Access Structures for Angular Similarity Queries

Tan Apaydin, Hakan Ferhatosmanoglu

The Ohio State University, Computer Science and Engineering

{apaydin,hakan}@cse.ohio-state.edu

Abstract

Angular similarity measures have been utilized by several database applications to define semantic similarity between various data types such as text documents, time-series, images, and scientific data. Although similarity searches based on Euclidean distance have been extensively studied in the database community, processing of angular similarity searches has been relatively untouched. Problems due to a mismatch in the underlying geometry as well as the high dimensionality of the data make current techniques either inapplicable or their use results in poor performance. This brings up the need for effective indexing methods for angular similarity queries. We first discuss how to efficiently process such queries and propose effective access structures suited to angular similarity measures. In particular, we propose two classes of access structures, namely *Angular-sweep* and *Cone-shell*, which perform different types of quantization based on the angular orientation of the data objects. We also develop query processing algorithms that utilize these structures as dense indices. Proposed techniques are shown to be scalable with respect to both dimensionality and the size of the data. Our experimental results

This research was partially supported by Department of Energy (DOE) grant DE-FG02-03ER25573 and National Science Foundation (NSF) grant CNS-0403342.

on real data sets from various applications show two to three orders of magnitude of speedup over the current techniques.

Index Terms: Angular query, Performance, Indexing, Angular similarity measures, High dimensional data.

I. Introduction

Similarity measures based on angular distances have been effectively utilized in a wide range of modern database applications. The general approach is to first generate a multi-dimensional feature vector for each data object, then use an angular distance between representative vectors as a measure of similarity in the semantic space. For example, the cosine angle measure computes the difference in direction, irrespective of vector lengths, where the distance is given by the angle between the two vectors. Being scale-invariant is a particularly useful property in heterogeneous or real-time databases since preprocessing for normalization is not required [30]. In fact, angular measures are closely related to the Euclidean distance metric. However, depending on the application, there are cases where one measure is preferred over the other.

A. Applications

Angular measures have been used to compare a large variety of data types. We list some examples below.

Astronomy and astrophysics The apparent positions and separations of constellations and objects in the sky are not determined by the linear distances between two objects but by their angular separation. Their positions are related to angular distances or angular separations from well known or readily identified reference positions or objects. The standards to measure some distances are the angles between imaginary lines coming from the objects or positions of interest and intersecting at the eye of the observer. In order to determine an arc-angle or distance between two vectors, the dot product and the Cartesian difference of the vectors are used. Being the natural underlying distance measure, angular measures are commonly used in querying astronomical data [41], [4], [36], [24].

Aviation An angular query in an Air Traffic Control (ATC) system is to find all objects within the flight route of the plane [13]. The route consists of several segments of lines and the query is defined as a series of cones because of the uncertainties as the distance from a starting point increases. Similarly, an angular query can be defined in a Space Transportation System to check the objects, e.g., satellites, within the route of a spacecraft [23].

Graphics Data processing based on angular regions are common in computer graphics applications. With spot light sources, to make an appearance determination of an object, a cone is specified and a spot direction which provides the center of the cone is defined. The light source to a surface direction and the inner product with the spot direction is computed. If the result is less than the cosine of spot angle, the light source is not visible at that surface [39].

Images Similarity measures for retrieval based on the angular distance are also shown to be efficient and robust for image processing applications [38], [29]. For example, feature vectors are generated based on segmentation and pixel analysis, and the angle between query vector and each indexed representative vector is utilized for a more accurate similarity searching. This is done by first calculating the cosine of vectors and then computing the angle between them [3], [17].

Protein structures For classifying protein folds and for revealing a global view of the protein structures, structural similarity measures based on angular distances, i.e. cosine, are utilized to provide an objective basis. Its efficiency and scale-invariance properties make the angular distance particularly useful in this domain [2], [16], [11].

Text documents Angular similarity has been popularly used in information retrieval for semantic analysis of text documents. After removing the stop words, articles, conjunctions, etc., the number of occurrences of each word/term is computed and stored as a feature vector for each document [42], [10], [33], [34], [35], [30], [37], [27]. Since the number of features may be very large, some information retrieval techniques, such as Latent Semantic Analysis (LSA/LSI) [1], [14], [15], apply some preprocessing to reduce the dimensionality of the vectors. Similarity between documents is then measured as the cosine of the angle between these feature vectors [35], [42], [10], [30].

Time series Correlation measures, which are angular distances for standard vectors, are widely used in the analysis of time series data [25], [31]. There have been recent studies that apply a spatial autocorrelation among spatial neighboring time series by locating them on the surface of a multi-dimensional unit sphere. Then the correlation coefficient (r) of these transformed time series is formulated in terms of the cosine of the angle between them [43], [44]. Actually, any transformation ending up with time series whose mean is zero, makes the same effect, i.e., r of the new two series is equal to the cosine of the angle between them.

The proposed techniques are targeted towards the applications that use a single origin for measuring the angular similarity. This is the case for most of the applications presented above, including images, protein structures, text documents, and time series analysis. Extensions are needed for the proposed techniques to be utilized in dynamic geographic applications, such as the aviation, where the angle needs to be dynamically computed with respect to changing origins.

B. Technical Motivation

For efficient processing of queries, indexing support is vital, and angular similarity queries are no exception. An angular similarity query corresponds to the shape of a conic in the geometric space. On the other hand, current index structures, including well-known families of R-trees [21], [5], grid files [28] and VA-Files [40], use rectangles and/or circles as the underlying geometric shapes. Similarly, partitioning-based approaches [9], [18] use an incompatible organization of the space. On the other hand, a popular similarity measure, *cosine*, is not a metric space function (which is the building block of M-trees) since it does not satisfy the triangle inequality, and for this reason, M-trees [12] can not be applied. Due to *mismatch of geometries* and high dimensionality of the feature space, current techniques are either inapplicable or perform poorly for applications that utilize angular distances. They have a poor performance even when the data objects are transformed into their native domain where they were originally developed (e.g., normalizing the data to use Euclidean distance). The need is further amplified for higher dimensions, where the current techniques retrieve the majority, if not all, of the disk pages that do not include any related information.

C. Our Approach

We propose access structures to enable efficient execution of queries seeking angular similarity. We explore quantization based indexing, which scales well with the dimensionality, and propose techniques that are better suited to angular measures than the conventional techniques. In particular, we propose two classes of scalar quantizers and index structures with query processing algorithms. A quantizer is designed for each data object considering its angular orientation. It is based on a partitioning technique optimized for angular similarity measures which results in significant pruning in processing of angular queries. The first technique partitions the space into multi-dimensional pyramids, and quantizes the data based on the partitions in a sweeping manner. The second technique quantizes the partitions following a shell structure.

Among the current techniques that are comparable with the proposed approaches for angular queries, VA-Files is the most convenient and has the best performance, and that is discussed in Section II-B. For this reason, the performance of angular range and k-NN queries are analyzed and compared with VA-Files, on synthetic and real data sets from a variety of applications mentioned earlier. Experimental results establish that each proposed technique has its unique advantages, and they both achieve a significant performance improvement over VA-Files (e.g., three orders of magnitude speedup for angular range queries over a text data set).

The paper is organized as follows. In the following section, we present a background information about angular similarity and quantization based approaches. We highlight the problem behind the conventional techniques and briefly introduce our approach. In Section III, we describe our first quantization technique and give details about the processing of angular range and angular k-NN queries. Section IV describes our second technique and explains the query processing. Representative experimental results are presented in Section V. Section VI concludes the paper with a discussion.

II. Background

In this section, we first define similarity queries with angular measures, e.g., cosine, inner product, and correlation coefficient, and then describe the quantization approach for high dimensional indexing.

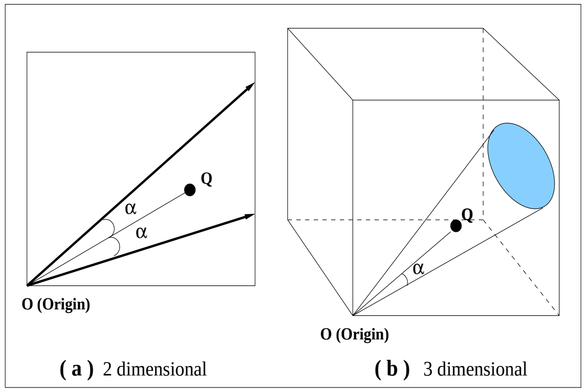


Fig. 1. Angular range query

A. Angular Similarity

An angular range query is defined by (Q, α), where Q is the query point $(q_1, q_2, \ldots, q_d)$ in a d dimensional space, and α denotes the angle that represents the range, and seeks all data in the cone whose axis ($\overline{OQ}$) is the line defined by the origin O and the query point, Q, and whose apex or the vertex is on the origin as illustrated in Figure 1. The angle between the axis and all the lines on the lateral surface of the cone is α. All the feature vectors that are equally similar to the query vector are on an equivalence region which corresponds to a conic surface.

If a feature vector is represented as $X(x_1, x_2, \ldots, x_d)$, the *cosine* angle measure (a widely used similarity measure) is defined by the following formula

$$cos(\alpha) = (\sum_{i=1}^{d} x_i q_i)/(||X|| \cdot ||Q||) \tag{1}$$

Without loss of generality, if we assume the query point to be normalized, then Equation 1 can be simplified to $cos(\alpha) = \frac{\sum_{i=1}^{d} x_i u_i}{||X||}$, where $U(u_1, u_2, \ldots, u_d)$ is the unit normalized query. Additionally, if the feature vectors are also normalized, then the equation becomes the *inner product* of the query with a feature vector in the domain. Similarly, Pearson's *correlation coefficient* [20], another popular measure, can be defined as the inner-product of two vectors when they are standardized, i.e., the means of the new vectors are 0 and the standard deviations are 1.

For simplicity, we based our initial discussions on a 3-dimensional model, which will later be extended to higher dimensions. Let Q be a 3-dimensional query point, and $u = (u_1, u_2, u_3)$ be the unit vector which is the normalization of the query vector. That is, $u_i{=}q_i/\sqrt{q_1^2 + q_2^2 + q_3^2}$ for i = 1, 2, 3.

The expression for an equivalence conic surface in angular space is the following equation.

$$(x_1u_1 + x_2u_2 + x_3u_3)^2 = (x_1^2 + x_2^2 + x_3^2)cos^2\alpha. \quad (2)$$

B. Quantization Based Access Structures

A large number of indexing techniques have been proposed in the literature to improve the efficiency of similarity queries in multi-dimensional data sets. It has been established that the well-known indexing techniques and their extensions are outperformed on average by a simple sequential scan if the number of dimensions exceeds 10 [8], [40]. Quantization has been proposed as a more effective alternative to the tree-based approaches. For example, the VA-File, a dense-index based on scalar quantization, has been shown to be superior to the traditional techniques [40]. In this technique, the data space is split into 2^b rectangular cells where b is the total number of bits specified by the user or the system requirements, such as available memory. Each dimension is allocated b_i bits, which are used to create 2^{b_i} splits in the corresponding dimension. As a result, each cell has a bit representation of length b which is used to approximate the data points that fall into the corresponding cell. The dense index, e.g. VA-File, is simply an array of these bit vector approximations (bit-strings) based on quantization of the original feature vectors. There have been extensions to VA-Files, e.g., IQ-tree [6] and A-tree [32], which are proposed to build the VA-File in a hierarchical way. From now on, we will interchangeably use the terms *bit vector approximation* and *bit-string*.

The quantization based indices can be used to approximately answer the query without accessing any real data, or to filter the data and eliminate some of the irrelevant data to give an exact answer. As an example, exact nearest neighbor queries can be executed as follows. In the first phase, quantized data is scanned sequentially and lower and upper bounds on the distance of each vector to the query vector are computed. If a bit-string is encountered such that its lower bound exceeds the (k-th) smallest upper bound found so far, the corresponding object can be eliminated. In the second phase, the algorithm traverses the real data that correspond to the candidate set, in the order of their lower bounds. If a lower bound is reached that is greater than the (k-th) actual nearest neighbor distance seen so far, then the algorithm stops retrieving the rest of the candidates. Other queries, such as range queries, can be

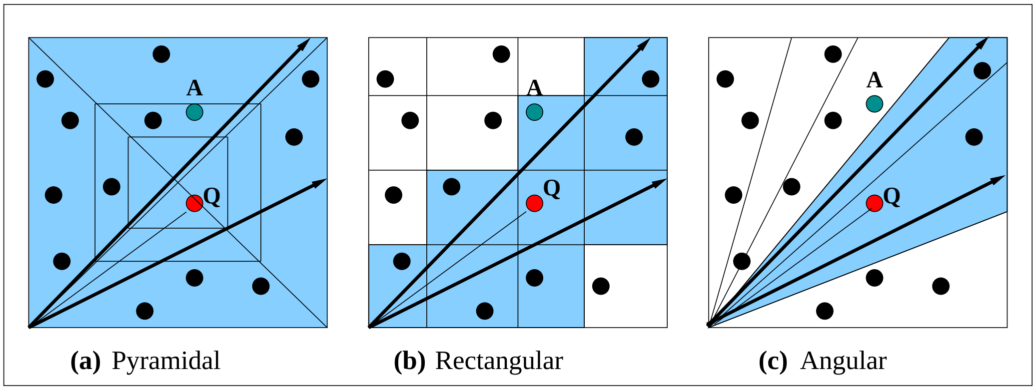


Fig. 2. Different Organizations

executed in a similar way where the first step identifies the candidates using the bit-strings and the second step computes the actual results. Two-step query processing guarantees that no actual result is missed.

A VA-File example is given in Figure 2(b). The rectangular partitioning based quantization represents grid based vector approximations. The data subspace between the arrows shows the angular query space. The goal is to find the feature vectors (data points) that are in this query space. This space intersects a large number of approximations (based on the rectangles) and, thus, the technique retrieves many irrelevant data points. For example, the data point A will be retrieved since its approximation is intersected with the query space, although A itself is not in the query space, and there will be many irrelevant points like A. In higher dimensions, the number of similar points gets higher and they cannot be eliminated.

All of the above mentioned techniques are specifically developed for Euclidean or general metric spaces. Due to a mismatch of geometries, they are either infeasible or ineffective for our purposes. For instance, our experiments on an adaptation of conventional VA-Files for angular measures (by normalization) show a very significant degradation on the performance. Similarly, Figure 2(a) depicts the partitioning structure of the Pyramid technique [9] which obviously has more irrelevant points due to geometric mismatch. We compared our techniques with VA-Files which achieves the best performance for current approaches.

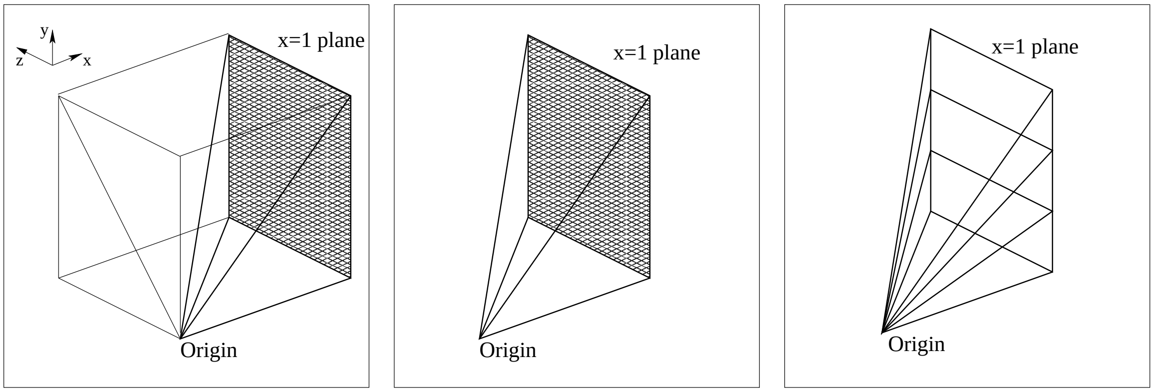
The only work that is considering the geometry mismatch problem into account is a declustering technique based on a conical partitioning [19]. However, this approach works only for uniform data and does not scale up with dimensionality, hence it is infeasible for the mentioned applications. There is a need for access structures that scale well with dimensionality and that are optimized for angular similarity measures which are used in several database applications.

III. Angular Sweep Quantizer (AS-Q)

We propose novel access structures based on an effective quantization of the data objects using their angular orientations with respect to the origin. A bit-string for each object is generated based on their angular positions as opposed to their values in each dimension. For 2 dimensions, the underlying partitioning for our quantization technique is illustrated in Figure 2(c), which is much better suited for angular queries than pyramidal or rectangular structures in Figures 2(a) and (b). For the clarity of the development, without loss of generality, we assume that the data space is a unit hypercube, i.e., $[0...1]^d$, and we only use the positive coordinates. The formulations do not depend on this assumption and knowing the limit values (minimum and maximum) for each dimension is enough for the development. In this section, we describe Angular Sweep Quantizer (AS-Q), our first access structure for high dimensional angular similarity searches. We first describe the partitioning used as a basis of the quantizer and develop a dense index by an effective quantization of the data. We then describe how to process range and k-NN queries.

A. Data-Space Partitioning

The first step for the AS-Q technique, for d number of dimensions, is to divide the data space into d major hyperpyramids, each having the side planes of the unit hypercube as the base area, and the origin $O = (0, 0, \ldots, 0)$ as the apex. Figures 3(a) and (b) illustrate the 3-dimensional example of a major pyramid whose base is $x = 1$ plane. Note that for 3-dimensions, there are 3 major pyramids (whose bases are $x = 1$, $y = 1$ and $z = 1$ planes - or squares) that cover the entire data space, namely unit cube. The major pyramids are then divided into sub-pyramids as shown in Figure 3(c). The sub-pyramids can be constructed either as equi-volumed, or as equi-populated as will be discussed in Section III-B. These



(a) (b) (c)

Fig. 3. Underlying partitions for the first technique

sub-pyramids are the underlying partitions for the quantization, and a bit-string will be derived for each of them.

B. Angular Sweep Quantization

Bit Allocation Once the partitioning is performed, the next step is to develop a bit allocation scheme where each partition is assigned to a bit-string. The number of bit-strings allocated for a major pyramid is equal to the number of sub-pyramids in that major pyramid, and thus proportional to the number of data points in the major pyramid (i.e., the higher the number of data points, the higher the number of assigned bit-strings). Hence the partitions are assigned non-uniform number of bit-strings, which is well suited to the distribution of the data.

The pyramids are defined by the equations of their enclosing planes. For instance, in 3-dimensions, the pyramid formed by the origin and the base plane $x = 1$, is enclosed by the planes $x = y$, $x = z$, $y = 0$, $z = 0$, and $x = 1$. We mainly name a major pyramid by its base plane, i.e., "*x=1 major pyramid*". Representing the dimensions x, y, z by x_1, x_2, x_3 respectively; a particular point $P_k(x_1, x_2, \ldots, x_d)$ is contained in major pyramid $x_{d_{max}} = 1$, where $x_{d_{max}}$ is the dimension with the greatest corresponding value, i.e., $\forall_i(x_{d_{max}} \geq x_i)$. For instance in 3-dimensions, P(0.7, 0.3, 0.2) will be in "*x_1=1 major pyramid*" since 0.7 (x_1) is greater than both 0.3 (x_2) and 0.2 (x_3). The bit allocation scheme is as follows.

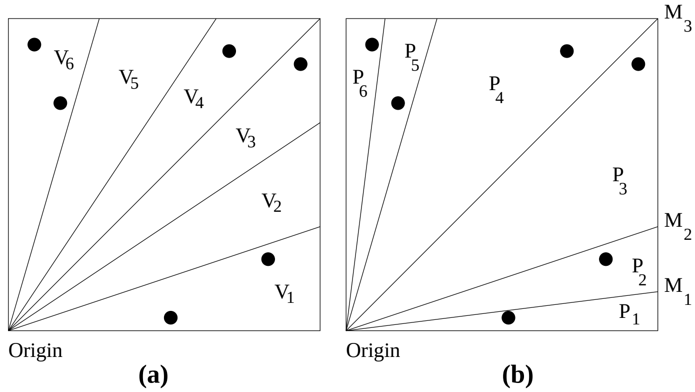


Fig. 4. a) equi-volume b) equi-populated structure

For d dimensions, a data set P consisting of N points, and total number of bits b. Let $Pop(i)$ give the population for major pyramid i, and $BS(i)$ is a list that will keep the bit-strings assigned for that major pyramid in sorted order.

1) For each major pyramid i, find $Pop(i)$.
2) For each major pyramid i, assign the next $\frac{Pop(i)}{N} \times 2^b$ bit-strings to $BS(i)$.

Generating Bit-Strings Major pyramids are sliced into sub-pyramids as we mentioned before. For example, in Figure 3(c), $x = 1$ major pyramid is sliced according to y dimension, i.e., y is the split dimension for $x = 1$. Similarly, $z = 1$ major pyramid is sliced according to x dimension, and in this case x is the split dimension for $z = 1$. The number of bit-strings allocated to each major pyramid is determined on the basis of this chosen split dimension. This dimension could be the one with the greatest spread or with the greatest variance. Alternatively, the split dimension could be chosen in a systematic manner. For instance, for all dimensions except the first dimension x_1, the base planes of the major pyramids can be divided according to the first dimension, x_1. And the first dimension, x_1, can be divided with respect to any of the others, say x_2. Another approach would slice the major pyramids in round robin manner. For instance, $x_1 = 1$ according to x_2, $x_2 = 1$ to x_3, $x_3 = 1$ to x_1 (in a cyclic manner). In the subsequent formulations, without loss of generality, we assume that the major pyramids are sliced in this manner, i.e., $x_i = 1$ with respect to x_{i+1} for $i < d$, and x_d with respect to x_1. The only reason for this assumption is the simplification of implementations.

We utilize both equi-volumed and equi-populated partitionings. In the equi-populated version, each bit-string represents an equal number of data points, as illustrated in Figure 4(b). Equi-volumed partitioning, as shown in Figure 4(a), is easier to compute and store. In order to produce a bit-string for a given data point P_k, the following general algorithm is used for both equi-volumed and equi-populated partitionings.

Algorithm: In a major pyramid, let $R(P_k)$ be the rank of the sub-pyramid (approximation) for point P_k, $1 \leq k \leq N$.

1) For each P_k, find $R(P_k)$.
2) The bit-string for each P_k will be the $R(P_k)^{th}$ bit-string in $BS(i)$.

Equi-population method sweeps the data points in a major pyramid in the chosen split dimension until the required number of points are found. Then the boundary values (i.e., M_1, M_2, M_3 in Figure 4(b)) of the split dimension are stored as the demarkation of the equivalence regions each of which corresponds to an approximation. While this technique takes into account the data distribution for a better performance, it also requires a large amount of storage for higher order bits.

C. Processing Angular Range Queries

An angular range query, defined in Section II-A, seeks all data similar to a given query within a given angular threshold α. To process such queries, we need to first identify the candidate approximations which intersect the conic query volume. The second step computes the actual results among the candidates.

Filtering Step For 2-dimensions, Figure 5(a) represents four underlying partitions (where A_1 and A_2 are in x_1 = 1 $major\ pyramid$, A_3 and A_4 are in x_2 = 1 $major\ pyramid$) and the unit square is the data space. The easiest way to decide whether an approximation intersects the range query space, is to look at the boundaries of the unit square which are not intersecting the origin. Here, these boundaries are the line segments from point $(1, 0)$ to $(1, 1)$ and from $(0, 1)$ to $(1, 1)$. Thus, finding the points K_1 and K_2 in Figure 5(b) will be sufficient to decide whether approximations A_1 and A_2 (in Figure 5(a)) intersect the query space or not. We only need to compare K_1 & K_2 with M & N.

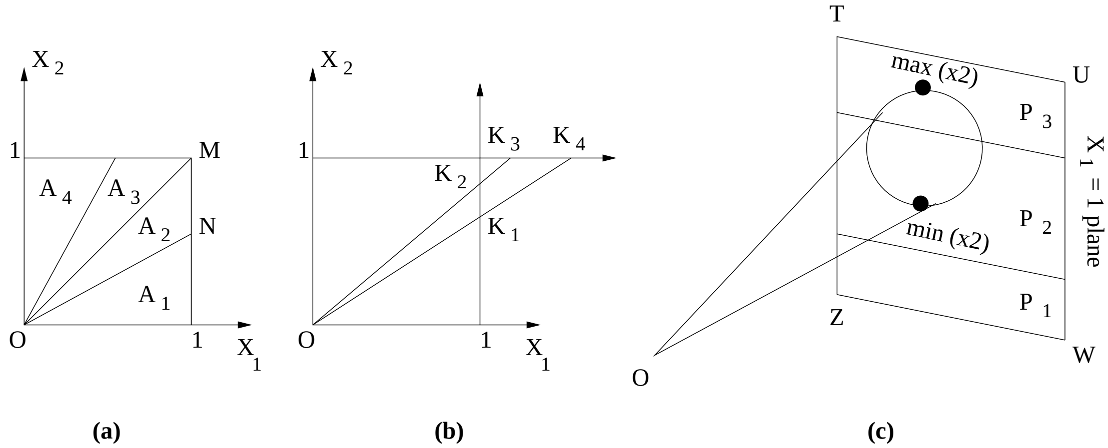


Fig. 5. Range query filtering approach

The query volumes will intersect all the infinite boundary planes that do not intersect the origin. Some intersections will be outside the unit hyper-cube. This can be used to eliminate some of the approximations. For 2-dimensions, the query in Figure 5(b) intersects both infinite boundary lines (i.e., $x_1 = 1$ $x_2 = 1$). However, the query does not intersect the $x_2 = 1$ line within the boundaries, i.e., K_3 and K_4 are outside the unit square. In this case, the approximations A_3 and A_4 are automatically eliminated. In 3-dimensions, in a similar case, major pyramid $x_2 = 1$ is totally eliminated.

However, if the query intersects a plane within the boundaries, then our goal is to find minimum (*min*) and maximum (*max*) values of the query on the boundary. For instance, in Figure 5(b) K_1 and K_2, in Figure 5(c) $max(x_2)$ and $min(x_2)$ will be such values.

Obtaining min and max values In order to find the ellipse-shaped intersection of the query on the $x_1 = 1$ plane, which is the base of the major pyramid, the equivalence surface equation (2) is used. For $x_1 = 1$, the closed form of the ellipse equation is

$$(u_1 + x_2u_2 + x_3u_3)^2 - (1 + x_2^2 + x_3^2)cos^2\alpha = 0. \tag{3}$$

Lagrange's multipliers approach is applied to the above equation. To maximize or minimize $f(p)$ subject to the constraint $g(p) = 0$, the following system of equations is solved.

$$\begin{aligned} \nabla f(p) &= \lambda \nabla g(p) \\ g(p) &= 0 \end{aligned} \tag{4}$$

To compute the extreme values for x_2 on $x_1 = 1$, take $f(x_1, ..., x_n) = x_2$, and $g(x_1, ..., x_n) = (u_1 + x_2u_2 + x_3u_3 + ... + x_nu_n)^2 - (1 + x_2^2 + x_3^2 + ... + x_n^2)cos^2\alpha$.

In order to compute the min-max values in a systematic and a fast manner, we arrange the equations for the min and max values as a linear system, i.e. $Ax = B$, where x is the solution set of the system, and A and B are coefficient matrices.

Identifying query results Once we have the min-max values, we can use them to retrieve the relevant approximations. These are the approximations in the specified α range neighborhood of the query. We earlier described two techniques for generating approximations - one based on equi-volume regions and the other on equi-populated regions of the major pyramid. We explain in this section how this design enables us to use these min-max values to effectively filter unrelated vectors.

For 3-dimensions, in Figure 5(c), $min(x_2)$ and $max(x_2)$ are the extreme values for dimension number 1 (i.e., for $x_1 = 1$ plane). In Figure 5(c), P_1, P_2 and P_3 represent the base rectangular planes of the corresponding sub-pyramids in $x_1 = 1$ major pyramid. We filter the approximation which is based on P_1 by utilizing $min(x_2)$ and $max(x_2)$ values.

In the general case, given the bounds (min_i, max_i) for each dimension i, the following algorithm computes the approximations (whose bases are on the $x_i = 1$ plane) we need. The algorithm filters the non-intersecting approximations.

Algorithm: Filter Approximations

Input: The extremes (min_i, max_i) for x_{i+1} on $x_i = 1$ plane. An empty set S_A.

1) For each approximation (a), if $min(a) \geq min_i$ and $max(a) \leq max_i$, then $S_A = S_A \cup \{a\}$. Here, $min(a)$ and $max(a)$ represent the minimum and maximum x_{i+1} values on the base of a.
2) If $min(a) \leq min_i \leq max(a)$ or $min(a) \leq max_i \leq max(a)$, then $S_A = S_A \cup \{a\}$.
3) The intersected approximations for $x_i = 1$ plane are now in S_A.

The previous algorithm retrieves the approximations intersecting with the angular range query space. However, some of the data points in these approximations might not be in the query space. We need to discard those data points. At this point, we start disk accesses.

Let $a_1, a_2 \ldots a_N$ denote the approximations in a major pyramid. Assume that $a_k, a_{k+1}, \ldots, a_{k+n-1}$ are the n approximations from this set which are identified as *intersecting* by the above algorithm. Note that they are physically consecutive, which means the partition of a_i comes physically between the partitions of a_{i-1} and a_{i+1}. We need to access all the candidate data points, and this fact is a motivation to sort the whole feature vectors once according to their vector approximations at the very beginning. Since they will be kept (in disk) in sorted order according to their approximations, I/O accesses of these points will be sequential, not random. Considering all the major pyramids that have candidate approximations in them, while processing a query, there will be at most d number of seek time for a d dimensional data space.

The performance can be further improved by applying the page access strategy proposed in [7]. Their strategy is not to access each candidate block using random I/O. Instead they keep reading sequentially if another candidate block is on the way. They read more pages sequentially than needed but eventually they beat the random-I/O-for-each-block approach. The same technique is applicable for our methods.

For the second pruning step, we need to compute the angular distance of every candidate point to the query point, and if a point is in the given range (α) then we output that point in the result set. The following algorithm is repeated for each major pyramid.

Algorithm: Identifying feature vectors in the given range for a major pyramid

Inputs: Set of intersected approximations for major pyramid i : $a_k, a_{k+1}, ..., a_{k+n-1}$. Angular range similarity parameters $Q(q_1, q_2, \ldots, q_d)$ and α. An empty set S_F.

1) a_f denotes the approximation of a vector f in $a_k, \ldots, a_{k+n-1}$. For each f, if $a_f \in \{a_k, \ldots, a_{k+n-1}\}$ and $\frac{\sum_{i=1}^{d} f_i q_i}{||f|| \cdot ||q||} \geq cos(\alpha)$, then $S_F = S_F \cup \{f\}$.
2) The resulting feature vectors in the given angular similarity range for $x_i = 1$ major pyramid are now in the set S_F.

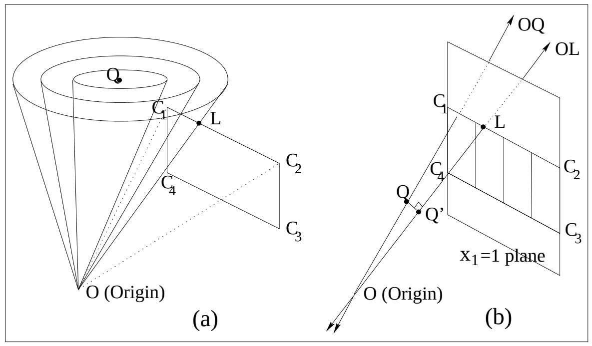


Fig. 6. Lowest angular distance calculation for an approximation

D. Processing Angular k-NN Queries

We now describe how to process k-NN queries using the AS-Q index. For filtering purposes, we will need to compute the lowest angular distance between a given query point and a pyramid. For a 3 dimensional visualization, Figure 6 represents an approximation which is based on the pyramid defined by the points $O(origin)$, C_1, C_2, C_3, C_4. Imagining the query space as a growing cone, at the very first time it touches the pyramid, we will get a line from the origin and along the lateral side of the cone, i.e., OL in Figure 6. The lowest angular distance from this pyramid to the query point (Q) is the angle between the lines defined by OQ and OL.

However, if L is not between C_1 and C_2, then we call it *out of bounds*. In this case, one of the corner points, C_1, ..., C_4, will be the point on the pyramid that makes the lowest angular distance, i.e., the angle defined by OQ and OC_1 will be less than the angle defined by OQ and OL.

For the 3 dimensional case, only the third dimensions of the points L, C_1, C_2 will be different, i.e., $C_1 = (1, x_2, 1)$, $C_2 = (1, x_2, 0)$ and $L = (1, x_2, z)$. The only unknown will be z, and if $0 \leq z \leq 1$, then L is *in the bounds* as in Figure 6. In this case, we calculate the angular distance between OQ and QL and give the angle as the lowest angular distance bound for the current approximation. Otherwise, we use the corner points for lowest bound calculation (i.e., C_1, ..., C_4 instead of L).

k-NN Filtering Having the lowest angular distances for the approximations, the next step is to use these values in filtering. For a given query point, we first find the approximation the query point is in. Naturally, the lowest possible angular distance from the query point to this approximation will be zero. We retrieve all the feature vectors in this approximation from disk and insert the k closest of them as the candidates in a list that we call $NNlist$ in nondecreasing order. Then, for the remaining approximations, we consider the lowest angular distances from the query point without retrieving any more feature vectors from the disk.

In-memory Filtering At this step we prune the approximations which have lowest angular distances greater than the k^{th} value in $NNlist$ we found so far. Then, we sort the remaining approximations according to their lowest values in nondecreasing order. At this moment, the second filtering step starts.

Filtering in Disk We retrieve the first approximation in the sorted order, and retrieve the feature vectors (points) in this approximation from the disk. If a retrieved point is closer than the k^{th} closest point in the $NNlist$, then we update $NNlist$, i.e., remove the previous k^{th} value with the new one and sort the $NNlist$ again. We repeat this updating step for all the feature vectors in the current approximation. After that, if the new k^{th} value is less than the lowest value of the next approximation in the sorted order, then we stop and return our $NNlist$ as the result of the k-NN search. Otherwise, we move on to this next approximation and repeat the same process until we stop.

Algorithm: k-Nearest Neighbor for query Q. The approximation (subpyramid) of Q is a_q. AD is abbreviation for *angular distance*.

1) Retrieve vectors in a_q from disk, keep k closest of them in $NNlist$ in nondecreasing order. $NNlist(i)$ is the i^{th} closest vector, $distance(\ NNlist(i))$ is AD of the i^{th} closest vector.
2) Find lowest angular distances from Q to the remaining approximations as described in Figure 6. $lowest(i)$ is the lowest AD of approximation i.
3) If $lowest(i) > distance(NNlist(k))$ prune i . Repeat for all approximations.
4) Sort candidates according to $lowest(i)$ values. $c(i)$ is the i^{th} candidate in sorted order.
5) Retrieve the vectors in $c(1)$ from the disk. If a vector has less AD than $distance(NNlist(k))$, update $NNlist$.
6) STOP if $distance(NNlist(k)) <= lowest(c(2))$. Otherwise, retrieve the vectors in $c(2)$ and process them as the previous step and repeat for the next $c(i)$ until we STOP.
7) The k-nearest neighbors will be in $NNlist$.

IV. Cone-shell Quantizer (CS-Q)

We now propose a second quantization based structure, Cone-shell Quantizer (CS-Q), which uses cone partitions, rather than pyramids, and is organized as shells, instead of the sweep approach followed by AS-Q. CS-Q is a variation of AS-Q and shares many of its fundamental algorithms. The underlying partitioning of the quantizer is shown in Figure 7(a). The axis for each of the cone-shells is the line from the origin to a reference point, i.e., OR. We have chosen the reference point as $R(0.5$, 0.5, ... , $0.5)$ which gives statistically better results. Figure 7(b) represents a cross-section of the cone-shells and an angular range query cone.

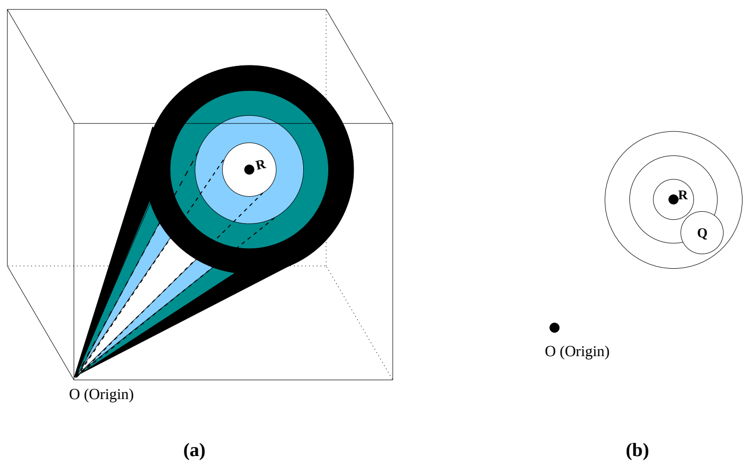


Fig. 7. Underlying partitions for CS-Q technique

As in AS-Q, we can follow an equi-volume or an equi-population based structure. Here, we only present the equi-populated one. The algorithm is as follows.

Angular Approximations based on Equal Populations, N is the number of data points, R is the reference point, S_a is the set of all approximations, and $S_a(i)$ is the i^{th} approximation.

1) For each data point P_k, $1 \leq k \leq N$, calculate the angular distance between P_k and R.
2) Sort the data points in nondecreasing order based on their angular distances to R.
3) Assume t is the given population for each approximation. Assign the first t number of points in sorted order to $S_a(1)$, the second t number of points to $S_a(2)$, and so on.

Equi-volume based structure is trivial, i.e., the only constraint for the cone-shells (from center to the out) in Figure 7(a) is to have angles of β, 2β, 3β, ... between their lateral surfaces and R.

The bit allocation scheme is as follows.

Total number of bits b, and $|S_a|$ is the total number of approximations.

1) Generate the approximations as described in the previous algorithm.

2) For each approximation i, assign the next $\frac{2^b \times i}{|S_a|}$ bit-strings to $S_a(i)$.

A. Query Processing

Angular range queries are handled similar to AS-Q. The difference lies in intersection formulas in the filtering step. Figure 8 shows two cases for the range query with CS-Q technique. Q is the given query point, the angle $Q\hat{O}K$ is α which is the given range, and R is the reference point. Let's denote the angle $R\hat{O}Q$ as θ. The lowest angular distance from shell i to R is l_i and the largest angular distance is u_i. Figure 8(a) represents the case for $\theta > \alpha$ and Figure 8(b) is for $\theta < \alpha$. The conditions for approximation (shell) i to intersect the query space are given as follows.

1) For ($\theta > \alpha$), if ($u_i \geq \theta - \alpha$) and ($l_i \leq \theta + \alpha$) then i is an intersecting approximation.

2) For ($\theta < \alpha$), if ($\theta + \alpha \geq l_i$) then i is an intersecting approximation.

Next the data points in these intersecting approximations are retrieved and checked. The remaining nonintersecting approximations are filtered out as in Section III-C. The main difference is, since there are no min and max values needed in this computation, the Lagrange multipliers are not utilized in this design.

Angular k-NN query with CS-Q is again similar to AS-Q. The only difference is the way to calculate the lower and upper angular distances from the query point, Q, to an approximation i.

1) For ($\theta > u_i$), the lowest angular distance is $(\theta - u_i)$, and largest distance is $(\theta + u_i)$.

2) For ($\theta < u_i$), the lowest angular distance is $(l_i - \theta)$, and largest distance is $(\theta + u_i)$.

The remainder of the algorithm is same as in Section III-D.

V. Experimental Results

This section summarizes the results of our experiments on the performance of the proposed access structures. We used 6 data sets, two synthetic and four real data sets from text, time-series, and image

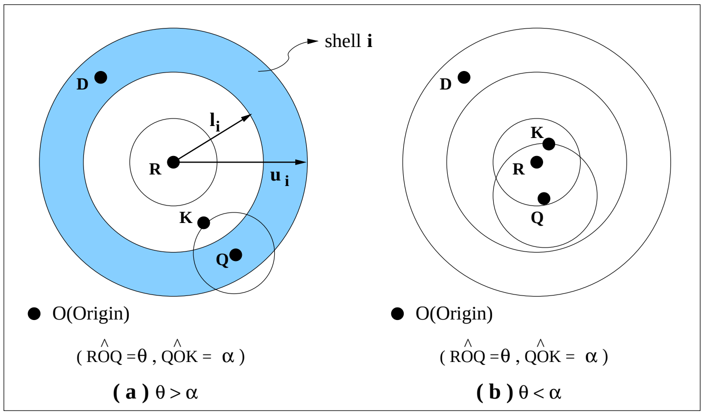


Fig. 8. Range Query with CS-Q

database applications where angular similarity is widely used. We generated synthetic data sets with *Uniform* and *Gaussian* distributions for dimensions 16 and 32. The real-life data sets are Satellite Imagery Data (*Landsat*), Newsgroups (*NG*), National Science Foundation abstract repository (*NSF*), and *Stock*.

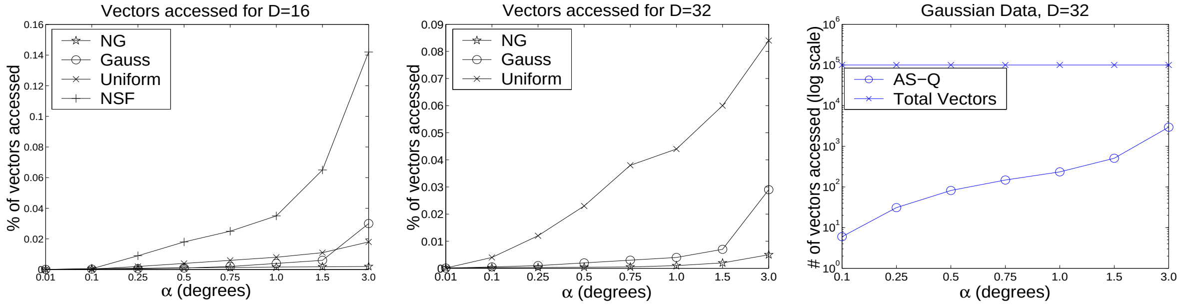
Landsat data, Satellite Image Texture, consists of 100,000 vectors representing 32 dimensional texture features of Landsat images [26]. The *NG* data set is a compilation of 20,000 Usenet postings in 20 different categories like religion, politics and sports. Another data set is a collection of abstracts describing *NSF* awards for basic research. We performed Latent Semantic Reduction on these text documents, and stored the products of the term frequency and the inverse document frequency (tf/idf) [33] for the terms occurring in each document, after eliminating the stop words. We applied SVD (Singular Value Decomposition) [22] over the term vector representations, and generated 8, 12, ..., 32 dimensional representations of the text documents. The *Stock* data, which is very skewed, is a time series data set which contain 16 (or 32) days (dimensions) stock price movement of 6500 different companies. In addition to these 6, we also produced 5 different data sets each having a different distribution and we talk about them in Section V-D.

We present performance results of angular range and angular k-NN queries. We compare the angular range and k-NN search performances of the two proposed techniques with respect to each other and to the VA-File approach. On each of the data sets mentioned above, we perform experiments for a variety of range queries such as 0.25, 0.50, ... , 3.0 degrees of angular ranges. For a measure of perspective, for the 32 dimensional Uniform data set, 1.0 degree corresponds to a selectivity of 0.01 and 5 degrees correspond to a selectivity of 0.1. This gives an insight into the performance for nearest neighbor queries on the same data sets, i.e., a range query of 1 degree would correspond to a k-NN query, where k is 1% of the data size. For each data set experiment, we choose 200 totally random query points from the data set itself and present the results as averages.

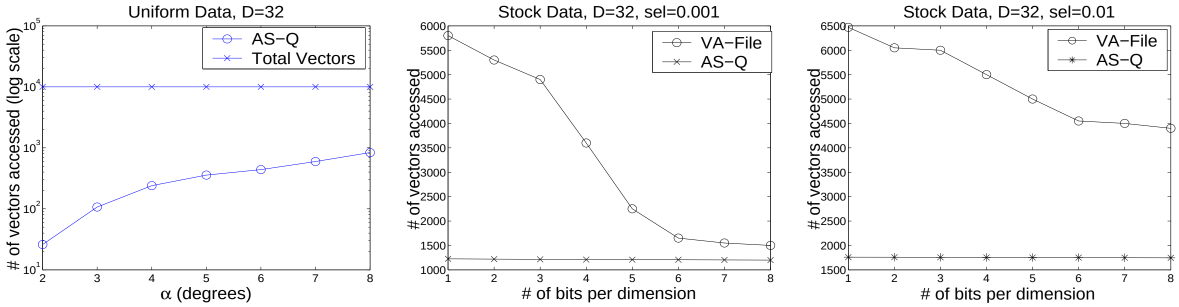
A. AS-Q Results

In Figure 9, we present the results for AS-Q. Figures 9(a) and 9(b) show the percentage of vectors retrieved by AS-Q technique (as α increases) for Uniform, Gaussian, NSF and NG data sets of dimensionality 16 and 32, and for reduced 64 bit representations of data objects. In Figures 9(c) and 9(d), we show the results for the number of vectors accessed compared to the total number of vectors (as α increases) for 32 dimensional Gaussian and Uniform data. Finally, in Figures 9(e) and 9(f), we present the number of vectors accessed by AS-Q technique compared to the VA-File as the number of bits per dimension increases, for 32 dimensional Stock data and for α=0.25 and α=1.5. Both because of the geometric mismatch and extreme skewness of the stock data, the query intersects most of the grid partitions of the VA-file, especially for small number of bits. Since the queries were chosen from the data set, they are also skewed. Even a relatively small cone around the queries is likely to intersect large rectangular partitions. The number of such intersections reduces as more number of bits is used for quantization.

None of the current techniques provide a direct solution to execute angular queries. We compare our results with an adapted version of the conventional VA-File based approach. The rectangular regions that intersect the angular query space is computationally hard to find. To enable an experimental comparison, we consider an approximation in VA-File to be intersecting if its representative value is in the angular neighborhood of $(\alpha + \epsilon)$, where ϵ is a derived parameter that guarantees the correctness of the results. We fix the geometric center of the regions (i.e., a rectangle in VA-File) as the representative value of the approximations for our experiments. An ϵ-space is needed to ensure that we do not miss those approximations that actually intersect but whose center is not in the angular similarity space. This



(a) *% of vectors accessed for D=16* (b) *% of vectors accessed for D=32* (c) *Gaussian32, bits=32*



(d) *Uniform32, bits=32* (e) *Stock32,* α *= 0.25* (f) *Stock32,* α *= 1.5*

Fig. 9. Performance of AS-Q for angular range query. D (dimensions) and α (range)

approach works well where the cell regions describing the approximations are small enough. After finding the intersecting approximations in this way, we look at the corresponding feature vectors and return the ones that are in the given similarity range.

For angular range queries, we observe that AS-Q technique outperforms the VA-File significantly. Even for very low selectivities, it is possible to filter a reasonable number of approximations using the VA-File approach, however the AS-Q approach performs much better. For instance, for the Stock data set, for an angular range query of 1.5 degrees, the number of vectors visited for AS-Q is 1741, while it is 4370 for the VA-File, where both approaches use bit-strings of 8 bits per dimension.

In other data sets, we observe a similar performance improvement and they follow similar patterns. For instance, in VA-File the number of vectors visited for a range query of 3.0 degrees for the NG data set of dimensionality 16 is 6157, whereas for AS-Q technique it is just 79, which corresponds to a speedup of 77 times. For 32 dimensions, the speedup is 97 times. For a range query of 0.25 degrees, which corresponds to a selectivity of roughly 0.0001, the number of vectors visited for AS-Q is 2, while it is 6 for VA-File. It is also important to note that the same performance is achieved by AS-Q with a 32-bit representation of the data, while the VA-File method would require a 256-bit representation of the data for 100% recall. Additionally, in the NSF data, for a selectivity of 0.001, the number of approximations after filtering is 745 for VA-File and 156 for our technique.

TABLE I

K-NN RESULTS FOR UNIFORM DATA

10-NN	Step 1	Step 2	total pruned	out of	**50-NN**	Step 1	Step 2	total pruned	out of
6 bits	22	23	45	48	6 bits	19	26	45	48
7 bits	26	67	93	96	7 bits	32	59	91	96
8 bits	67	122	189	192	8 bits	52	136	188	192

TABLE II

K-NN RESULTS FOR NG REAL DATA

10-NN	Step 1	Step 2	total pruned	out of	**50-NN**	Step 1	Step 2	total pruned	out of
7 bits	74	24	98	101	7 bits	53	44	97	101
8 bits	157	42	199	202	8 bits	137	61	198	202
9 bits	251	81	332	335	9 bits	174	156	330	335

The results of angular k-NN search are similar to angular range query. Table I presents the k-NN results for a synthetic data set which has equi-volumed and equi-populated approximations and also equal number of approximations per dimension. We used 7, 8, 9 bits and we present the averages for 10-NN and 50-NN queries. The numbers in the *Step 1* column represents the number of approximations that are filtered in the first step of the pruning algorithm. Similarly, the numbers in the *Step 2* column represents the number of approximations that are filtered in the second step. The column *total filtered* is the total number of approximations that are totally filtered in the first and second steps, i.e., the summation of the *Step 1* and *Step 2* columns. The last column, namely *out of*, is the actual total number of approximations in the system.

In Table I, i.e., for 10-NN and 8 bits, 67 out of 192 approximations are filtered at the first step. If there are $10,000$ feature vectors in the data set, there will be approximately 50 feature vectors in an approximation. This means, $50 \times 67 = 3,350$ feature vectors out of 10,000 are pruned in the first step. Similarly, 122 out of 192 approximations are filtered in the second step, and this means, $50 \times 122 = 6100$ feature vectors out of 10,000 are pruned in the second step. Similarly, Table II presents the experiment results for k-NN similarity search approach for NG real data set. Tables I and II reveal the effectiveness of our approach not only for very narrow queries but also for wider queries (i.e., 50-NN) as well.

B. CS-Q Results

Figure 10 presents the results for CS-Q technique for different data sets. Figure 10(a) represents the results for an average selectivity of 0.00025, e.g., 25 results out of 100,000 data points. This selectivity is the average taken for all the dimensions presented in the graph, i.e., 8, 12, ..., 32. In this figure, for 8 dimensions, 1740 vectors are accessed by the CS-Q technique and the number of vectors retrieved increases as the dimensionality increases for NG data. Figures 10(c), 10(d), 10(e), and 10(f) show the number and % of vectors accessed by CS-Q as the number of bits increases and as alpha increases.

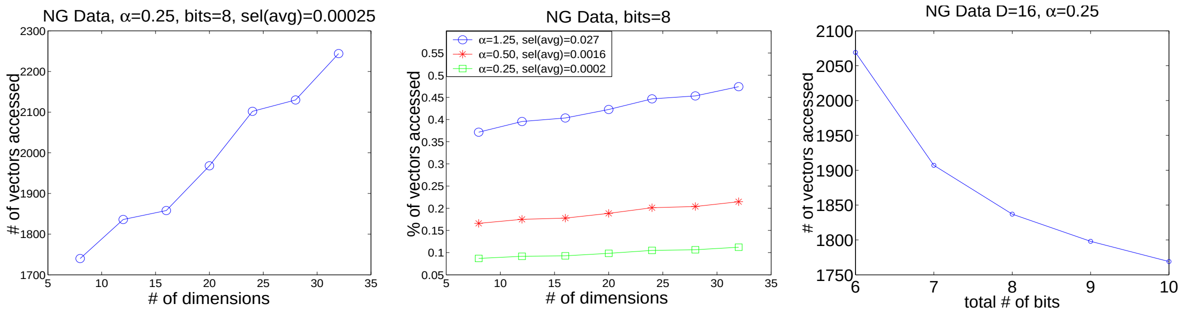
TABLE III

OF VECTORS ACCESSED FOR NG16 & NG32

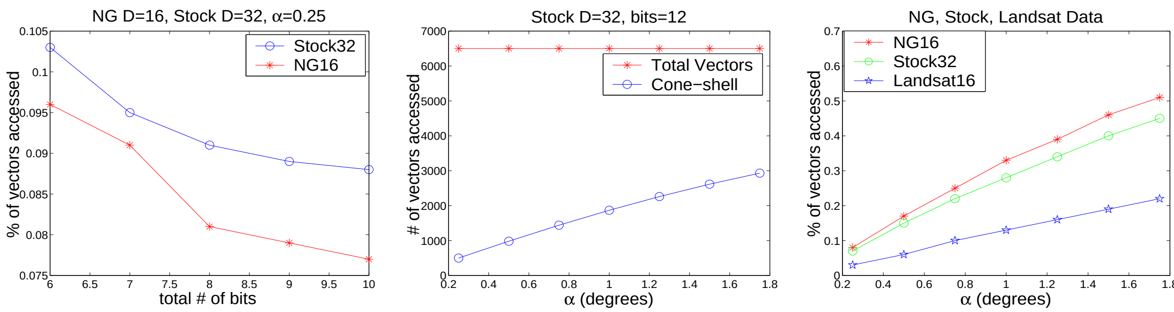
NG16	CS-Q 11bits	CS-Q 16bits	VA-File 16bits	**NG32**	CS-Q 11bits	CS-Q 32bits	VA-File 32bits
α=0.25	1769	1759	7856	α=0.25	2154	2144	7373
α=0.50	3470	3460	10281	α=0.50	4207	4197	9165
α=0.75	5094	5084	12069	α=0.75	6119	6110	11786
α=1.00	6603	6593	13968	α=1.00	7862	7852	13943
α=1.50	9245	9237	15793	α=1.50	10752	10744	16202
α=3.00	14260	14259	18276	α=3.00	15519	15513	18697

C. AS-Q vs. CS-Q

We compare the CS-Q and VA-File results in Table III. For 16 dimensional NG data and for 16 bits, CS-Q outperforms the VA-File approach. For example, for $alpha = 0.25$, VA-File accesses 7856



(a) *# of vectors accessed vs # of dimensions* (b) *% of vectors accessed vs # of dimensions* (c) *# of vectors accessed vs bits*



(d) *% of vectors accessed vs bits* (e) *# of vectors accessed vs α* (f) *% of vectors accessed vs α*

Fig. 10. Performance of CS-Q for angular range query. D (dimensions) and α (range)

data points while CS-Q retrieves only 1759. One interesting property of CS-Q is, it achieves a better performance than VA-File with much less number of bits than VA-File requires. That's why we put the *11bits* column in Table III. For $\alpha = 0.25$ in Table III, CS-Q accesses 1769 points for only 11 bits which is again a much better performance than VA-File achieves. The results for 32 dimensional NG data are similar.

As a performance comparison between the two proposed techniques, the CS-Q technique achieves better results than AS-Q when we use *small and equal* number of total bits for both of them. For example, for 32 dimensional Stock data, $alpha = 0.25$, and for 1 bit per dimension which makes totally 32 bits per data point, the AS-Q retrieves around 1200 data points. However, in Figure 10(e), for the same data and angle, the CS-Q retrieves 499 points with only 12 bits. Thus, CS-Q achieves better performance results than AS-Q with less number of bits (12 vs 32).

Table IV presents another interesting property between the two proposed quantization techniques.

TABLE IV

OF VECTORS ACCESSED FOR STOCK16

Stock16	α=1.50	α=3.00
AS-Q(32bits)	1704	1753
CS-Q(12bits)	2922	4336
CS-Q(32bits)	2920	4335
VA-File(32bits)	5816	6011

In this case, AS-Q seems to have a better result, i.e., 1704 for AS-Q and 2920 for CS-Q. This is because after some number of bits, the CS-Q approach's performance only increases very slowly as we increase the total bits. For example, when there are only a few data points per bit-string, decreasing the volumes of underlying partitioning by increasing the total number of bits will not continue to distribute the data points to many bit-strings. In other words, after some number of bits, the data points per bit-string will not change so that the filtering will not continue to prune many points. For example, in Table IV, increasing bits from 12 to 32 will only decrease the number of vectors accessed from 2922 to 2920 for CS-Q. The two techniques both perform better than VA-File.

In general, if there is enough memory space available, AS-Q technique achieves better results. The drawback of CS-Q is, it needs to check some points that are away from the query space if their approximations intersects the query, i.e., point D in Figure 8. For higher number of bits, AS-Q can distribute the points into different bit-strings that are convenient for pruning and that's why it performs better. On the other hand, for lower number of bits (when the resources are limited), the CS-Q performs better.

D. Results on Scalability

In order to investigate the scalability issue, we produced 5 different 16-dimensional synthetic data, each of them in different distributions. For each distribution there are 4 files that have 1K, 10K, 100K, and 1000K number of data points. Our experimental setup for the scalability tests differs from our other setups. We take every 500^{th} data point as a query point, run the queries and present the average over

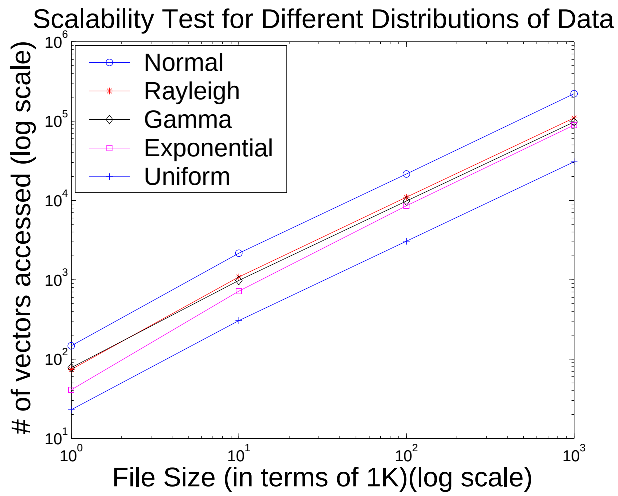


Fig. 11. Scalability Test Results, $\alpha = 0.25$

these queries as the result. That means we have 2000 queries from a 1000K file, and 200 queries from a 100K file. We kept the range parameter constant, i.e., $\alpha = 0.25$. The results are depicted in Figure 11 and they indicate that our approach is linear to the number of data points in a data set.

E. Comparison with Sequential Scan

An important property of our techniques is that we keep the data points in sorted order in disk according to their physical partitions. We have the discussion about this issue for AS-Q in Section III-C. For CS-Q approach, we have a better outcome for the query processing in terms of the candidate approximations. The data vectors are kept in the order of their angular distances from the reference point, and since any query space (Figure 7(b)) has to intersect only consecutive approximations, the accessed vectors will always be in physically consecutive approximations. Thus, by keeping these vectors in consecutive disk blocks, the second step (disk access) of the query processing will require only one seek time and the consecutive blocks are accessed in sequential order.

In order to validate our discussion, we compared our approach with sequential scan. We followed the same experimental setup and used the same data sets as in Section V-D. We calculated wall clock time results as a function of the data set size. For the Exponential data, with 10K number of points, the total time for sequential scan is 237 msec, whereas our technique only takes 19.7 msec. The difference gets higher as the number of data points increases. For instance, for 100K number of points, sequential scan time is 2294 msec, and our approach's time is 195 msec. For 1000K, the comparison [1] is 21180 vs. 2070 msec. For the Uniform data, the values are 222 vs. 9.2 msec for 10K. In addition, for 100K and 1000K points, the results are 2035 vs. 62, and 19397 vs. 616 msec. We also made experiments for the other distributions, and the comparisons follow very similar paths. Thus, we conclude that our technique performs significantly better than the sequential scan in terms of time.

F. Results on Normalized Data

We now summarize our experimental results on the performance of indexing the normalized data: We set the norm of all the vectors to unit length to be able to utilize the Euclidean space based indices. SDSS SkyServer [36] follows a similar approach and develops a 3-dimensional quad tree over the normalized data. Instead of an underlying partitioning that considers the angular nature of the data, the normalization is used to transform the data into an Euclidean space. Besides being limited to three dimensions, and having the geometric mismatch, our experiments also showed that the normalization introduces additional problems for Euclidean distance based index structures. We have repeated our previous experiments with normalized data both for VA-files and for the proposed techniques. In Table V, for 32 dimensional Stock data and for 32 bits, the CS-Q retrieves 499 points while the VA-File accesses 4861. For normalized data, the CS-Q again achieves the same number of points (499) but VA-File accesses 6415 which is worse than the non-normalized case. Since the partitioning behind the quantization of our techniques considers an angular organization of the data, the bit-strings of data points do not change when the data is normalized. The data points map onto a hyper-sphere after normalization and this mapping is along the line between the original data point and the origin. On the other hand, from the perspective of Euclidean distance based index structures, the normalization causes the data points become closer to each other and are harder to separate by the index. This causes degradation in the performance of the traditional index structures, including VA-files.

[1] From now on, first number refers to the sequential scan time and the second number refers to our approach.

TABLE V

NORMALIZED VS NON-NORMALIZED, ALPHA=0.25

Stock32	CS-Q(32bits)	VA-File(32bits)
Nonnormalized	499	4861
Normalized	499	6415

VI. CONCLUSION

We studied the problem of efficient execution of similarity queries that are based on angular measures. Although, angular measures have been popularly utilized by several important applications, such as information retrieval and time-series data repositories, we are not aware of any indexing and query processing technique for efficient execution of such queries. We first sought to overcome the geometric mismatch between the underlying partitions of the conventional indexing techniques, and built index structures that are better suited to angular queries. We developed two scalar quantization methods, one is based on pyramid partitions angularly sweeping the data space, and the other is based on cone partitions that are organized as shells. The quantized approximations are used as a dense index for efficient execution of similarity queries. Each technique has its own unique strength. CS-Q technique achieves better performance under limited memory conditions, i.e., for lower quota of bits. For a higher number of bits, AS-Q becomes more successful in distributing the data objects into separate partitions which enables more accurate summaries. We compared the proposed techniques with an adaptation of VA-files. In all cases, the proposed techniques perform significantly better than VA-Files.

We have focused on optimizing query performance. An efficient update routine is necessary if there are frequent and non-stationary changes over the data, and if sequential access is desired. If we want to optimize insertions, we could just add the new items to the end of the file, with no changes in our algorithms/structures. Clearly, this will introduce random I/Os. If one wants to avoid random I/Os and guarantee sequential access to the data, one can use the common storage approaches such as keeping empty space in blocks, using overflow files for insertions, and logical markers for deletions.

We chose to apply quantization as the last step of our techniques because of its well-known scalability with higher dimensions. However, the proposed techniques can be applied to different classes of access structures, e.g., indices based on data-space partitioning, tree structures, or the Pyramid-technique. For example, the proposed angular structure can be used to map the data objects into 1-dimension which is then indexed by a B+tree (as in Pyramid-technique [9]).

The use of an effective structure and algorithms for angular similarity searches will impact the practical performance of several applications. We plan to extend our work to include angular similarity joins, which are popularly used, among others, in financial market applications.

REFERENCES

[1] R. K. Ando. Latent semantic space: Iterative scaling improves precision of inter-document similarity measurement. In *Proceedings of the 23rd SIGIR*, pages 216–223, 2000.

[2] M. Andrec, P. Du, and R. M. Levy. Protein structural motif recognition via nmr residual dipolar couplings. *J. Am. Chem. Soc.*, 123:1222–1229, 2001.

[3] D. Androutsos, K. Plataniotis, and A. Venetsanopoulos. A novel vector-based approach to color image retrieval using a vector angular-based distance measure. *Computer Vision and Image Understanding*, 75(1/2):46–58, Aug. 1999.

[4] A. Baruffolo. R-trees for astronomical data indexing. *ASP Conf. Ser., Astronomical Data Analysis Software and Systems VIII*, 172:375, 1999.

[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, H. Jagadish, H. Kriegel, and J. Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proc. 16th Int. Conf. on Data Engineering*, pages 577–588, San Diego, CA, 2000.

[7] S. Berchtold, C. Bohm, H. V. Jagadish, H.-P. Kriegel, and J. Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *ICDE*, pages 577–588, 2000.

[8] 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.

[9] 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.

[10] P. V. Biron and D. H. Kraft. New methods for relevance feedback: Improving information retrieval performance. In *ACM Symposium on Applied Computing*, pages 482–487, 1995.

[11] I. Choi, J. Kwon, and S. Kim. Local feature frequency profile: A method to measure structural similarity in proteins. *PNAS*, 101:3797–3802, Mar. 2004.

[12] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric space. In *Proc. of the 23rd VLDB Conference, Athens, Greece*, pages 426–435, 1997.

[13] A. Debelack, J. Dehn, L. Muchinsky, and D. Smith. Next generation air traffic control automation. *IBM Syst. J. 24*, 1:63–77, 1995.

[14] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[15] D.Hull. Improving text retrieval for the routing problem using latent semantic indexing. In *Proc. of the 17th ACM-SIGIR Conference*, pages 282–291, 1994.

[16] M. P. T. Doom and M. Raymer. Ga-facilitated knowledge discovery and pattern recognition optimization applied to the biochemistry of protein solvation. In *Genetic and Evolutionary Computation Conference*, pages 426–437, May 2004.

[17] A. Dumitras and A. N. Venetsanopoulos. Angular map-driven snakes with application to object shape description in color images. *IEEE Trans. Image Process*, 10(12):1851–1859, Dec. 2001.

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

[19] H. Ferhatosmanoglu, D. Agrawal, and A. E. Abbadi. Efficient processing of conical queries. In *CIKM*, pages 1–8, 2001.

[20] W. M. G. Introduction to statistics for bioinformatics, with applications to expression microarrays. *CSB Tutorial sessions*, 2003.

[21] 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.

[22] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. pages 289–300, 1997.

[23] P. Kostiuk, M. Adams, D. Allinger, G. Rosch, and J. Kuchar. An integrated safety analysis methodology for emerging air transport technologies. *NASA/CR-1998-207661, pp. 66*, Apr. 1998.

[24] LEDAS. Leicester database and archive service. http://ledas-www.star.le.ac.uk/, 2004.

[25] P. O. B. M. B. Eisen, P. T. Spellman and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *PNAS*, 95:14863–14868, Dec. 1998.

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

[27] Z. D. Michael W. Berry and E. R. Jessup. Matrices, vector spaces and information retrieval. In *SIAM Review, Vol.41, No. 2*, pages 335–362, 1999.

[28] J. Nievergelt, H. Hans, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.

[29] D. K. P. E. Trahanias and A. N. Venetsanopoulos. Directional processing of color images: Theory and experimental results. *IEEE Trans. Image Process*, 5(6):868–880, Oct. 1996.

[30] G. Qian, S. Sural, Y. Gu, and S. Pramanik. Similarity between euclidean and cosine angle distance for nearest neighbor queries. In *ACM symposium on Applied computing*, pages 1232 – 1237, Nicosia, Cyprus, 2004.

[31] A. Razdan. Wavelet correlation coefficient of strongly correlated financial time series. *arXiv:cond-mat*, 2, Oct. 2003.

[32] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The a-tree: An index structure for high-dimensional spaces using relative approximation. In *Proc. of 26th Int. Conf. on VLDB, September 10-14, 2000*, pages 516–526, Cairo, Egypt, 2000.

[33] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[34] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24(4):35–43, 2001.

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

[36] A. S. Szalay, J. Gray, and A. R. Thakar. The sdss skyserver: public access to the sloan digital sky server data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 570–581, 2002.

[37] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *ACM HotNets-I*, October 2002.

[38] P. E. Trahanias and A. N.Venetsanopoulos. Vector directional filters: Anew class of multichannel image processing filters. *IEEE Trans. Image Process*, 2(4):528–534, Oct. 1993.

[39] D. R. Warn. Lighting controls for synthetic images. *Computer Graphics*, 17(3):13–21, 1983.

[40] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. of the Int. Conf. on VLDB*, pages 194–205, New York City, New York, Aug. 1998.

[41] A. Wicenec and M. Albrecht. Methods for structuring and searching very large catalogs. *ASP Conf. Ser., Astronomical Data Analysis Software and Systems VII*, 145:512, 1998.

[42] R. Wilkinson and P. Hingston. Using cosine distance in a neural network for document retrieval. In *ACM SIGIR*, pages 202–10, 1991.

[43] P. Zhang, Y. Huang, S. Shekhar, and V. Kumar. Correlation analysis of spatial time series datasets: A filter-and-refine approach. In *7th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, 2003.

[44] P. Zhang, Y. Huang, S. Shekhar, and V. Kumar. Exploiting spatial autocorrelation to efficiently process correlation-based similarity queries. In *SSTD*, Santorini Island, Greece, July 2003.