Efficient Processing of Conical Queries

Hakan Ferhatosmanoglu, Divyakant Agrawal, Amr El Abbadi
Department of Computer Science
University of California
Santa Barbara, CA 93106
{hakan,agrawal,amr}@cs.ucsb.edu

Abstract

Conical queries are a novel type of query with an increasing number of applications. Traditional index structures and retrieval mechanisms, in general, have been optimized for rectangular and circular queries, rather than conical queries. In this paper, we focus on conical queries which can be defined as a multi-dimensional cone in a multi-dimensional data space. We develop a model for expressing such queries and suggest efficient techniques for evaluating them. In particular, we explore the retrieval problem in the context of conical query processing and develop multi-disk allocation methods specifically for processing conical queries.

1 Introduction

Recent developments in technology have introduced different types of multimedia data into modern applications. Examples of such applications include multimedia information systems [16, 31, 32], astronomical [35, 1, 23], geographical [10], and document [32, 12, 14, 13] data repositories. The data objects in these applications are typically represented by multi-dimensional feature vectors. A similarity function between these feature vectors is defined and is used as a measure for the similarity between the corresponding data objects. Several indexing techniques have been proposed for multidimensional Euclidean spaces [20, 29, 28, 25, 21, 2, 34, 24, 5, 4]. Most of these structures are optimized specifically for point, range, and nearest neighbor queries with Euclidean distance as the similarity metric between feature vectors.

Conical queries are a novel type of query with an increasing number of applications. Traditional index structures and retrieval mechanisms, in general, have been optimized for rectangular and circular queries, rather than conical queries. In this paper, we focus on conical queries which can be defined as a multi-dimensional cone in a multi-dimensional data space of items. Conical queries occur in a variety of different applications. One such example is in Air Traffic Control (ATC) [11, 22]. A conical query in an ATC is to find all objects within the flight route of the plane. 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. A similar conical query can be defined in a Space Transportation System [22] to check the objects, e.g., satellites, within the the route of a spacecraft.

A conical query can be created by asking the similarity between feature vectors using the *angular distance* as the similarity measure. The cosine angle distance is effectively used in various applications to represent the similarity/dissimilarity between feature vector representations of data. It is used extensively in the context of astronomical, geographical, and document data repositories. In an astronomical data repository [1, 23], a conical query can specify an object and an angle and ask for all objects, e.g., all stars, that have an angular distance, with the given object, less than the given angle. In a geographical or astronomical database, a conical query selects all database entries whose coordinates lie within a specified angular distance of the input longitude and latitude [23]. In document databases [32], documents are compared in the semantic space by comparing their multi-dimensional representations created by statistical analysis. For example, after ignoring stop words, e.g., articles, prepositions, etc., the number of occurrences of keywords are computed and are stored in a single feature vector for each document. Since the table of feature vectors of documents may be very large, techniques such as singular valued decomposition (SVD) are used to reduce the dimensionality of the vectors. Similarity between documents is measured by the cosine of the angle between feature vectors [32]. Latent Semantic Analysis (LSA) is a well-known example of such applications [12, 14, 13].

The cosine angle metric computes the difference in direction, irrespective of vector lengths, where the distance is given by the angle between the two vectors (An illustrative example is in Appendix). In fact, the cosine distance metric is closely related to the Euclidean distance metric. However, depending on the application, there may be cases where one metric is preferred over the others. More detailed examples and discussions about the applications of cosine angle distance can be found at [36, 32]. As we will show later, cosine distance can be transformed into Euclidean distance. The geometries of the shapes created by these transformations will be *conic* regions in multi-dimensional space. Therefore, it is actually possible to develop query processing techniques using current index structures based on rectangular or spherical page organization of the data points. These standard techniques are not suitable for conical queries due to the *geometric mismatch*. For example, index structures based on R-trees usually use minimum bounding rectangles [21, 2], or circles [34], as underlying geometric shapes. These structures are shown to be effective, in terms of I/O cost, for rectangular and circular queries. However, there is a mismatch between the geometry of conical queries and the geometry underlying the design of current index structures. It is likely that traditional approaches will end up retrieving almost all the pages. Efficient processing and indexing techniques developed for conical queries, which are specifically focused to reduce the I/O cost of such queries, are needed.

In this paper, we first formally define *conical queries*, as they correspond to the shape of a conic in the geometrical spaces. We develop the model for expressing such queries and develop efficient retrieval techniques for evaluating them. The technique proposed in this paper is specifically optimized for conical queries. To the best of our knowledge, there is no such work optimized for conical queries. Since typical applications that involve high-dimensional data have large amounts of data, it is also necessary to optimize the retrieval for multi-disk architectures. We develop disk and page allocation methods for the storage of multi-dimensional data to efficiently process conical queries. We show that the resulting techniques are very efficient both for single and multi-disk architectures.

Section 2 formally defines conical queries and Section 3 introduces the data space partitioning technique to organize the data points. In Section 4, we show how to process conical queries by first identifying the conical region within the data space and then identifying the partitions to be retrieved. We also discuss the layout of the partitions in a single disk for efficient processing. Section 5 develops the layout for multi-disk architectures. We evaluate the performance of the proposed technique in Section 6. Section 7 concludes the paper with a discussion.

2 Conical Queries

A conical query can be defined as all points in the data space that lie within a region defined by an angle with respect to the origin. It can be described by the pair (Q, α), where Q denotes the query point $(q_1, q_2, \ldots, q_d)$, and α denotes the angle. The axis of the cone is the half line $\overline{OQ}$, starting from the origin, O, and going through the query point Q. From now on, we will refer to this axis as the *query line*. α is the angle between the query line and the boundaries of the region defined by this query. A conical region is created by the set of lines that make an angle β, with the half line $\overline{OQ}$ at the origin, such that $\beta \leq \alpha$. Geometrically, the query returns all the points inside the intersection of the data space with the conical region where the origin, O is the apex, alternatively called vertex, of the cone.

The similarity of two feature vectors can also be defined by the *cosine angle* between them. In fact, the cosine angle distance metric is related to the Euclidean distance metric. Suppose X and Y are two d-dimensional real vectors. The Euclidean distance between these two vectors is given by $\|X - Y\|_2 = \sqrt{||X||^2 + ||Y||^2 - 2\langle X, Y\rangle}$, where $||X||$ is the norm (length) of $X = (x_1, x_2, \ldots, x_d)$ and $\langle X, Y\rangle$ is the standard inner product of the vectors X and Y and defined as $\sum_{i=1}^{d} x_i y_i = ||X|| \cdot ||Y|| \cdot cos\alpha$, where α is the angle between the vectors. By normalizing norms of all feature vectors and a given query to 1, a simple correlation between the two metrics, cosine angle and Euclidean distance, can be achieved.

Given a query (Q, α), the surface of the query region is the cone composed of all lines that have angle α with the query line. Let $X = (x_1, x_2, \dots, x_d)$ be the set of all points on the data space defined by these lines. They satisfy the following equation:

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

The same equation can be written for the unit vector $u = (u_1, u_2, \dots, u_d)$, i.e., $||u|| = 1$, which is the normalization of the query vector, $\sum_{i=1}^{d} u_i x_i = ||X|| \cdot cos\alpha$. If the data space is defined as the [0..1] unit hypercube, then the query region becomes the intersection of the conic and the unit hypercube. Two and three dimensional illustrations are in Figure 1.

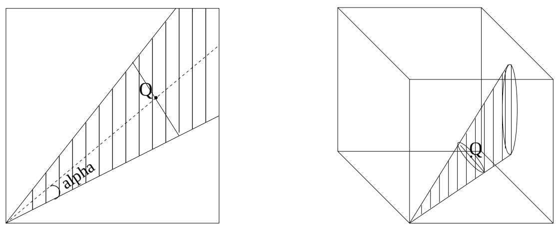


Figure 1: Conical Query

Let Q be the 3-dimensional query point, (q_1, q_2, q_3) and $u = (u_1, u_2, u_3)$ be the unit vector that is the normalization of the query vector. That is, $u_1 = q_1/\sqrt{q_1^2 + q_2^2 + q_3^2}$, $u_2 = q_2/\sqrt{q_1^2 + q_2^2 + q_3^2}$, and $u_3 = q_3/\sqrt{q_1^2 + q_2^2 + q_3^2}$. For the three dimensional case, the surface of the cone is the set of all vectors $X = (x_1, x_2, x_3)$ that satisfy the following equation.

$$(x_1 u_1 + x_2 u_2 + x_3 u_3)^2 = (x_1^2 + x_2^2 + x_3^2) cos^2\alpha. \tag{1}$$

We will use this equation and extend it later for higher dimensions.

3 Data Space Partitioning

We can classify, in general, multidimensional retrieval techniques into two basic classes: *data point* and *data space partitioning* based approaches [29, 20, 8]. The data point based approach usually arranges the data points in an index structure with a spatial containment hierarchy with possible overlaps. In the space partitioning approach, the data space is usually partitioned into mutually disjoint partitions. The data points are stored in buckets that are described by these partitions. When a query is issued, the partitions that intersect the query region are found and these partitions are retrieved as the result of the query. Especially, in the case of uniform or dense data, the data space partitioning approach is very effective [20, 4, 18, 17]. Recently, an index structure for nearest neighbor queries, VA-files [33, 19], has been proposed that uses space partitioning to quantize the multi-dimensional data and performs the query first on quantized data which is smaller in size. The approximations and the search are done using the non-overlapping rectangular partitions of the data space.

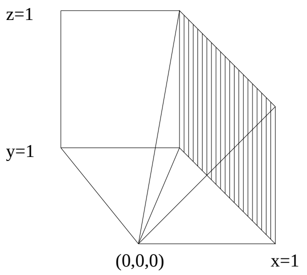
We now define two important parameters that play an important role in the efficient query processing of data space based partitioning methods.

- *Number of Buckets*: It is important to reduce the number of buckets retrieved by a query to reduce the I/O time. The number of buckets retrieved by the same query may change according to the underlying partitioning technique. The partitioning technique must be developed such that the expected number of retrieved buckets is minimized. In the literature, techniques have been optimized for only range and similarity queries [4, 18, 15] and not for conical queries. A popular partitioning approach for traditional queries is the regular grid file [29, 26].

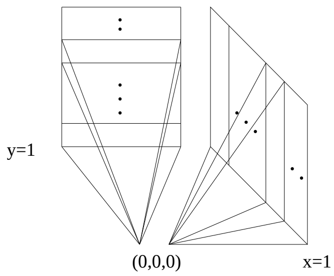
- *Disk Arm Movement*: Data transfer capacities are improving rapidly, while disk arm speed which is limited by the physical movement of the arm, is improving relatively slowly. Therefore it is very important to develop techniques that reduce disk arm movement [6]. This factor had not been considered enough before, when compared to the I/O cost optimization, but with the current trends in technology it becomes crucial to reduce the seek time spent during query execution.

Standard index structures and grid partitioning are not suitable for conical queries due to the geometric mismatch. Conical shapes differ from rectangular and spherical shapes and it is likely that traditional approaches will end up retrieving almost all the pages. Considering the geometrical properties of conical queries we now develop a data space partitioning technique to efficiently retrieve the data points defined by such queries. For the clarity of the development we assume, without loss of generality, that the data space is a unit hypercube, i.e., $[0..1]^d$. The formulations do not depend on this assumption and knowing the limit values (minimum and maximum) for each dimension is enough for the development. We start partitioning by creating d major hyperpyramids, where d is the dimensionality. The apex of all of these hyperpyramids is the origin, $O = (0, 0, \dots, 0)$. The bases of the hyperpyramids are the surfaces which do not intersect the origin O. In a d-dimensional space, there are $2d$ surfaces, where d of them do not intersect the origin point, e.g., $x = 1$. Figure 2(a) illustrates the 3-dimensional example of a major hyperpyramid whose base is the $x = 1$ plane.

After creating the major hyperpyramids, the following procedure is repeated for each of the d hyperpyramids. If the overall number of partitions is p, then each major hyperpyramid needs to be partitioned into p/d partitions. We first divide the surface into p/d parallel slices according to a chosen dimension, such that the volume of the slices is equal to each other. This dimension can be chosen in a systematic manner. For instance, for all dimensions except the first dimension x, the surface can be divided according to the first dimension, x. And the first dimension, x, can be divided with respect to any of the others, say y. Another approach would slice in round robin manner over the different dimensions. For example, in 3 dimensions, for $x = 1$ surface, we can divide the surface into slices with respect to y dimension, for $y = 1$ with respect to z, and for $z = 1$ with respect to x, i.e., in a cyclic manner (Figure 2(b)). Our analysis in this paper assumes 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. However the same analysis can easily be applied to the other approaches of slicing. Then p/d new hyperpyramids are created such that



(a) A major pyramid



(b) Partitioning of major pyramids(with basis $y = 1$,and $x = 1$)

the slices on the surface are the bases and the origin is the apex of these hyperpyramids. Thus, we have created p hyperpyramid partitions, all with a common apex, the origin O.

4 Conical Query Retrieval

In this section, we identify the conical region within the data space boundaries. In order to execute queries, we need to determine the intersection of a conical query with the underlying partitions. As a result of the query, intersected partitions are retrieved since they may have relevant data for the query. The partition identification is developed and the layout of these partitions to disk is discussed. We then establish that conical queries can be efficiently processed with the proposed techniques.

4.1 Identifying the conical region on the data space

We need to first find the intersection of a conical region within the boundaries of the unit hypercube data space. The query point is given anywhere within the general boundaries. We start by computing the intersection of the conical region with the hyper-cubic surfaces. The intersection of a cone and a plane may create a hyperbola, ellipse, or parabola on the plane depending on the position of the plane and the cone. Consider a query (Q, α). Let θ be the smaller angle formed by the query line and the plane defining the boundary of the data space unit hypercube. The relationship between α and θ defines the shape of the curve on the planes. If $\theta < \alpha$ it is a hyperbola, if $\theta = \alpha$ it is a parabola, and if $\theta > \alpha$ it is an ellipse.

We start by defining the intersection of a query cone with the boundary planes. We only need to find the intersection of the query with the planes that do not intersect the origin. After finding the intersection of the query with each surface plane we will use this information to find the partitions that are intersected in the unit hypercube. For the three dimensional case, the planes that do not intersect the origin are the planes $x_1 = 1$, $x_2 = 1$, and $x_3 = 1$. To find the intersection of a conical region with the $x_1 = 1$ plane, we will compute the intersection of $x_1 = 1$ with Equation 1 (the surface of the conical query). For $x_1 = 1$ this intersection is defined as:

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

Equation 2 represents the conic region that is created on the $x_1 = 1$ plane. Since query points Q are defined within the boundaries of the data space, i.e., unit hypercube, conical queries intersect all d infinite planes that do not pass through the origin, e.g., $x_1 = 1$. Cones with $\alpha < \pi/4$ radians have the following nice properties. If the conical region intersects the unit hyper-cubic surface within the boundaries, the intersection of the conical region with the corresponding infinite plane is always an ellipse. If the intersection is not an ellipse, we know that the conical region does not intersect the data space surface, i.e, within the unit hyper-cubic boundaries, at all. Not intersecting the data space surface means that the query does not retrieve any of the partitions that lie within the major hyperpyramid that has that surface as its base. We note that an ellipse in the infinite plane does not have to be an ellipse in the hyper-cubic surface. Cones with $\alpha > \pi/4$ radians have some degenerate cases to be handled. In general, to prevent very huge result sets, queries are expected to have angles much less than $\pi/4$ radians. In this section, we develop our analysis based on conical regions with $\alpha < \pi/4$ radians. Similar analysis can be extended for $\alpha > \pi/4$ radians.

Note that, the $x_1 = 1$ plane is divided with respect to x_2 dimension, and these slices are equi-volumed. We will compute the maximum and the minimum x_2 value of the ellipse created on the $x_1 = 1$ plane. If any of these extrema are out of the unit hypercube border, we discard them and take the boundary of the hypercube as the extremum on the surface. Once we find the partitions intersected on the surface, those partitions are exactly the partitions that are intersected by the query inside the corresponding hyperpyramid. In other words, for each major hyperpyramid, it is enough to find the intersected partitions on the base of the hyperpyramid to determine the intersected partitions in general.

To determine the extreme x_2 values inside the hypercube, we need to find the extreme x_2 values of the ellipse given in Equation 2. This is a general extrema problem, and can be solved by using the Lagrange multipliers. Since we are trying to find the extreme x_2 values constrained to Equation 2, the corresponding Lagrange equation is

$$L \;=\; x_2 - \lambda[(u_1 + x_2u_2 + x_3u_3)^2 - (1 + x_2^2 + x_3^2)cos^2\alpha]. \tag{3}$$

Taking the derivation w.r.t. x_2,

$$\begin{aligned}
\frac{\partial L}{\partial x_2} = 0 &= 1 - \lambda[2(u_1 + x_2u_2 + x_3u_3)u_2 - 2x_2cos^2\alpha] \\
\frac{1}{\lambda} &= 2u_1u_2 + 2x_2u_2^2 + 2x_3u_2u_3 - 2x_2cos^2\alpha \\
\frac{1}{\lambda} - 2u_1u_2 &= x_2(2u_2^2 - 2cos^2\alpha) + x_3(2u_2u_3)
\end{aligned} \tag{4}$$

Similarly, for x_3,

$$
\begin{aligned}
\frac{\partial L}{\partial x_3} = 0 &= -\lambda[2(u_1 + x_2 u_2 + x_3 u_3)u_3 - 2x_3 cos^2\alpha] \\
-u_1 u_3 &= x_2(u_2 u_3) + x_3(u_3^2 - cos^2\alpha)
\end{aligned} \tag{5}
$$

Using Equation 5, x_3 can be written in terms of x_2 as the following:

$$
x_3 = \frac{-u_1 u_3 - x_2(u_2 u_3)}{u_3^2 - cos^2\alpha} \tag{6}
$$

Now replace x_3 in Equation 2. In this way, we have a single quadratic equation which has a single variable, x_2. By solving this quadratic equation, we derive two values for x_2, which are actually, by the nature of Lagrange equations, the extreme x_2 values, on the $x_1 = 1$ plane, under the given restriction. The extreme x_3 values on the $x_2 = 1$ plane and the extreme x_1 values on the $x_3 = 1$ plane can be found in a similar way.

Example: We calculate the formula of the ellipse created on the $x_1 = 1$ plane, by taking the unit vector as $(1, 0, 0)$. Since, the plane is vertical to the query line, we expect to have a circular region on the plane. Let us replace the necessary values in Equation 2 and Equation 5, and find the final equation of the intersection. Since $u_1 = 1, u_2 = u_3 = 0$, Equation 5 reduces to $x_3 = 0$ and Equation 2 changes as follows:

$$
\begin{aligned}
1 &= (1 + x_2^2 + x_3^2) cos\alpha \\
x_2^2 + x_3^2 &= \frac{1}{cos^2\alpha} - 1 \\
x_2^2 + x_3^2 &= tan^2\alpha.
\end{aligned} \tag{7}
$$

Equation 7 is the equation of a circle, and since $x_3 = 0$, then $x_2 = \pm tan\alpha$. Therefore $-tan\alpha$ and $+tan\alpha$ are the extremum x_2 points of the circle. □

Alternatively, Equations 4 and 5 can be arranged as a linear system of $Ax = B$, where A and B are coefficient matrices and x is the solution to the system. Solving these equations in this manner supports a very efficient generalization of query processing for higher dimensions. For example, the linear system for 3 dimensions is as follows:

$$
\begin{pmatrix} 1 & -2u_2u_3 \\ 0 & u_3^2 - cos^2\alpha \end{pmatrix} \begin{pmatrix} 1/\lambda \\ x_3 \end{pmatrix} = \begin{pmatrix} 2x_2u_2^2 - 2x_2cos^2\alpha + 2u_1u_2 \\ -x_2u_2u_3 - u_1u_3 \end{pmatrix}
$$

and for 5 dimensions the system is:

$$
\begin{pmatrix} 1 & -2u_2u_3 & -2u_2u_4 & -2u_2u_5 \\ 0 & u_3^2 - cos^2\alpha & u_4u_3 & u_5u_3 \\ 0 & u_3u_4 & u_4^2 - cos^2\alpha & u_5u_4 \\ 0 & u_3u_5 & u_4u_5 & u_5^2 - cos^2\alpha \end{pmatrix} \begin{pmatrix} 1/\lambda \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 2x_2u_2^2 - 2x_2cos^2\alpha + 2u_1u_2 \\ -x_2u_2u_3 - u_1u_3 \\ -x_2u_2u_4 - u_1u_4 \\ -x_2u_2u_5 - u_1u_5 \end{pmatrix}
$$

```
Procedure Matrices

BEGIN
    A(1,1)=1;
    for i = 2 : d − 1,
        A(1, i) = −2 ∗ (u2 ∗ u_{i+1});
        A(i, 1) = 0;
    end;
    for j = 2 : d − 1,
        A(j, j) = u_{j+1}^2 − cos^2 α;
    end;
    for i = 2 : d − 1,
        for j = 2 : d − 1,
            if i! = j then A(i, j) = u_{i+1} ∗ u_{j+1};
        end;
    end;
    end;
    B(1, 1) = 2 ∗ x2 ∗ u2 ∗ u2 − 2 ∗ x2 ∗ cos^2 α + 2 ∗ u1 ∗ u2;
    for i = 2 : d − 1
        B(i, 1) = −x2 ∗ u2 ∗ u_{i+1} − u1 ∗ u_{i+1};
    end;
END
```

Figure 2: Multi-dimensional Matrices

```
Procedure Extrema

BEGIN
    min = min_{x2};
    max = max_{x2};
    if (min > 1)||(max < 0) then INTERSECTS = 0;
    else INTERSECTS = 1;
    if min < 0 then min = 0;
    if max > 1 then max = 1;
END
```

Figure 3: Finding Extrema

In general, Procedure Matrices (Figure 2) illustrates how to create matrices A and B for any given number of dimensions. Each variable is solved in terms of x_2 and is plugged into Equation 2. The extrema for x_2 are the two roots of Equation 2.

We computed the extremum points on the plane, e.g., max_{x_2} and min_{x_2}. But what we really need is to find those points within the boundaries of the data space. Next, we compute the extremum points of the intersection within the unit surface: min and max represent these extremums. Procedure Extrema (Figure 3) checks whether the surface is intersected at all and if there is an intersection with the query, it computes the extremum intersected points on the surface.

4.2 Partition Identification and Layout

Once we know min and max, we can compute the corresponding partitions. Note that, since the partitions are created in a sequential way, all partitions between these two extreme partitions are retrieved by the query so it is sufficient to identify the two extremum partitions. We again assume that there are p partitions, so that each of the major hyperpyramid surface has p/d partitions. These partitions can be numbered as follows. Since there are d disjoint major hyperpyramids each can be numbered independently. We rank those pyramids according to their surfaces. In three dimensions, the pyramid with surface $x = 1$ is ranked as first, the pyramid with surface $y = 1$ is ranked as second and the one with surface $z = 1$ is ranked third. After ranking the partitions inside each pyramid separately, we can globally rank the partitions by ordering them according to the rank of each pyramid. In this way, we number all partitions and each partition is assigned a partition identification number.

Given the extremum points min and max, let p_{min} and p_{max} be the rank of the partitions that have these extremum points in the corresponding surface. To compute p_{min} and p_{max} we will use a function that maps a point, m, on the plane to the rank of the corresponding partition, $p(m)$. Assuming we rank equi-volumed partitions starting from $x_2 = 0$ to $x_2 = 1$, $p(m)$ is defined as

$$p(m) = \begin{cases} 1 & \text{if } m = 0 \\ \lceil m * p/d \rceil & \text{otherwise} \end{cases}$$

And extrema are found with $p_{min} = p(min)$, $p_{max} = p(max)$. All the partitions ranked between p_{min} and p_{max} need to be retrieved. By repeating this procedure for all d surfaces that do not intersect the origin, all partitions needed to be retrieved as a result of the conical query are identified.

Because of the nature of the partitioning and conical queries, and the round robin ranking of the partitions, a query retrieves all the partitions within a major pyramid sequentially with respect to rankings. In other words, all the partitions, in the same pyramid, retrieved by the query will be sequentially ranked with no gap between any of them. This sequentiality property of conical partitioning supports the efficient search of conical queries. If a query retrieves sequentially ranked pages it performs to make a single seek to access the necessary set of pages. With the current partitioning, this sequential ranking property is true for each major hyperpyramid, and may only be violated in the intersection regions of the pyramids. In the worst case, each pyramid's partitions may be retrieved separately with one single seek (The worst case arises when the query intersects all d major hyperpyramids). Therefore, in the worst case d seeks are enough to retrieve any query.

5 Partition Layout in Multi-disk Architecture

We now extend our technique for multi-disk architectures. In typical applications, the amount of data is usually high and multi-disk architectures are needed. Therefore, we add a third important parameter, I/O parallelism, to be optimized. The degree of parallelism obtained by the partitioning is very important for efficient multi-disk searching. Obviously, the partitioning technique itself is not enough to get the desired parallelism. Effective ways to distribute the partitions to different disks are needed. An effective approach is declustering, which has attracted the attention of several researchers [3, 27, 18, 9]. By declustering, the data partitions or buckets created by the partitioning technique are allocated to multiple I/O devices such that partitions that are most likely to be retrieved together are allocated to different disks.

- *I/O Parallelism*: Performance improvements for queries occur when the buckets involved in query processing are stored on different disks, and hence can be retrieved in parallel. The probability of being accessed together depends on the shape of the query region. The partitioning technique must be optimized for efficient declustering to reduce overall I/O cost.

We need to develop a scheme for the placement of the created partitions to multiple disks. The overall goal is to distribute the neighboring buckets across different disks while clustering within a disk to reduce seek latency on the same disk. We allocate each partition to an I/O device in a round robin fashion, i.e., $disk(p_i) = ((p_i - 1) \bmod M) + 1$, where p_i is the rank of the partition, as described in the previous section, and M is the number of available disks in the system.

In [18] we proved that if a query retrieves the qualifying buckets in sequential ranked order, then the resulting I/O parallelism is optimal for the query. If this is true for all queries then the allocation scheme is said to be optimal. With the new partitioning technique, this sequential ranking property is true for each pyramid, and may only be violated in the intersection regions of the pyramids. In the worst case, each pyramid's partitions may be retrieved separately with optimal parallelism, and this corresponds to the following I/O cost formula

$$\lceil \frac{b_1}{M} \rceil + \lceil \frac{b_2}{M} \rceil + \ldots + \lceil \frac{b_r}{M}, \rceil$$

where M is the number of disks and $b_1, b_2, \ldots, b_r$ are the number of partitions retrieved from each major pyramid (assuming r major pyramids out of d are intersected). The optimal cost is $\lceil \frac{B}{M} \rceil$, where $B = b_1 + b_2 + \ldots + b_r$.

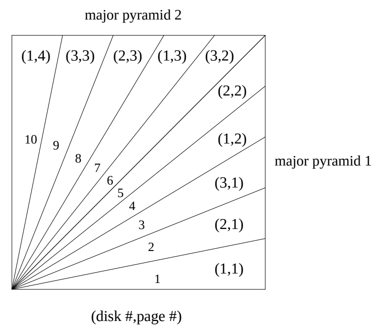


Figure 4: Disk and Page Allocations

After allocating each partition to a different disk, we need to organize partitions that are allocated to the same disk. Given partition number p_i, the *page allocation* function is $page(p_i) = \lceil \frac{p_i}{M} \rceil$. Figure 4 illustrates the two-dimensional disk and page allocations for 3 disks. By using this page allocation formula, pages that are assigned to the same disk are clustered within that disk in a round-robin way. Therefore, we have the same properties as the single disk case discussed in previous section. Pages that are assigned to the same disk and are in the same major hyperpyramid are read sequentially in that disk. Similar to the single disk case, since there are d major hyperpyramids, in the worst case d seeks in each disk are enough to retrieve all partitions corresponding to any query.

6 Performance Evaluation

We evaluate the performance of conical queries on the newly proposed multi-dimensional partitioning technique. In our simulation we implemented the partitioning technique and execute various conical queries. Note that a conical query is defined with a pair (Q, α), where Q is the query point and α is the similarity threshold.

A geometric property that is worth mentioning about conical queries is the following. Given a constant α, as the angle of the conical query, the volume of the query covers more area in the unit hypercube as the dimensionality of the data increases. The selectivity of a query can be defined as $\frac{r}{D}$,

TIME	Fast Disk	Average Disk
Average Seek Time(msec)	5.0	6.5
Latency(msec)	2.99	2.99
Transfer(MByte/sec)	60	20

Table 1: Disk Specification

where r is the number of elements in the result set of q and D is the total size of the database. For example, if the data is assumed to be uniformly distributed, the volume of the conical query is the same as the selectivity of the query. For higher dimensions, large angles will cause huge sizes of result sets.

We compare our technique with sequential scan in both single and multi-disk architectures. There has been no other technique proposed for conical queries in either domain. The current Euclidean-metric based index structures are not geometrically compatible for conical queries. Extensions for rectangular tiling based structures, e.g., grid, for conical queries are complicated and difficult to develop due to the mismatch of the geometries. For instance, one needs to find the intersection formulations for conical queries with rectangular partitions which is an expensive and hard problem for high number of dimensions. We expect the extensions to perform poorly because the original design of these index structures was based on point, range and nearest neighbor queries. Recently, in the context of high-dimensional searching, it has been shown that sequential scan is a reasonable competitor with the current index structures even if the queries are the ones these index structures are optimized for [4, 33, 7]. We therefore compare with sequential scan in all our experiments.

We evaluate the techniques for two different disk specifications. The disk specifications are based on the latest information in [30]. One of the disk specifications illustrates average speed disks available, and the other one illustrates faster disks. Data transfer rates and other specifications of these disks are shown in Table 1.

In the first setup, we evaluate the performance of conical queries in single disk environments, and present experimental results for 12 dimensions as representative. We compare both techniques by computing the time it takes conical queries to run in both average and fast disks. The processing of a query in this setup is carried out as follows. First, the partitions that need to be retrieved by the query are identified using our partition identification techniques. Then using the disk and page allocation functions, the locations of the buckets are determined and retrieved by processing appropriate seeks and transfers. The performance metric is the average time spent by a query, by running 200 random queries. The query points are randomly chosen from the data space. We vary the query angle, α, from $\alpha = \pi/60$ to $\alpha = \pi/180$ radians, i.e., $\alpha = 1..3$ degrees. Figure 5 illustrates the comparison of the query running times for 12 dimensions for average and fast single disk architectures. The total number of partitions was 100,000 and the size of each partition is 8Kbytes. For fast disks, conical partitioning achieves a speedup of 385% for $\alpha = \pi/60$ radians and a speedup of 915% for $\alpha = \pi/180$ over sequential scan. Similar speedups are achieved for average disks.

As a different cost metric, we measure the percentage of the buckets accessed to the total number of buckets, which also means the number of I/O operations needed. Note that these buckets are the ones whose bounding regions intersect the query. We compute the average percentage of buckets accessed

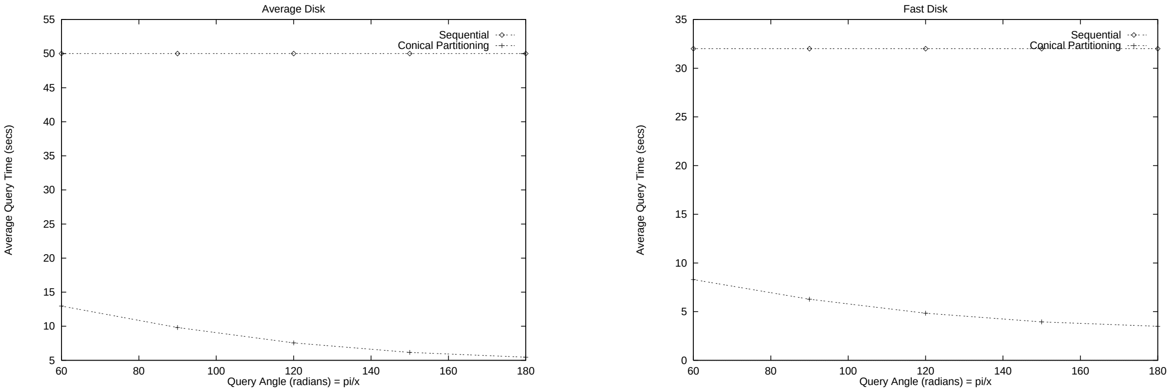


Figure 5: Query Running Times for 12 Dimensions, 100,000 Partitions

by a query, by running 200 random queries with varying angles. The query points are randomly chosen from the data space. This time, the query angle is chosen to be less than a maximum angle which corresponds to the maximum selectivity of the query result. In this part of experiment, we vary this angle from $\alpha = \pi/30$ to $\alpha = \pi/90$ radians, i.e., from 2 to 6 degrees. Figure 6 illustrates the percentage of the buckets accessed by conical queries for 12 dimensions. The percentage of buckets accessed for queries of angles up to $\pi/30$ radians is 25%. The percentage of buckets accessed is 10% for angles up to $\pi/90$ radians, and it drops as α decreases.

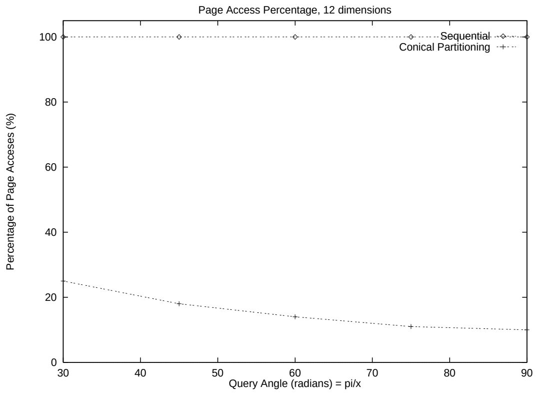


Figure 6: Percentage of Page Accesses

In the second experimental setup, we evaluate the technique for multi-disk architectures and report the results for 16 dimensions. For the multi-disk case the overall time of a query is identified by the maximum retrieval time spent at a single disk. The partitions that are retrieved during the query are read from the disks which are identified using the round-robin declustering technique. By using the disk and page allocation functions, the location of the buckets are determined and retrieved by processing appropriate seeks and transfers. The total number of buckets is 512,000 and the bucket size is 8KBytes. We vary the number of disks from $M = 4$ to $M = 32$. The query points, Q, are randomly chosen from the data space and the query angles, α, are randomly chosen up to $\pi/45$ radians, i.e., 4 degrees. Figure 7 illustrates the average running time of 200 random queries in multi-disk architectures with specifications taken from Table 1. Conical partitioning scales perfectly with the number of disks. For both fast and average disks conical partitioning achieves a speedup of 495% over sequential scan. As the number of disks increases the parallelism achieved by our technique scales optimally in almost all steps, and very close to the optimal in remaining steps.

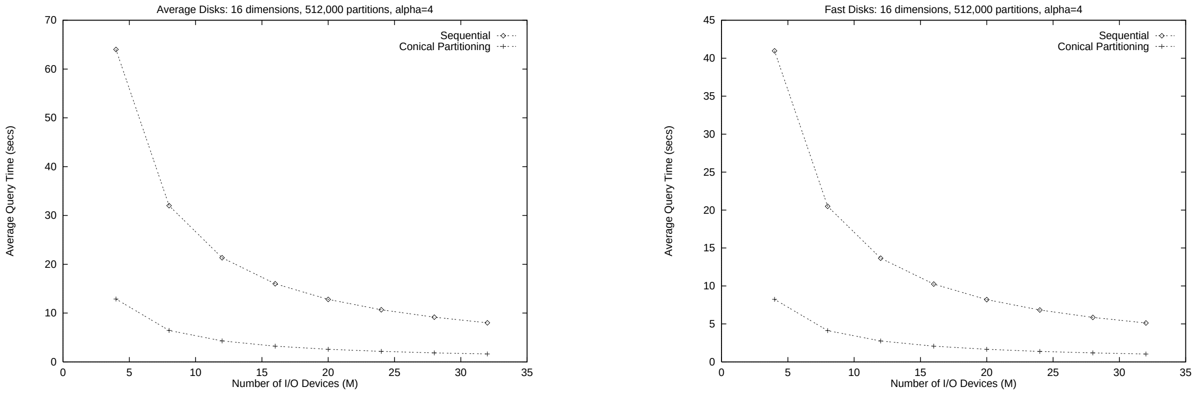


Figure 7: Query Time Performance for Multi-disk Architectures

In addition to the time metric, we evaluate our technique with another cost metric. We define the cost of a query to be the maximum number of buckets retrieved from the same disk. In a multi-disk architecture, since the data is transfered from multiple disks in parallel, the overall I/O cost is the transfer cost from a single disk that causes the maximum I/O cost among all disks. We have $512,000$ partitions and again vary the number of disks from $M = 4$ to $M = 32$. The query points, Q, are randomly chosen from the data space and the query angles, α, are randomly chosen up to $\pi/45$ radians, i.e., 4 degrees. Figure 8 illustrates the results for 16 dimensions. It can be seen that significant speedups are achieved by conical partitioning, and it also scales almost optimally with the number of disks as seen in the previous setup.

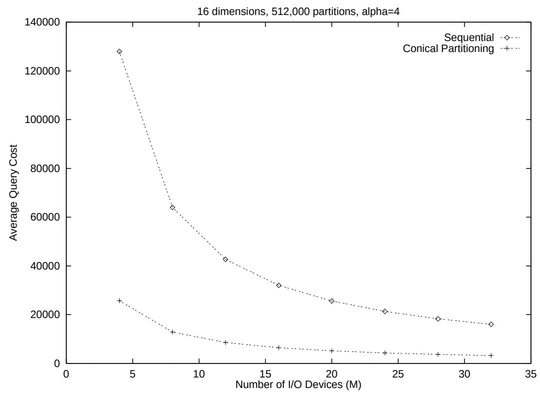


Figure 8: Parallel I/O Cost

7 Concluding Remarks

Most database system support in terms of access mechanisms, index methods, and disk allocation has been limited to range and nearest-neighbor queries in multimedia databases. In this paper, we have identified a new class of queries in multimedia databases that needs a radically different type of system level support. This class of queries is referred to as conical queries and are generally used to retrieve information from large textual collections. We started with a mathematical formulation of conical queries in a multidimensional space. We then used this formulation to develop a data space partitioning approach for retrieval of conical queries. This approach is then used to develop disk and page allocation methods for minimizing I/O costs during conical query processing by (i) minimizing the number of page accesses; (ii) maximizing the I/O parallelism; and (iii) minimizing the number of seeks. We concluded the paper with an experimental evaluation of our proposed method for both single and multiple disk environment. The results clearly establish the superiority of our approach for processing conical queries.

References

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

[2] 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.

[3] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim, and H-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, Arizona, U.S.A., 1997.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] C.-M. Chen, R. Bhatia, and R. Sinha. Declustering using golden ratio sequences. In *International Conference on Data Engineering*, pages 271–280, San Diego, California, Feb 2000.

[10] 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.

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

[12] S. Deerwester, S.T. Dumais, G.W.Furnas, T.K. Launder, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[13] 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.

[14] S. T. Dumais. Imrpving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, 23:229–236, 1991.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.

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

[23] LEDAS. Leicester database and archive service. http://ledas-www.star.le.ac.uk/, July 2000.

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

[25] 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.

[26] J. Nievergelt, H. Hinterberger, and K.C.Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems 9*, 1:38–71, March 1984.

[27] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Cyclic allocation of two-dimensional data. In *International Conference on Data Engineering*, pages 94–101, Orlando, Florida, Feb 1998.

[28] 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.

[29] H. Samet. *The Design and Analysis of Spatial Structures.* Addison Wesley Publishing Company, Inc., Massachusetts, 1989.

[30] Seagate. Performance considerations. http://www.seagate.com:80/support/kb/presales/performance.html, May 2000.

[31] 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.

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

[33] 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.

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

[35] A. J. 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.

[36] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: compressing and indexing documents and images.* Van Nostrand Reinhold, New York, NY, 1994.

Appendix

A simple two-dimensional example to motivate document similarity and querying over document repositories is as follows. Suppose the documents in the repository are described by only two keywords: 'hospital' and 'computer'. The feature vector of a document is a two-dimensional vector which has the number of occurrences of these words in that document. Instead of storing in a table, we plot these vectors as representative points for documents on a two-dimensional plane. The x-axis is the number of occurrences of the word 'hospital' and the y-axis is the number of occurrences of the word 'computer'. For example, document C in Figure 9 has a feature vector of $(400, 1)$ and is most likely to be about hospitals and not about computers. Similarly, document A with feature vector $(150, 1)$ is about hospitals. Document B on the other hand, with feature vector $(498, 495)$ has almost equal number of the words computers and hospitals in it. The semantics of this document has a mixed content of computers and hospitals such as 'the technology and computers used in hospitals'. Suppose a user gives a small document and asks for the most similar document to the given document. The feature vector of the document is computed in the same way it has been done for the documents in the repository. Let Q in Figure 9 be the query corresponding to such a document with a feature vector $(60, 60)$ and the set of documents in the repository be documents A, B, C in the same figure. Traditional range or nearest neighbor query based on Euclidean distance would return the document A, since it is the closest with respect to spatial locality. On the other hand, we expect document B to be the semantically closest to the query document Q although it is the spatially furthest in the repository.

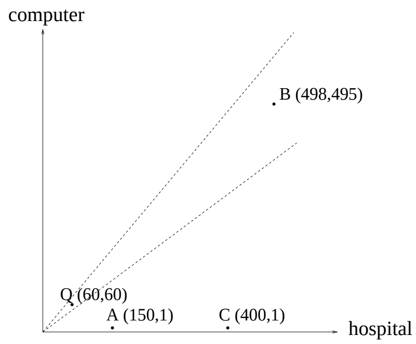


Figure 9: Documents with two-dimensional feature vectors