by reading 80 pages, the KLT-based and the partial sequential scan techniques achieve $D = 1.27$ and $D = 2.5$, respectively, whereas the VA$^+$-file technique achieves $D = 1.03$. Figure 6(b) depicts the number of false hits in each of the approximate techniques.

6.3 Performance Evaluation of Cluster-based Approach

We implemented the proposed cluster-based technique and analyzed the performance of k-nearest neighbor queries. For clarity, we illustrate the comparisons with the VA$^+$-file based approximation technique since it outperformed all other methods. We summarize the results on three data sets, i.e., Color Histogram, Stock Time-series and Landsat Image Texture. First, we illustrate the detailed performance evaluation on the image color histogram data set, then we summarize the results for the stock time-series and Landsat image texture feature data sets.

We start with the experiments on *color histogram* data. As described before, first, a dimensionality reduction (from 64 to 8) is performed on each point in the data set using KLT. We implemented the modified K-means clustering algorithm to create the clusters and the corresponding cluster centroids. 4 different minimum cluster sizes are selected: 10, 20, 30, and 40. The number of clusters corresponding to these four scenarios are: 94, 47, 34, and 24, respectively. Since the representative dimensionality is 8, the amount of storage these representatives need is only 0.5%, 0.29%, 0.2%, and 0.15% of the data set. Decreasing the representative dimensionality or increasing the minimum cluster size results in less storage needs for the representatives. This percentage is much less for larger data sets or data sets with higher dimensionalities, such as the Landsat image feature and the time-series data sets that we use later in our performance evaluation. The largest ratio for the additional storage needed for representatives is only 0.029% of the entire time-series data set and 0.026% of the Landsat image data set. Since the size of the representative set is very small, the necessary processing on the representatives is also negligible compared to the entire data set.

We analyzed the performance by using the 5×4 progressive refinement discussed before. We read 8, 16, 24, or 32 dimensions in the first refinement direction, and we read 1, 2, 3, 4, or 5 best clusters in the second refinement direction.

The performance of the algorithm with minimum cluster size 10 is presented in Table 1. The path indicated by the rows with bold characters in Table 1, i.e., the path 1×8, 1×16, 2×16, 3×16, 3×24, 4×24, 4×32, and 5×32, corresponds to the path illustrated in Figure 4, and is actually the best path according to the error measure D. This path can be found by using dynamic programming technique on possible paths in the progressive refinement framework. The goal is to achieve best accuracy by retrieving minimal number of pages. The same performance results, in comparison with those of the VA$^+$-file based technique, are plotted in Figure 7. In the figures, only the best paths are considered for clarity.

# Clusters × Dimensions	# Pages accessed	# False Hits	D
1×8	**1.0000**	**4.8410**	**1.5669**
2×8	2.0000	3.7195	1.3935
3×8	3.0000	3.3420	1.3502
4×8	4.0000	3.1965	1.3333
5×8	5.0000	3.1365	1.3229
1×16	**2.0000**	**4.5020**	**1.3730**
2×16	**4.0000**	**2.8320**	**1.1638**
3×16	**6.0000**	**2.1190**	**1.1104**
4×16	8.0000	1.7890	1.0885
5×16	10.0000	1.6335	1.0777
1×24	3.0000	4.4530	1.3415
2×24	6.0000	2.6035	1.1287
3×24	**9.0000**	**1.7110**	**1.0728**
4×24	**12.0000**	**1.2375**	**1.0490**
5×24	15.0000	0.9825	1.0380
1×32	4.0000	4.4450	1.3258
2×32	8.0000	2.5530	1.1112
3×32	12.0000	1.6015	1.0536
4×32	**16.0000**	**1.0795**	**1.0305**
5×32	**20.0000**	**0.7795**	**1.0200**

Table 1: Time-accuracy performance achieved by the Clustering with minimum cluster size 10

As an illustrative example, by reading 16 pages, the cluster-based algorithm achieves an error of $D = 1.03$, where as the VA$^+$-file technique reaches an error level of $D = 1.56$. Similarly, to achieve the accuracy that the clustering based technique achieves reading 16 pages, the VA$^+$-file technique reads about 75 pages. The cluster-based technique achieves speedup from 4 to 15 compared to the VA$^+$-file technique.

As mentioned before, the size of the representatives is negligible compared to the size of the data set. The overhead due to the representatives is very small: 4, 2, 2, and 1 additional page accesses for cluster sizes 10, 20, 30 and 40, respectively.

The performance results are very similar for various minimum cluster sizes. Figure 8 illustrates the comparison of time-accuracy performances achieved by the VA$^+$-file based technique and clustering with

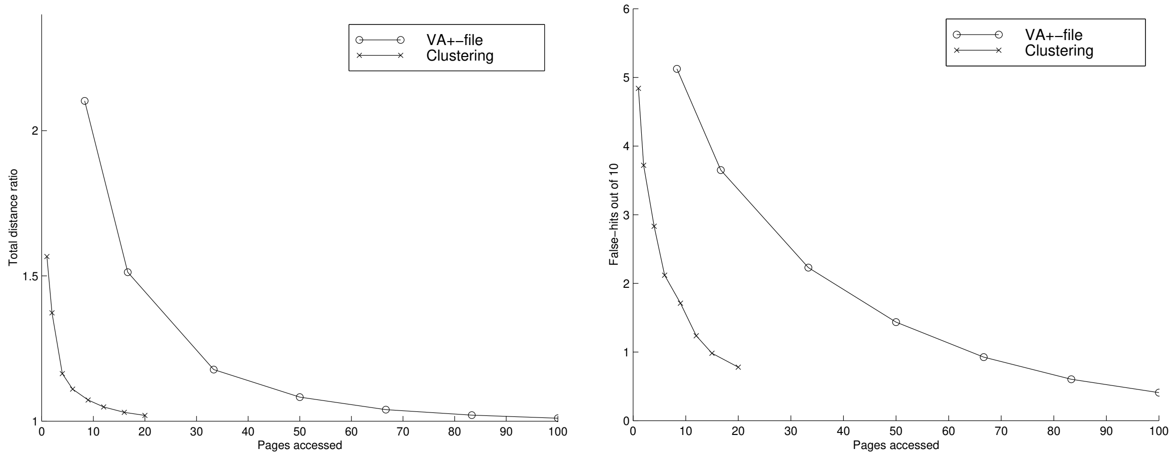


Figure 7: Comparison of time-accuracy performances achieved by the Cluster-based technique (with minimum cluster size 10) and VA$^+$-file

minimum cluster size 40. The figure is again drawn for the best path.

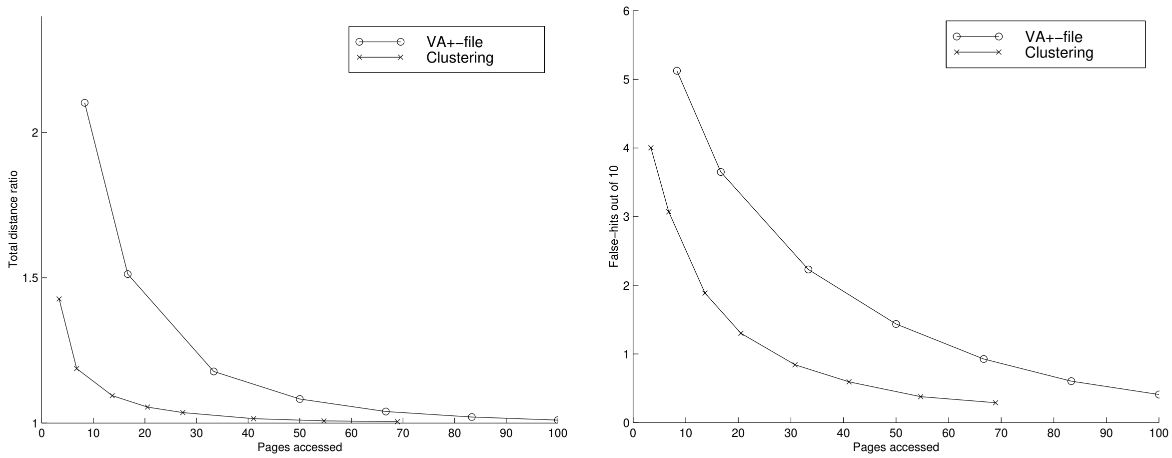


Figure 8: Comparison of time-accuracy performances achieved by the Cluster-based technique (with minimum cluster size 40) and VA$^+$-file

We also analyze the scalability of our techniques as the data set size grows. We evaluate the performance of full and partial sequential search, KLT-based dimensionality reduction, VA$^+$-file, and clustering based technique by comparing the number of pages accessed as the data set size grows. Figure 9 illustrates the comparison of five techniques. The curves for sequential scan, KLT, VA$^+$-file, and clustering based techniques represent the result set with a relative distance of $D = 1.05$, where $D = 1$ represents the actual result without error. The VA$^+$-file based approach achieves significant speedup compared to the full and partial sequential scan, and the KLT-based technique. The clustering based approach achieves very high speedups compared to the other two techniques and scales much better as the data size grows. For the data set with 1000 data objects, our cluster-based technique accesses 30 times less pages than sequential scan, 29 times less than the partial sequential scan, 9.3 times less than KLT-based technique, and finally 3.5 times less than the VA$^+$-file approach access for $D = 1.05$. As the data size increases,

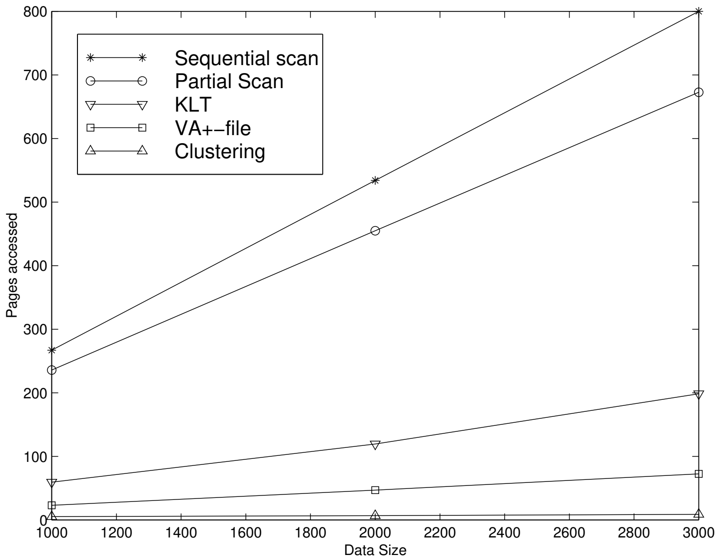


Figure 9: Comparison of number of pages accessed among sequential scan, partial scan, KLT, VA$^+$-file, and cluster-based approximations, as the data set size grows linearly. All the curves are for $D = 1.05$.

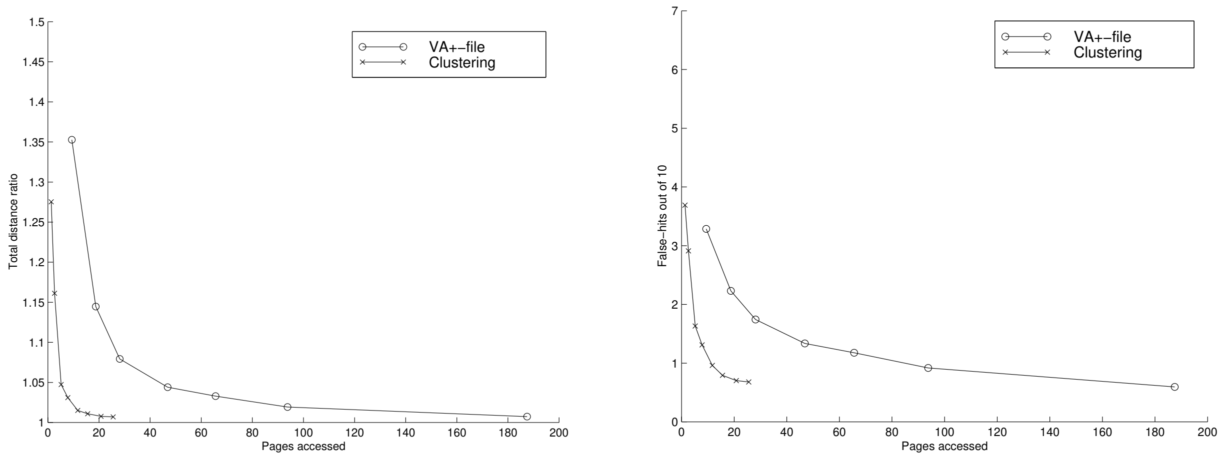
Data Size(# of pages)	1000	2000	3000	Data Size(# of pages)	1000	2000	3000
EXACT	267.00	534.00	800.00	EXACT	267.00	534.00	800.00
SEQ. (partial) $D = 1.05$	255.19	498.99	740.89	SEQ. (partial) $F = 1$	250.12	490.47	729.67
SEQ. (partial) $D = 1.1$	235.75	454.99	672.78	SEQ. (partial) $F = 1.5$	237.01	464.21	690.00
KLT $D = 1.05$	81.31	160.32	258.72	KLT $F = 1$	82.28	177.02	274.24
KLT $D = 1.1$	59.41	119.50	198.39	KLT $F = 1.5$	62.68	135.97	222.66
VA$^+$-FILE $D = 1.05$	30.67	62.73	96.94	VA$^+$-FILE $F = 1$	30.92	64.23	100.85
VA$^+$-FILE $D = 1.1$	23.19	47.01	72.61	VA$^+$-FILE $F = 1.5$	23.23	48.65	77.92
CLUSTER $D = 1.05$	8.78	11.87	15.65	CLUSTER $F = 1$	11.37	14.79	24.06
CLUSTER $D = 1.1$	5.36	6.83	8.95	CLUSTER $F = 1.5$	7.26	10.10	13.75

Table 2: Comparison of number of pages accessed for data sets with 1000, 2000, and 3000 data objects. Metrics: Left table: D, Right table: *False Hits out of 10.*

the speedups achieved by our techniques increase. For example, for 3000 data objects and $D = 1.05$, our cluster-based technique achieves speedups 51.1, 47.3, 16.5, and 6.2 compared to full sequential scan, partial sequential scan, KLT-based, and VA$^+$-file based techniques, respectively.

The speedups are higher for the case $D = 1.1$, i.e., 49.8, 44, 11.1, and 4.32 times better than full sequential scan, partial sequential scan, KLT-based, and VA$^+$-file techniques, respectively. Again as the data size increases, the speedups achieved by our techniques increase. For 3000 data objects and $D = 1.1$, the cluster-based technique achieves speedups 89.4, 75.2, 22.2, and 8.1 compared to full sequential scan, partial scan, KLT-based, and VA$^+$-file based techniques, respectively. We repeated the similar experiments and got similar results for the metric that considers false hits. Table 2 illustrates the results of all these experiments, i.e., the number of page accesses in 10-NN queries on data sets containing 1000,2000, and 3000 data points, for D to be 1.05 and 1.1, and for number of false hits to be 1 and 1.5.

For the 360-dimensional time-series data set, the performance comparison of the techniques is very similar to the performance comparison for the image color histogram data set. We note that, for the



(a) Metric: D

(b) Metric: Number of False Hits

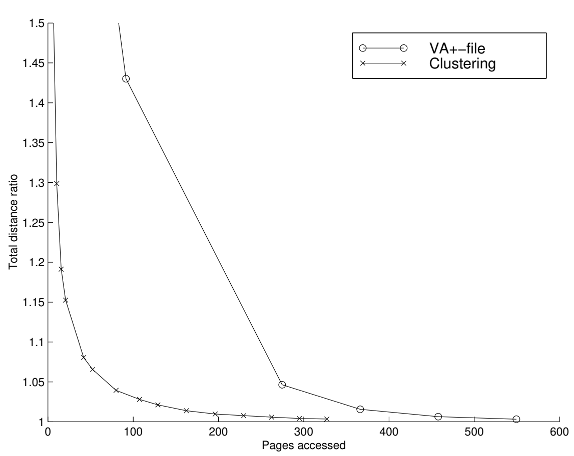
Figure 10: Comparison of time-accuracy performances achieved by the Clustering based technique and VA$^+$-file on Time-series Dataset

time-series data set, the speedups achieved by our technique are more than the speedups that we obtained for image feature data set. Figure 10 illustrates a sample performance comparison of the VA$^+$-file and our clustering based technique for both metrics, i.e., D and the number of false hits. Some representative numbers from these experiments are as follows. To achieve $D = 1.1$, our clustering based technique reads an average of 4 pages whereas VA-file reads an average of more than 25 pages. Similarly, for $D = 1.05$, the clustering based technique reads an average of 5.1 pages, whereas VA-file reads average of 43.7 pages in each query. The speedup achieved for $D = 1.1$ is 6.3, and the speedup achieved for $D = 1.05$ is 8.5. These speedups were less for the image color histogram data set, i.e., 5.3 and 6.8, respectively.

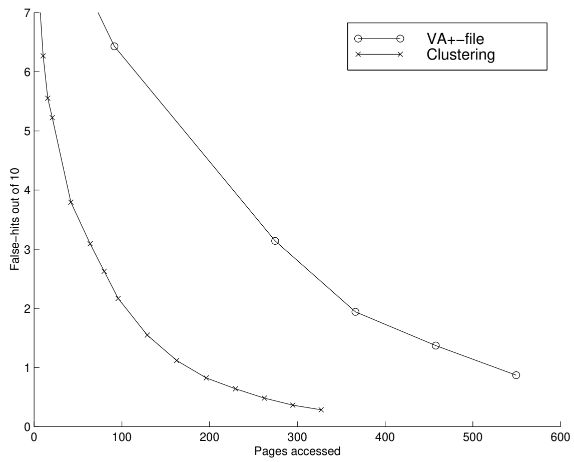
We repeated the experiments for the Landsat image data set which contains 100,000 60-dimensional texture feature vectors. Since the data set size is much larger, the page size is taken as 8KBytes. The percentage of storage needed for the representative set is only 0.026% of the data set. In general, the speedups that are achieved for this data set are better than the color histogram data set and comparable with the time-series data set. The results show that our clustering based technique maintains its superiority also for large data sets. Figure 11 illustrates a sample performance comparison of the VA$^+$-file and our clustering based technique for both metrics, i.e., D and the number of false hits. Some representative numbers from these experiments are as follows. To achieve $D = 1.1$, our clustering based technique reads an average of 36 pages where VA-file reads an average 249 pages. Similarly, for $D = 1.05$, the clustering based technique reads an average of 68 pages, where VA-file reads average of 273 pages in each query. The speedup achieved for $D = 1.05$ is 4, and the speedup achieved for $D = 1.1$ is 6.9.

7 Discussion

In this paper, we developed a general framework for approximate nearest neighbor queries. We first categorized the current approaches into two classes: *Retrieved Set Reduction* based techniques, which



(a) Metric: D



(b) Metric: Number of False Hits

Figure 11: Comparison of time-accuracy performances achieved by the Clustering based technique and VA$^+$-file on Landsat Image Dataset

reduce the data set that needs to be examined, and *Representative Size Reduction* based techniques, which reduce the representation of each data object. We developed two simple progressive approximate k-NN techniques that are instances of these classes, i.e., searching with partial sequential scan and searching on sub-vectors based on KLT. In both approaches, a user can stop the retrieval process once enough information has been retrieved. We then proposed a new technique that effectively combines the advantages of the two class of approaches into a single framework, i.e., reducing the size of the retrieved set and feature vector sizes for efficient approximate searching. The retrieved portion of data is reduced by using clustering (an adaptation of K-means clustering), and the feature vectors within a cluster are organized to support efficient interactive approximate searching. Our cluster-based approach allows a user to progressively explore the approximate results with increasing accuracy. By evaluating these approaches in the context of several real data sets, we first demonstrated the feasibility and efficiency of approximate nearest neighbor searching by using the main metrics discussed in the paper. We then showed the superiority of the proposed cluster-based technique over the current best techniques for approximate searching. We performed experiments on several real data sets and obtained significant speed-ups by using the proposed cluster-based technique over current approaches.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. In *ASPLOS-VIII*, pages 81–91, September 1998.

[2] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *4th Int. Conference on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[3] S. Arya and D. M. Mount. Approximate range searching. In *11th Annual Symposium on Computational Geometry*, pages 172–181, June 1995.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. In *5th Ann. ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[5] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[6] S. Berchtold, C. Bohm, D. Keim, and H. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. ACM Symp. on Principles of Database Systems*, pages 78–86, Tuscon, Arizona, June 1997.

[7] S. Berchtold, C. Bohm, and H.-P. Kriegel. The Pyramid-Technique: Towards breaking the curse of dimensionality. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 142–153, Seattle, Washington, USA, June 1998.

[8] S. Berchtold, D. Keim, and H. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 28–39, Bombay, India, 1996.

[9] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. The Asilomar report on database research. *ACM Sigmod Record*, 27(4), December 1998.

[10] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful. In *Int. Conf. on Database Theory*, pages 217–225, Jerusalem, Israel, January 1999.

[11] K. R. Castleman. *Digital Image Processing.* Prentice-Hall, Inc., 1996.

[12] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proc. Int. Conf. Data Engineering*, pages 440–447, Sydney, Australia, 1999.

[13] X. Cheng, R. Dolin, M. Neary, S. Prabhakar, K. Ravikanth, D. Wu, D. Agrawal, A. El Abbadi, M. Freeston, A. Singh, T. Smith, and J. Su. Scalable access within the context of digital libraries. In *IEEE Proceedings of the International Conference on Advances in Digital Libraries, ADL*, pages 70–81, Washington, D.C., 1997.

[14] P. Ciaccia and M. Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proc. Int. Conf. Data Engineering*, pages 244–255, San Diego, California, March 2000.

[15] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[16] O. Egecioglu and H. Ferhatosmanoglu. Circular data-space partitioning for similarity queries and parallel disk allocation. In *Proc. of IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 194–200, November 1999.

[17] O. Egecioglu and H. Ferhatosmanoglu. Dynamic dimensionality reduction and similarity distance computation by inner product approximations. In *Proceedings of the 9th ACM Int. Conf. on Information and Knowledge Management*, pages 219–226, McLean, Virginia, November 2000.

[18] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[19] H. Ferhatosmanoglu, D. Agrawal, and A. El Abbadi. Clustering declustered data for efficient retrieval. In *Proc. Conf. on Information and Knowledge Management*, pages 343–350, Kansas City, Missouri, November 1999.

[20] H. Ferhatosmanoglu, D. Agrawal, and A. El Abbadi. Concentric hyperspaces and disk allocation for fast parallel range searching. In *Proc. Int. Conf. Data Engineering*, pages 608–615, Sydney, Australia, March 1999.

[21] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the 9th ACM Int. Conf. on Information and Knowledge Management*, pages 202–209, McLean, Virginia, November 2000.

[22] A. Gionis, P. Indyk, and R. Motwani. Similarity searching in high dimensions via hashing. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 518–529, Edinburgh, Scotland, UK, September 1999.

[23] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[24] R. M. Haralick and L. G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100–132, January 1985.

[25] T. Kailath. *Modern Signal Processing.* Springer Verlag, 1985.

[26] K. V. R. Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 166–176, Seattle, Washington, June 1998.

[27] H. Karhunen. Uber lineare methoden in der wahrscheinlich-keitsrechnung. *Ann. Acad. Science Fenn*, 1947.

[28] K. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3:517–542, 1995.

[29] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28:84–95, January 1980.

[30] M. Loeve. Fonctions aleatoires de seconde ordre. *Processus Stochastiques et Mouvement Brownien*, 1948.

[31] D. B. Lomet and B. Salzberg. The hb-tree: A multi-attribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, December 1990.

[32] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Math. Stat. and Prob*, volume 1, pages 281–196, 1967.

[33] B. S. Manjunath. Airphoto dataset. http://vivaldi.ece.ucsb.edu/Manjunath/research.htm, May 2000.

[34] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–42, August 1996.

[35] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture and shape. In *Proc. of the SPIE Conf. 1908 on Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187, February 1993.

[36] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall, Inc., 1989.

[37] J. T. Robinson. The kdb-tree: A search structure for large multi-dimensional dynamic indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.

[38] N. Roussopoulos, S. Kelly, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, San Jose, California, May 1995.

[39] T. Seidl and Kriegel H.-P. Efficient user-adaptable similarity search in large multimedia databases. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 506–515, Athens, Greece, 1997.

[40] V.S. Subrahmanian. *Principles of Multimedia Database Systems.* Morgan Kaufmann Publishers, Inc., San Francisco, California, 1999.

[41] R. Weber and K. Bohm. Trading quality for time with nearest-neighbor search. In *Proc. Int. Conf. on Extending Database Technology*, pages 21–35, Konstanz, Germany, March 2000.

[42] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 194–205, New York City, New York, August 1998.

[43] D. White and R. Jain. Similarity indexing with the SS-tree. In *Proc. Int. Conf. Data Engineering*, pages 516–523, 1996.

[44] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with M-trees. *The VLDB Journal*, 4:275–293, 1998.