ELSEVIER

Data & Knowledge Engineering 34 (2000) 77–97

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

SPY-TEC: An efficient indexing method for similarity search in high-dimensional data spaces ☆

Dong-Ho Lee *, Hyoung-Joo Kim

Department of Computer Engineering, OOPSLA Laboratory, Seoul National University, Shilim-Dong Gwanak-Gu, Seoul 151-742, South Korea

Received 9 March 1999; received in revised form 10 September 1999; accepted 3 February 2000

Abstract

Most of all index structures based on the R-tree have failed to support efficient indexing mechanisms for similarity search in high-dimensional data spaces. This is due to the fact that most of the index structures commonly use balanced split strategy in order to guarantee storage utilization and the shape of queries for similarity search is a hypersphere in high-dimensional spaces. In this paper, we propose the Spherical Pyramid-Technique (SPY-TEC), an efficient indexing method for similarity search in high-dimensional data space. The SPY-TEC is based on a special partitioning strategy, which is to divide the *d*-dimensional data space first into 2*d* spherical pyramids, and then cut the single spherical pyramid into several spherical slices. This partition provides a transformation of *d*-dimensional space into one-dimensional space as the Pyramid-Technique [14] does. Thus, we are able to use a B^+-tree to manage the transformed one-dimensional data. We also propose the algorithms to process hyperspherical range queries on the data space partitioned by this partitioning strategy. Finally, we show that the SPY-TEC clearly outperforms other related techniques including the Pyramid-Technique in processing hyperspherical range queries through various experiments using synthetic and real data. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Similarity search; High-dimensional index technique; Multimedia database

1. Introduction

The high-dimensional index technique is one of the most important techniques needed for next-generation database applications.

Recently, a variety of new database applications have been developed which substantially differ from conventional database applications in many respects [14]. For example, new database applications, such as data warehousing, produce very large relations which require a multi-dimensional view on the data, and in areas such as multimedia, high-dimensional feature vectors extracted from multimedia data have been used for similarity search in content-based retrieval

☆ This research was partially funded by the 1999 BK21 IT area grant of the Ministry of Education in Korea.
* Corresponding author. Tel.: +82-2-880-1830; fax: +82-2-882-0269; Web: http://wwwoopsla.snu.ac.kr.
E-mail addresses: dhlee@oopsla.snu.ac.kr (D.-H. Lee), hjk@oopsla.snu.ac.kr (H.-J. Kim).

applications. Generally, these applications use high-dimensional data. Thus, the underlying database systems have to support efficient indexing methods for high-dimensional data spaces to provide fast retrieval in very large databases.

Recent research activities [4,13–15] reported the result that basically none of the querying and indexing techniques which provide good results on low-dimensional data perform sufficiently well on high-dimensional data for larger queries. Many database related researchers have called this problem "curse of dimensionality" [5] and many database related projects have tried to tackle it. As a result of these research efforts, a variety of new index structures [3,7,15], cost models [12,13] and query processing techniques [16] have been proposed. Most of the index structures are extensions of multi-dimensional index structures adapted to the requirements of high-dimensional indexing [14]. Thus, all of these index structures are limited with respect to the data space partitioning and suffer from the well-known drawbacks of multi-dimensional index structures, such as high costs for insert and delete operations and a poor support of concurrency control and recovery [14].

To overcome these drawbacks, the Pyramid-Technique [14] proposed the method which can use the B^+-tree to manage fast insert, update, and delete operations on high-dimensional data by transforming d-dimensional space into a one-dimensional value. In [14], Berchtold et al. proposed a special partitioning strategy which divides the data space first into $2d$ pyramids and then, cut the single pyramid into several slices. They also proposed the algorithms for processing hypercubic range queries on the space partitioned by this strategy. However, in content-based retrieval, which is one of the most important applications in multimedia database, similarity search has been frequently used and the shape of queries used in similarity search is not a hypercube, but a hypersphere [1,2,5,17]. Thus, when processing hyperspherical range queries with the Pyramid-Technique, there is a drawback which exists in all the index structures based on the bounding rectangle. Namely, they first have to perform the query using the minimum boundary rectangle (MBR) of hyperspherical queries, and then perform post-processing which determines whether the object inside the MBR of the range query is really lying in the hyperspherical region [5]. This processing needs unnecessary data page accesses and deteriorates the overall performance [1,5].

In this paper, we propose the new special space-partitioning strategy, the SPY-TEC, which is optimized for similarity search in high-dimensional spaces, and propose the algorithms for processing hyperspherical range queries on the data space partitioned by this strategy.

The SPY-TEC partitions the data space in two steps as the Pyramid-Technique does. In the first step, we first divide the d-dimensional space into $2d$ spherical pyramids having the center point of the space as their top, and the curved $(d-1)$-dimensional surface as their basis. In the second step, each of the spherical pyramids is cut into spherical slices, such as fan-shaped partitions, having their top as the center point of the data space. We show experimentally that the SPY-TEC clearly outperforms other related techniques, including the Pyramid-Technique, when processing hyperspherical range queries.

Another advantage of the SPY-TEC is the fact that we can use the main advantage of the Pyramid-Technique. Namely, we can transform the given d-dimensional data space into a one-dimensional value by the partitioning strategy of the SPY-TEC. Thus, we can use a B^+-tree to store and access data items, and take advantage of all the benefits of a B^+-tree, such as fast insert, update and delete operations, and good concurrency control and recovery [14]. The SPY-TEC can easily be implemented on top of an existing DBMS as the Pyramid-Technique does.

This paper is organized as follows. Section 2 describes related work in high-dimensional indexing techniques in recent years. In Section 3, we explain the Pyramid-Technique in detail and the disadvantages of this method in similarity search. In Sections 4 and 5, we present our new method, especially focusing on the query processing algorithm of the SPY-TEC. Then, in Section 6, we show a variety of experiments using synthetic data and real data to demonstrate the impact of the SPY-TEC. Finally, Section 7 contains conclusions and our future works.

2. Related work

A few high-dimensional index structures have been proposed in recent years.

The TV-tree [7] presented by Jagadish and Faloutsos is an R*-tree-like index structure for high-dimensional feature vectors. It improves the performance of the R*-tree by employing the reduction of dimensionality and the shift (telescoping) of active dimensions [10]. The reduction of dimensionality is achieved by activating only a few of the more important dimensions, not activating all the dimensions for indexing. If feature vectors in a sub-tree have the same coordinates on the most important active dimensions, the shift of active dimensions, in which the dimension used for sub-tree branching is made inactive and the less important dimension is newly activated for indexing, occurs [10]. As mentioned in [4,10,14], the major drawback of the TV-tree is that information on the behavior of the dimensions is required. In other words, there exists no such feature vector, that always allows the shift of active dimensions because real-valued feature vectors usually have wide diversity. Thus, the effectiveness of the TV-tree is dependent on the application domain.

The SS-tree [4] is another R-tree-like index structure designed for the similarity indexing of high-dimensional point data. It uses bounding spheres instead of bounding boxes in the directory to enhance the performance of the nearest neighbor query. As depicted in [14], although the SS-tree outperforms the R*-tree, spheres tend to overlap in high-dimensional spaces.

The SR-tree [10] is an index structure integrating the advantage of the R-tree and the SS-tree to avoid the drawbacks that appear when using only bounding spheres in the SS-tree. In other words, it uses both bounding boxes and bounding spheres in the directory. Thus, the performance of the SR-tree clearly outperforms the R*-tree and the SS-tree. The SR-tree and the SS-tree mainly focused on the improvement of the performance on nearest neighborhood queries.

The X-tree [15] is another variant of the R*-tree and improves the performance of the R*-tree using two techniques. The first is an overlap-free split algorithm which divides the search space into disjoint regions like the K-D-B-tree. The second is the concept of a supernode. If the overlap-free split algorithm leads to an unbalanced directory, the X-tree omits the split and the according directory node becomes the so-called supernode [14]. Supernodes are oversized directory nodes which are arranged to circumvent the overlap among nodes. The X-tree improves the performance of point queries and enhances the I/O throughput for reading and writing nodes [10].

The VAMSplit R-tree [3] is another optimized R-tree, i.e., it is built in the top-down manner with a given data set. Thus, all data items must be available at the time of creating the index [10,14]. According to [3], the VAMSplit R-tree outperforms both the R*-tree and the SS-tree. However, the size of the VAMSplit R-tree is limited by the main memory because it must be built in the main memory and then stored on a secondary storage [3,14]. The Hilbert R-tree [6] has the properties analogous to those of the VAMSplit R-tree. Thus, these memory-based indexing methods are called *static index structures* [14].

As mentioned in [12,14], all of the above approaches have the common property that they must use the balanced splits when splitting a data page in order to guarantee storage utilization. The Pyramid-Technique [14] showed analytically that this balanced split is the worst case in high-dimensional indexing because the resulting pages have an access probability close to 100% when going to high-dimensional data space [14].

To overcome this drawback, Berchtold et al. proposed the Pyramid-Technique, a new space-partitioning strategy and the algorithms for processing hypercubic range queries on the space partitioned by this strategy. In the next section, we will describe the Pyramid-Technique in detail because this technique is a motivation of our work.

3. The Pyramid-Technique

The main concept of the Pyramid-Technique is that they can use the B^+-tree to store and access the transformed one-dimensional value after transforming d-dimensional data space into one-dimensional data space. Thus, they can use all the advantages of the B^+-tree, such as fast insert, update and delete operations, good concurrency control and recovery, and easy implementation.

The transformation of d-dimensional data space into one-dimensional data space is based on a special space partitioning which is achieved in two steps. In the first step, the data space is split into $2d$ pyramids having the center point of the data space $(0.5, 0.5, \dots, 0.5)$ as their top and a $(d-1)$-dimensional surface of the data space as their base. In the second step, each of the $2d$ pyramids is divided into several partitions parallel to the basis of the pyramid. Each of the partitions corresponds to one data page of the B^+-tree.

The (I) of Fig. 1 shows the space-partitioning strategy of the Pyramid-Technique in a two-dimensional example. In (a) of Fig. 1(I), the space has been divided into four triangles, which all have the center of the data space as the top and one edge of the data space as the base. In (b) of Fig. 1(I), these four triangles are split again into several partitions, each corresponding to one data page of the B^+-tree. This strategy can be extended to d-dimensional data space in a straight-forward way [14].

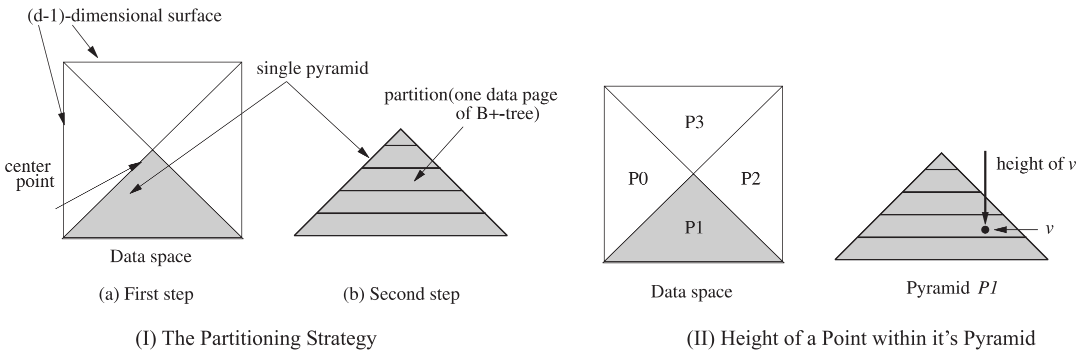

(d-1)-dimensional surface
single pyramid
partition(one data page of B+-tree)
center point
Data space
(a) First step
(b) Second step
P3
P0
P2
P1
height of v
v
Data space
Pyramid P1
(I) The Partitioning Strategy
(II) Height of a Point within it's Pyramid

Fig. 1. Pyramid-Technique.

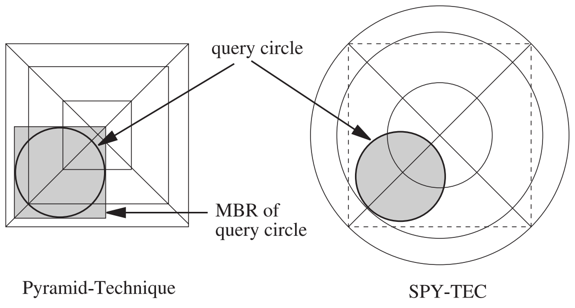


Fig. 2. Partitioning strategies.

Given a *d*-dimensional point, they first determine the number of the pyramid that the point belongs to, and then calculate the height of the point within its pyramid. By adding this height to the number of the pyramid, they can transform the *d*-dimensional point to one-dimensional point. The (II) of Fig. 1 shows this processing. They insert this one-dimensional point into the B^+-tree and perform queries on the B^+-tree. They also proposed the algorithms for processing point queries and hypercubic range queries based on this space-partitioning strategy.

However, the shape of queries used for similarity search is not a hypercube but a hypersphere. Thus, as mentioned above, if we were to process hyperspherical range queries using the Pyramid-Technique, we have to access unnecessary data page. Therefore, the overall performance is degraded. To overcome this drawback, we proposed a new space-partitioning strategy in which the data space is partitioned as like the annual ring of a tree. The right of Fig. 2 shows the partitions resulting from a split of the SPY-TEC, which is shaped like the annual ring of a tree, in a two-dimensional example.

Given a query circle as depicted in Fig. 2, the Pyramid-Technique accesses unnecessary data pages because it first has to perform the query using an MBR of the query circle. In the left of Fig. 2, we can see that the MBR of the query circle intersects eight partitions out of a total of 12 partitions. Thus, the Pyramid-Technique has to access eight pages for processing this query circle. However, the SPY-TEC can process the query with accessing only four pages. Thus, four page accesses are saved in the SPY-TEC. As we will show in our experimental evaluation, the SPY-TEC will save more page accesses when going to higher dimensions and more data objects.

4. The SPY-TEC

The main idea of the SPY-TEC is based on the observation that spherical splits will be better than right-angled splits of the Pyramid-Technique for similarity search. This observation is due to the fact that the shape of the queries used in similarity search is not a hypercube, but a hypersphere. The SPY-TEC is to transform the *d*-dimensional data points into one-dimensional values and then store and access the values using the B^+-tree as the Pyramid-Technique does. Also, we store a *d*-dimensional point *plus* the corresponding one-dimensional key as a record in the leaf nodes of the B^+-tree. Therefore, we do not need an inverse mechanism of this transformation.

The transformation itself is based on a specific partitioning of the SPY-TEC. To define the transformation, we first explain the data space partitioning strategy of the SPY-TEC.

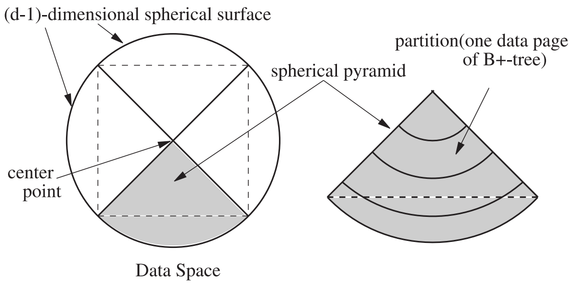

(d-1)-dimensional spherical surface
partition(one data page of B+-tree)
spherical pyramid
center point
Data Space

Fig. 3. Partitioning strategy of the SPY-TEC.

4.1. Data space partitioning

The SPY-TEC partitions the data space in two steps: The first step is the same as the Pyramid-Technique. Namely, we split the d-dimensional data space into $2d$ spherical pyramids having the center point of the data space $(0.5, 0.5, \ldots, 0.5)$ as their top and a $(d-1)$-dimensional curved surface of the data space as their bases. The second step is to divide each of the $2d$ spherical pyramids into several spherical slices with a single slice corresponding to one data page of the B^+-tree. Fig. 3 shows the data space partitioning of the SPY-TEC in a two-dimensional example. First, the two-dimensional data space has been divided into four spherical pyramids resembling a fan. All of these spherical pyramids have the center point of the data space as their top and one curved line of the data space as their bases. In the second step, each of these four spherical pyramids is split again into several data pages which are shaped like the annual ring of a tree. Given a d-dimensional space instead of the two-dimensional space, the base of the spherical pyramid is not a one-dimensional curved line as in the example, but a $(d-1)$-dimensional curved hyperplane. As a sphere of dimension d has $2d$ $(d-1)$-dimensional curved hyperplane as a surface, we obviously obtain $2d$ spherical pyramids such as in the case of the Pyramid-Technique.

Numbering the spherical pyramids is the same as in the Pyramid-Technique. Given a point v, we have to find the dimension i having the maximum deviation $|0.5 - v_i|$ from the center to determine the spherical pyramid which the point v belongs to. If v_i is greater than, or equal to 0.5, then the spherical pyramid which the point v belongs to is sp_{i+d}. If it is smaller than 0.5, the spherical pyramid which the point v belongs to is sp_i. As depicted in (I) of Fig. 4, the value of $|0.5 - v_1|$ of a point v in two-dimensional space is greater than the value of $|0.5 - v_0|$. Thus, the dimension having the maximum deviation $|0.5 - v_i|$ from the center is d_1 and the value of v_1 is smaller than 0.5. Therefore, the point v belongs to the spherical pyramid sp_1. For example, consider another point $v' = (0.8, 0.4)$. The dimension having the maximum deviation from the center for each dimension of v' is $d_0(0.3 = |0.5 - v'_0| > |0.5 - v'_1| = 0.1)$. And, the value of v'_0 is greater than 0.5. Therefore, the point v' belongs to the spherical pyramid $sp_{(0+2)}$. Although the formal expression about this processing is the same as [14], we redefine it formally to help in understanding the partitioning strategy of the SPY-TEC.

Definition 1 (*Spherical pyramid of a point v*). A d-dimensional point v is defined to be located in spherical pyramid sp_i.

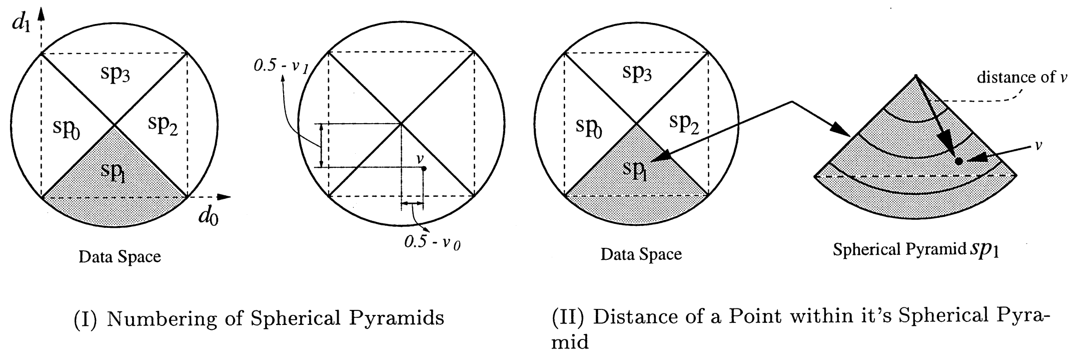


Fig. 4. The SPY-TEC.

$$i = \begin{cases} j_{\max} & \text{if } v_{j_{\max}} < 0.5, \\ (j_{\max} + d) & \text{if } v_{j_{\max}} \geqslant 0.5, \end{cases}$$

$$j_{\max} = (j \,|(\forall k, 0 \leqslant (j,k) < d,\ j \neq k : |0.5 - v_j| \geqslant |0.5 - v_k|)).$$

In Definition 1, $j_{\max}$ is the dimension having the maximum deviation $|0.5 - v_i|$ from the center for each dimension of a d-dimensional point v and i is the number of the spherical pyramid which v belongs to.

In order to transform d-dimensional data into a one-dimensional value, we have to determine the location of a point v within its spherical pyramid. The Pyramid-Technique uses the height of the point within the pyramid as the location of the point. However, we use the distance from the point to the center point of the data space as the location of the point. The (II) of Fig. 4 shows the process of determining the distance of the point v as the location within its spherical pyramid. We assume that the distance function is the Euclidean distance which is frequently used for similarity measurement in content-based retrieval. More formally:

Definition 2 (*Distance of a point v*). Given a d-dimensional point v, the distance d_v of the point v is defined as

$$d_v = \sqrt{\sum_{i=0}^{d-1} (0.5 - v_i)^2}.$$

According to Definitions 1 and 2, we are able to transform a d-dimensional point v into a one-dimensional value $(i \cdot \mathrm{ceil}(\sqrt{d}) + d_v)$. In this one-dimensional value, i is the number of the spherical pyramid which the point v belongs to, d is the dimension of the point v, and d_v is the distance from the point v to the top of its spherical pyramid. More formally:

Definition 3 (*Spherical pyramid value of a point v*). Given a d-dimensional point v, let i be the number of the spherical pyramid to which v belongs according to Definition 1, and d_v be the distance of v according to Definition 2. Then, the spherical pyramid value spv_v of v is defined as

$$spv_v = \left(i \cdot \mathrm{ceil}\left(\sqrt{d}\right) + d_v\right).$$

Note that i is an integer in the range $[0, 2d]$, d_v is a real number in the range $[0, 0.5\sqrt{d}]$ and $\mathrm{ceil}(\sqrt{d})$ is the smallest integer not less than or equal to $\sqrt{d}$. Therefore, every point within a spherical pyramid sp_i has a value in the interval of $[i \cdot \mathrm{ceil}(\sqrt{d}), (i \cdot \mathrm{ceil}(\sqrt{d}) + 0.5\sqrt{d})]$. In order to make the sets of spherical pyramid values covered by any two spherical pyramids sp_i and sp_j to be disjunct, we multiply i by $\mathrm{ceil}(\sqrt{d})$. If we would not multiply i by $ceil(\sqrt{d})$, then the interval of every point within a spherical pyramid sp_i is $[i, (i + 0.5\sqrt{d})]$. Thus, there may be intersections in the sets of spherical pyramid values covered by any two spherical pyramids sp_i and sp_j when the dimension is higher than 4.

For example, in a 16-dimensional data space, the interval of every point within a spherical pyramid sp_1 is $[1, 3]$, and the interval of every point within sp_2 is $[2, 4]$. Therefore, these two intervals have an intersection. This intersection may cause the key values of the B$^+$-tree to be redundant. The redundancy of the key values degrades the performance of the B$^+$-tree. In order to avoid this effect, we do multiply the spherical pyramid number i by $\mathrm{ceil}(\sqrt{d})$. However, note that the multiplier needs not to be $\mathrm{ceil}(\sqrt{d})$. Any multiplier can be used if it prevents the sets of spherical pyramid values from being overlapped. Note further that this transformation is not injective, i.e., two points v and v' may have the same pyramid value, but, as mentioned above, we do not need an inverse transformation because we store a d-dimensional point *plus* the corresponding one-dimensional key as a record in the leaf nodes of the B$^+$-tree. Therefore, our technique does not require a bijective transformation as the Pyramid-Technique [14] does not.

4.2. Index creation

It is a very simple task to build an index using the SPY-TEC as for the Pyramid-Technique. Given a d-dimensional point v, we first determine the spherical pyramid value spv_v of the point and then insert the point into a B$^+$-tree using spv_v as a key. Finally, we store the point v and spv_v in the according data page of the B$^+$-tree. Update and delete operations can be done similarly.

The spherical pyramid values of points which all belong to the same spherical pyramid lies in an interval given by the minimum and maximum key values of the data pages. Thus, a single B$^+$-tree data page corresponds to a partition of a spherical pyramid as shown in the right of Fig. 3.

5. Query processing

There are two types of queries which are used in similarity search [4]. One is the k-nearest neighbor query that returns the k most similar objects to the query object. The other is the hyperspherical range query that returns all objects within a threshold level of similarity to the query objects. Although the shape of these two queries is a hypersphere in high-dimensional spaces, both have totally different implications when processing queries. For example, in the case of range

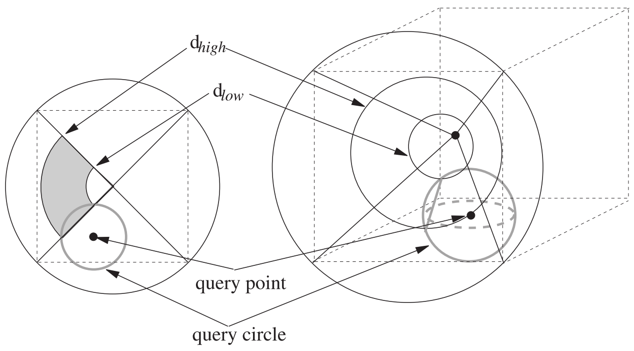


Fig. 5. Transformation of hyperspherical range queries.

queries, the order in which the pages have to be processed is known in advance, whereas in the case of k-nearest neighbor queries, this order is rather arbitrary.

In this section, we propose the algorithms for processing hyperspherical range queries. The task of processing this query using the SPY-TEC is a very complex operation in contrast to the insert, delete and update operations. The point queries which are defined as *Given a query point q*, *decide whether q is in the database*, can be processed as for the Pyramid-Technique.

For hyperspherical range queries, two parameters are given. The first parameter is a query point which represents the query object and the second is a threshold level of similarity which represents the limit of distance from a query point. The problem is defined as follows:

Given a query point and a threshold (ε) *of similarity*

$$\text{query point}(Q) : [q_0, q_1, \ldots, q_{d-1}], \qquad \text{distance} : \varepsilon$$

find the points in the database which are within the distance (ε) *from the query point*.

Note that the geometric correspondence of this similarity query is a hypersphere having the query point as the center point and ε as the radius of the sphere. To process this hyperspherical range query, we have to transform the d-dimensional query into one-dimensional interval queries on the B^+-tree. However, as the simple two-dimensional example depicted in the left of Fig. 5 demonstrates, a query circle may intersect several spherical pyramids and the computation of the area of intersection is not trivial.

We process a hyperspherical range query in two main steps: in the first step, we have to determine which spherical pyramids are affected by the query, and then in the second step, we have to determine the ranges inside the spherical pyramids. The test to see whether a point is inside the ranges or not is based on a single attribute criterion (d_v between two values). Therefore, determining all such objects is a one-dimensional indexing, i.e., a B^+-tree indexing problem. Objects outside the ranges are guaranteed not to be contained in the query circle. Points lying inside the ranges, are candidates for a further investigation. As depicted in the left of Fig. 5, points lying between d_{low} and d_{high} are candidates. Some of the candidates are hits, others are false hits. Thus, in the refinement step, we have to investigate whether a point is inside the query circle or not. However, in order to investigate whether or not a point is inside the query circle, we have to calculate the distances from the query point to all of the points of the candidates. Through our experiments, we found out that this task consumes a lot of CPU time. Thus, before starting the

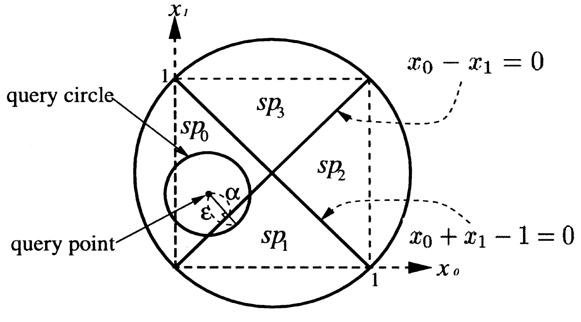


Fig. 6. Intersection of a spherical pyramid and a query circle.

refinement step, we perform the filtering step, which tests whether or not the *i*th coordinate of a point is in the interval $[q_i - \varepsilon, q_i + \varepsilon]$. This is a simple comparison operation so that its CPU time is less than that of the arithmetic operation. Therefore, we can eliminate a large amount of false hits before the refinement step and can save a lot of CPU time.

For the sake of simplicity, we focus on the description of the algorithm only on spherical pyramids sp_i where $i < d$. However, our algorithm can be extended to all spherical pyramids in a straight-forward manner. Given a query point and an ε, we have to determine whether or not a spherical pyramid sp_i is affected by a given query area in the first step of our algorithm.

In Fig. 6, the query circle intersects the spherical pyramids sp_0 and sp_1. The spherical pyramid sp_0 intersects the query circle because the query point falls into sp_0. And, in the case of sp_1, the distance (α) from the query point to the closest-side plane of sp_1 is shorter than ε which is the radius of query circle. Thus, the sp_1 intersects the query circle. As the example of Fig. 6, the basic idea of the following Lemma 1 is that we can determine whether or not a spherical pyramid sp_i intersects the query circle using a simple formula, which calculates the distance from a point to a hyperplane, used in geometry [8].

Lemma 1 (Intersection of a spherical pyramid and a query circle). *Given a query point $Q = [q_0, q_1, \ldots, q_{d-1}]$ and ε, let j ($j < d$) be the number of a spherical pyramid to which a query point belongs and i be the number of a spherical pyramid which will be tested for an intersection. The intersection of a query circle and a spherical pyramid sp_i is defined as*

Case 1 ($i = j$): *In this case, the spherical pyramid sp_i always intersects the query circle.*

Case 2 ($|i - j| = d$): *In this case, the spherical pyramid sp_i is in the opposite side of sp_j. Let β be the distance from the query point to the center of the data space. According to Definition* 2, $\beta = d_Q$.

$$\beta - \varepsilon \leqslant 0.$$

Case 3 ($i < d$):

$$\frac{|q_j - q_i|}{\sqrt{2}} \leqslant \varepsilon.$$

Case 4 ($i \geqslant d$):

$$\frac{|q_j + q_i - 1|}{\sqrt{2}} \leqslant \varepsilon.$$

Proof. Given a point ($[q_0, q_1, \ldots, q_{d-1}]$) and a hyperplane ($k_0x_0 + k_1x_1 + \cdots + k_{d-1}x_{d-1} + C = 0$), the distance from the point to the hyperplane in geometry is defined as

$$\text{Distance} = \frac{|k_0q_0 + k_1q_1 + \cdots + k_{d-1}q_{d-1} + C|}{\sqrt{k_0^2 + k_1^2 + \cdots + k_{d-1}^2}}. \tag{1}$$

We are able to prove Cases 3 and 4 using this formula.
1. sp_i is a spherical pyramid to which a query point belongs, because $i = j$. Therefore, the query region intersects the spherical pyramid sp_i.
2. sp_i is a spherical pyramid in the opposite side of sp_j. Thus, if $\beta \leqslant \varepsilon$, then the center point of the data space is included in the query region. Therefore, sp_i is affected by the query region.
3. In formula (1), the index k_n and the constant C have discrete values $[-1, 0, 1]$ because of unit space. If $i < d$, then an equation for the closest-side plane of a spherical pyramid adjacent to the query point is $k_jx_j + k_ix_i = 0$ as depicted in the two-dimensional example of Fig. 6. This formula can be extended to d-dimensional data space in a straight-forward way. Given a d-dimensional space instead of the two-dimensional space, the side plane of a spherical pyramid is not a one-dimensional line as in the example of Fig. 6, but a $(d-1)$-dimensional hyperplane, and the equation for this $(d-1)$-dimensional hyperplane has the common property that all indices except k_i and k_j are 0. In this case, $k_j = 1$ and $k_i = -1$ because $i < d$. Thus, the distance from the query point to the closest-side plane of an adjacent spherical pyramid sp_i is $|q_j - q_i|/\sqrt{2}$. Therefore, if $|q_j - q_i|/\sqrt{2} \leqslant \varepsilon$, then a part of the query region intersects sp_i.
4. If $i \geqslant d$, then an equation for the closest-side plane of a spherical pyramid adjacent to the query point is $k_jx_j + k_ix_i - 1 = 0$ (refer to Fig. 6). In this case, $k_j = 1$ and $k_i = 1$ because $i \geqslant d$. Thus, the distance from the query point to the closest-side plane of adjacent spherical pyramid sp_i is $|q_j + q_i - 1|/\sqrt{2}$. Therefore, if $|q_j + q_i - 1|/\sqrt{2} \leqslant \varepsilon$, sp_i is affected by the query region. □

If the query circle includes the center point of the data space, all the spherical pyramids are affected by the query region such as in Case 2.

In the second step, we have to determine which spherical pyramid values inside an affected spherical pyramid sp_i are affected by the query. Thus, we should find an interval $[d_{\text{low}}, d_{\text{high}}]$ in the range of $[0, 0.5\sqrt{d}]$ so that the spherical pyramid values of all points inside the intersection of the query circle and spherical pyramid sp_i are in the interval $[i \cdot \text{ceil}(\sqrt{d}) + d_{\text{low}}, i \cdot \text{ceil}(\sqrt{d}) + d_{\text{high}}]$.

We are able to determine the values of d_{low} and d_{high} using a simple mathematical formula such as the *Pythagoras theorem* [8].

For a case in which the center point of the data space is in the query region, the value d_{low} is always 0. Thus, we have only to determine the value d_{high}. In case (a) of Fig. 7(I), d_{high} can be determined by using the distance from the center point of the data space to the query point and ε as the radius of query circle. If we let β be the distance from the center to the query point, then d_{high} is $(\beta + \varepsilon)$. Also, in cases (b) or (d) of Fig. 7(I), d_{high} can be determined by using the Pythagoras theorem. Let α be the distance from the closest-side plane of a spherical pyramid adjacent to the

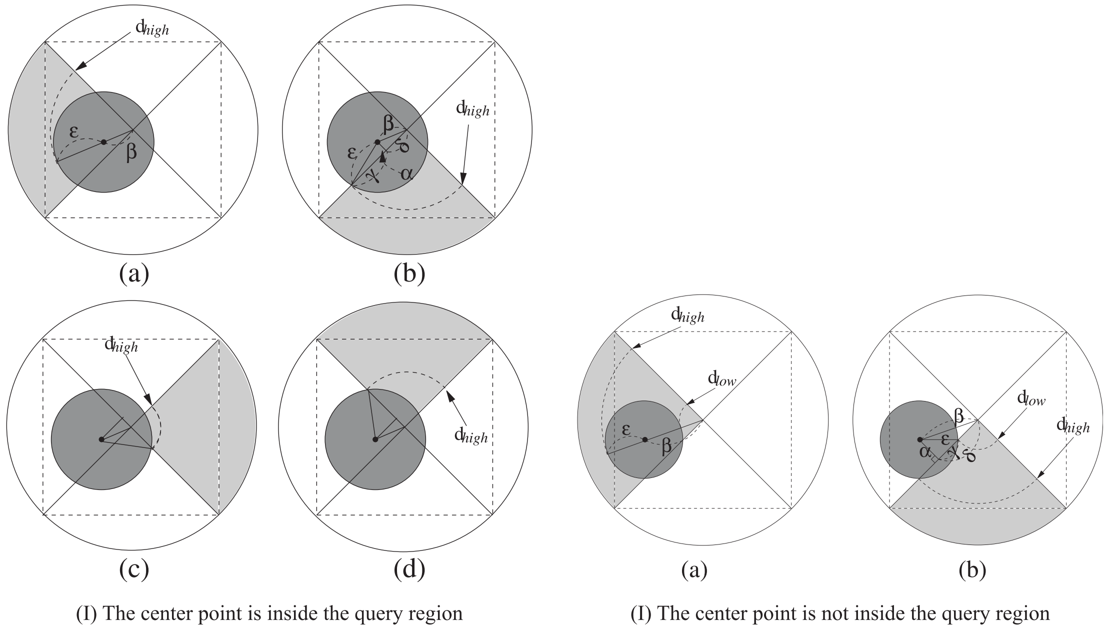

d_high
ε
β
δ
α
γ
d_low
(a)
(b)
(c)
(d)
(I) The center point is inside the query region
(I) The center point is not inside the query region

Fig. 7. Interval of intersection of query and spherical pyramid.

query point. The length (δ) of the base line in a right-angled triangle which consists of two sides, α and β, is $\sqrt{\beta^2 - \alpha^2}$, and the length ($\gamma$) of the base line in a right-angled triangle which consists of two sides, α and ε, is $\sqrt{\varepsilon^2 - \alpha^2}$. In this case, d_{high} is $(\gamma + \delta)$. Finally, in case (c) of Fig. 7(I), the spherical pyramid sp_i is in the opposite side of the spherical pyramid sp_j which the query point belongs to. Thus, d_{high} is $(\gamma - \delta)$, but in this case, note that α is the maximum value of the distances from the query point to the closest-side plane of all adjacent spherical pyramids.

For a case in which the center point of the data space is not inside the query region, we are able to determine the values of d_{low} and d_{high} analogously. As depicted in (a) of Fig. 7(II), in a spherical pyramid which the query point belongs to, d_{low} and d_{high} are determined as follows: $d_{\text{low}} = \beta - \varepsilon,\ d_{\text{high}} = \beta + \varepsilon$. And, in a spherical pyramid which the query point does not belong to (refer to (b) of Fig. 7(II)), d_{low} and d_{high} are determined as follows: $d_{\text{low}} = \delta - \gamma,\ d_{\text{high}} = \delta + \gamma$. As depicted in an example of Fig. 7, the basic idea of the following Lemma 2 is to determine the interval $[d_{\text{low}}, d_{\text{high}}]$, in which the query circle intersects the spherical pyramids, using the Pythagoras theorem.

Lemma 2 (Interval of intersection of query and spherical pyramid). *Given the number j of a spherical pyramid to which the query point belongs, and the number i of a spherical pyramid which is affected by the query region, the intersection interval* $[d_{\text{low}}, d_{\text{high}}]$ *is defined as follows*:

Case 1: *The case in which the center point of the data space is inside the query region.*

Subcase 1.1 ($j = i$): *Let* β *be the distance from the center point to the query point and* ε *be the radius of the query circle.*

$$d_{\text{low}} = 0, \quad d_{\text{high}} = \beta + \varepsilon.$$

Subcase 1.2 ($|j-i|=d$): *Let α be the maximum value of the distances from the query point to the closest-side plane of all the adjacent spherical pyramids and δ be the length of the base line in a right-angled triangle which consists of two sides, α and β. Also, let γ be the length of the base line in a right-angled triangle which consists of two sides, α and ε.*

$$d_{\text{low}} = 0, \quad d_{\text{high}} = \gamma - \delta = \sqrt{\varepsilon^2 - \alpha^2} - \sqrt{\beta^2 - \alpha^2}.$$

Subcase 1.3 (Otherwise): *Let α be the distance from the query point to the closest-side plane of an adjacent spherical pyramid and δ, γ be the same as in Subcase* 1.2.

$$d_{\text{low}} = 0, \quad d_{\text{high}} = \gamma + \delta = \sqrt{\varepsilon^2 - \alpha^2} + \sqrt{\beta^2 - \alpha^2}.$$

Case 2: *The case in which the center point of the data space is not inside the query region.*
Subcase 2.1 ($j = i$):

$$d_{\text{low}} = \beta - \varepsilon, \quad d_{\text{high}} = \beta + \varepsilon.$$

Subcase 2.2 ($j \neq i$):

$$d_{\text{low}} = \delta - \gamma, \quad d_{\text{high}} = \delta + \gamma.$$

Proof. We have to show for any point v which is located inside the query circle and an affected spherical pyramid sp_i that d_v is lying in the resulting query interval $[d_{\text{low}}, d_{\text{high}}]$. Therefore, we have to prove that

$$d_{\text{low}} \leqslant d_v \leqslant d_{\text{high}}.$$

1. $d_{\text{low}} \leqslant d_v$: This holds because the center of the data space is included in the query region so that $d_{\text{low}} = 0 \leqslant d_v$.
   1.1. $d_v \leqslant d_{\text{high}}$: This case corresponds to (a) of Fig. 7(I). Let v be the point which has the maximum of the distances of points lying inside the query region and an affected spherical pyramid sp_i. Then, d_v is less than, or equal to $(\beta + \varepsilon)$. Therefore, $d_v \leqslant d_{\text{high}} = \beta + \varepsilon$.
   1.2. $d_v \leqslant d_{\text{high}}$: In this case, a spherical pyramid sp_i is in the opposite side of a spherical pyramid sp_j so that there is no side plane of a spherical pyramid adjacent to the query point (refer to (c) of Fig. 7(I)). δ, γ are defined as follows by the Pythagoras theorem:

$$\delta = \sqrt{\beta^2 - \alpha^2}, \quad \gamma = \sqrt{\varepsilon^2 - \alpha^2}.$$

Let v be the point which has the maximum of the distances of points lying inside the query region and an affected spherical pyramid sp_i. Then, d_v is less than, or equal to $(\gamma - \delta)$. Therefore, $d_v \leqslant d_{\text{high}} = \gamma - \delta$.
   1.3. In this case, there is the closest-side plane of a spherical pyramid adjacent to the query point (refer to (b), (d) of Fig. 7(I)). Let v be the point which has the maximum of the distances of points lying inside the query region and an affected spherical pyramid sp_i. Then, d_v is less than, or equal to $(\gamma + \delta)$. Therefore, $d_v \leqslant d_{\text{high}} = (\gamma + \delta)$.

2. $d_{\text{low}} \leqslant d_v \leqslant d_{\text{high}}$: In this case, d_{low} is not 0 and a spherical pyramid in the opposite side of the spherical pyramid which the query point belongs to, is not affected by the query region. Thus, we have to show that $d_{\text{low}} \leqslant d_v \leqslant d_{\text{high}}$.
   2.1. In this case, sp_i is the spherical pyramid which the query point belongs to (refer to (a) of Fig. 7(II)). Let v be the point which has the minimum of the distances of points lying inside the query region and an affected spherical pyramid sp_i. Then, d_v is greater than, or equal to $(\beta - \varepsilon)$, and if v is the point which has the maximum of the distances of points lying inside the query region and an affected spherical pyramid sp_i, then, d_v is less than, or equal to $(\beta + \varepsilon)$. Therefore, $(\beta - \varepsilon) = d_{\text{low}} \leqslant d_v \leqslant d_{\text{high}} = (\beta + \varepsilon)$.
   2.2. δ, γ are the same as in Subcase 1.3. This case corresponds to (b) of Fig. 7(II). Let v be the point which has the minimum of the distances of points lying inside the query region and an affected spherical pyramid sp_i. Then, d_v is greater than, or equal to $(\delta - \gamma)$, and if v is the point which has the maximum of the distances of points lying inside the query region and an affected spherical pyramid sp_i, then, d_v is less than, or equal to $(\delta + \gamma)$. Therefore, $(\delta - \gamma) = d_{\text{low}} \leqslant d_v \leqslant d_{\text{high}} = (\delta + \gamma)$. □

If $\beta + \varepsilon > 0.5\sqrt{d}$ or $\delta + \gamma > 0.5\sqrt{d}$, then d_{high} is properly $0.5\sqrt{d}$. With Lemmas 1 and 2, we can process the hyperspherical range query by Algorithm 1.

Algorithm 1 Processing the hyperspherical range query

```
INPUT: $qp \leftarrow$ query point, $\varepsilon \leftarrow$ range
OUTPUT: set of points satisfying the query($result$)

for $i = 0$ to $2d - 1$ do
  if intersect($sp[i]$, $qp$, $\varepsilon$) then
    /* Using Lemma1, we test that a spherical pyramid $sp_i$ intersects a query sphere having $qp$
    as the query point and $\varepsilon$ as the radius of the sphere */

    determine_interval($sp[i]$, $qp$, $\varepsilon$, $d_{low}$, $d_{high}$);
    /* Using Lemma 2, we determine which spherical pyramid values inside an affected spherical
    pyramid $sp_i$ are affected by the query sphere. Namely, we find an interval [$d_{low}$, $d_{high}$] */

    cs = btree_interval_query($i \cdot ceil(\sqrt{d}) + d_{low}$, $i \cdot ceil(\sqrt{d}) + d_{high}$);
    /* perform 1-dimensional interval query on the $B^+$-tree */

    for $c$ = cs.first to cs.end do
      if inside_DiameterOfCircle($c$, $qp$, $\varepsilon$) then
        /* filtering step */
        if point_in_circle($c$, $qp$, $\varepsilon$) then
          /* refinement step */
          $result \leftarrow c$;
        end if
      end if
    end for

  end if
end for
```

6. Experimental evaluation

We performed various experiments to show the practical impact of the SPY-TEC and compared it to the following related techniques:
- Pyramid-Technique [14].
- R*-tree [9].
- X-tree [15].
- Sequential scan.

The Pyramid-Technique has been chosen for comparison, because it is a motivation of our technique. The R*-tree has been most commonly used in multi-dimensional indexing applications and the X-tree was proposed as an indexing structure for high-dimensional data. Thus, we included these techniques in our experiments. Recently, the criticism arose that index-based query processing is generally inefficient in high-dimensional data spaces and that sequential scan processing yields better performance in this case [11,14]. Therefore, we also included the sequential scan in our experiments.

For clear comparison, we assume that all the relevant information is stored in the various indexes, as well as in the file used for the sequential scan. Therefore, no additional accesses to fetch objects for presentation or further processing are needed in any of the techniques applied in our experiments.

Our experiments have been performed on SUN Sparc 20 workstations with 8 GBytes of secondary storage. We performed our experiments using both real and synthetic data sets and hyperspherical range queries with a constant radius (ε) in all the experiments. The query points were selected randomly from the data space so that the distribution of the queries equals the distribution of the data set itself. Thus, in the case of uniform data distribution, we used 100 query points that are uniformly distributed.

6.1. Evaluation using synthetic data

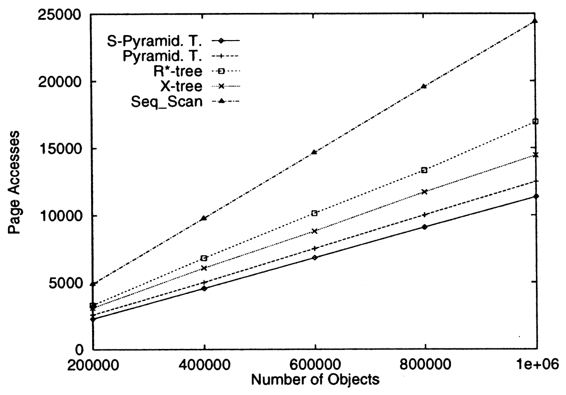
The synthetic data set contains 200,000–1,000,000 uniformly distributed points in 8–24-dimensional data spaces.

In our first experiment, we measured the performance behavior while we varied the number of objects. Before the experiment, in order to select the reasonable radii of query circles, we measured the average result set size with varying the radius. We performed hyperspherical range queries with 100 query points in a 16-dimensional data space and varied the radius of query circle from 0.3 to 0.9. We also varied the database size from 200,000 to 1,000,000. Table 1 shows the result of this experiment. In this procedure, we found that we could obtain adequate average result set size when the radii of query circles are from 0.6 to 0.8. Thus, we performed hyperspherical range queries with 100 query points and the same constant radii for each database size and measured the number of page accesses, CPU time and finally the total elapsed time needed for processing hyperspherical range queries. The page size is 4096 Bytes and the effective page capacity is 41.0 objects per page in all the index structures.

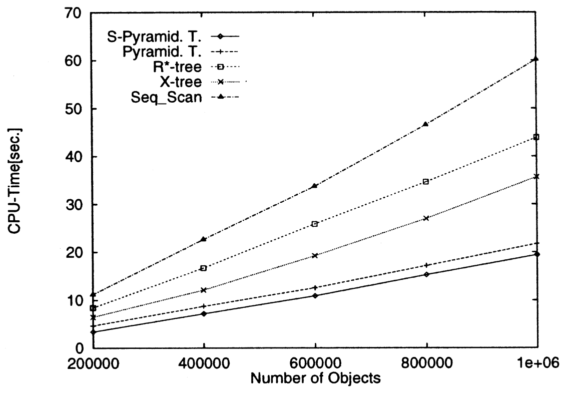
Fig. 8 shows the results of our first experiment. The (d) of Fig. 8 shows the radii of query circles used in this experiment and their average result set sizes. The speed-up with respect to the number of page accesses seems to be almost constant and ranges between 1.10 and 1.13 over the Pyramid-

Table 1
Average result set size over the radius (ε) for each DB size

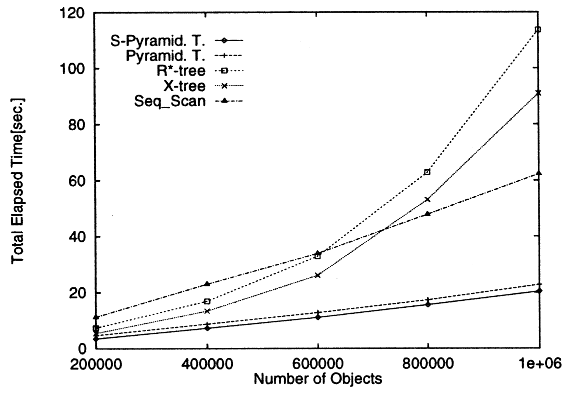
DB size	Radius (ε)						
	0.3	0.4	0.5	0.6	0.7	0.8	0.9
200,000	1.02	1.04	1.08	2.18	10.92	54.44	224.98
400,000	1.04	1.06	1.29	3.41	19.04	104.24	447.67
600,000	1.07	1.12	1.58	5.13	29.27	160.56	681.74
800,000	1.09	1.17	1.62	6.31	38.48	209.92	899.71
1,000,000	1.13	1.23	1.69	7.23	47.81	260.46	1118.59



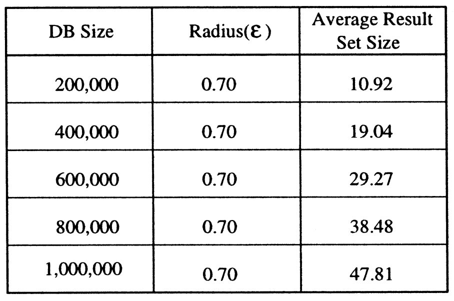
(a) Page Accesses



(b) CPU Time(sec.)



(c) Total Elapsed Time(sec.)



DB Size	Radius(ε)	Average Result Set Size
200,000	0.70	10.92
400,000	0.70	19.04
600,000	0.70	29.27
800,000	0.70	38.48
1,000,000	0.70	47.81

(d) Average Result Set Size

Fig. 8. Performance behavior over database size.

Technique, between 1.45 and 1.49 over the R*-tree, between 1.29 and 1.36 over the X-tree, and between 2.14 and 2.15 over the sequential scan. The speed-up in CPU time is higher than the speed-up in page accesses, but is only increasing with growing database sizes.

The CPU time used for processing the query has been almost exhausted in the refinement step in both the SPY-TEC and the Pyramid-Technique. Although the algorithms to process the query in the SPY-TEC are more complex than those of the Pyramid-Technique, we were able to save a large amount of CPU time by filtering the candidate before the refinement step.

Finally, most important is the speed-up in total elapsed time. It is higher than the speed-up in the number of page accesses and CPU time so that reaches its highest value with the largest database. The SPY-TEC with one million objects performs hyperspherical range queries 1.12 times faster than the Pyramid-Technique, 5.58 times faster than R*-tree, 4.47 times faster than the X-tree, and 3.06 times faster than the sequential scan.

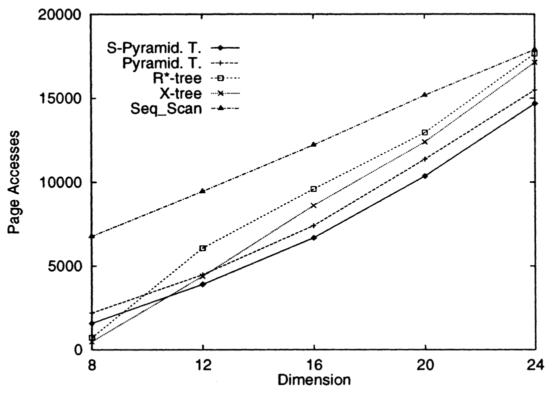
Range query processing on B$^+$-tree can be performed much more efficient than on index structures based on the R*-tree because large parts of the tree can be traversed efficiently by following the side links in the data pages, and long-distance seek operations including expensive disk head movements have a lower probability due to better disk clustering possibilities [14]. Specially, when processing hyperspherical range query, the SPY-TEC is more efficient than the Pyramid-Technique because its spherical split strategy is more suitable for the shape of queries than the right-angled split strategy of the Pyramid-Technique.

In our second experiment, we measured the performance of query processing while we varied data space dimension. For this experiment, we created five files with the dimensionalities 8, 12, 16, 20 and 24. The database size fixed at 500,000 objects. The page size is 4096 Bytes as in the first experiment and the effective data page capacity depends on the dimension and ranges from 28 to 74 objects per page. Before our second experiment, we measured the average result set size with varying the radius for each data space dimension as in the first experiment. Table 2 shows the result of this procedure. As depicted in Table 2, the average result set sizes are changed a lot with varying the data space dimension and the radius of query circle. Thus, it is not appropriate to use the same radii for each data space dimension as in the first experiment. Therefore, we chose the radii of query circles so that the average result set sizes would be very similar (about 20). The (d) of Fig. 9 shows the radii of query circles used in this experiment and their average result set sizes. We performed hyperspherical range queries with 100 query points and a constant radius for each data space dimension.

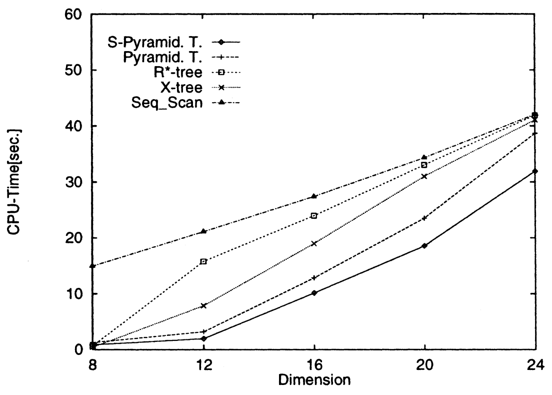
Fig. 9 shows the results of our second experiment. We observed that the R*-tree is more efficient than our technique, including the Pyramid-Technique, and sequential scan in eight-dimensional data spaces, but rapidly decreases with increasing dimension up to the 12-dimensional

Table 2
Average result set size over the radius (ε) for each data space dimension

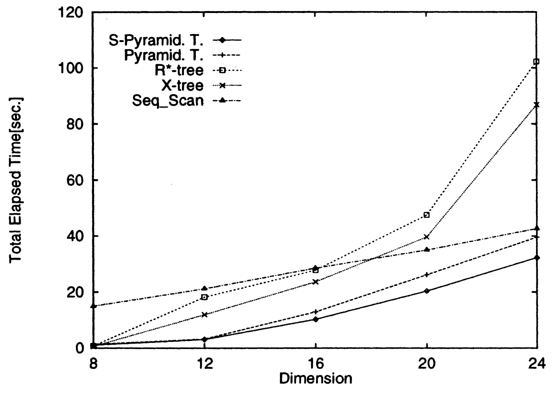
Dimension	Radius (ε)						
	0.3	0.4	0.5	0.6	0.7	0.8	0.9
8	55.92	432.84	1997.99	6702.37	17694.53	39077.42	74947.06
12	1.14	4.53	30.37	192.72	868.08	3023.17	8804.22
16	1.02	1.12	1.38	4.33	25.09	131.72	558.6
20	1.01	1.01	1.02	1.09	1.32	4.37	23.26
24	1.01	1.01	1.01	1.02	1.07	1.12	1.29



(a) Page Accesses



(b) CPU Time(sec.)



(c) Total Elapsed Time(sec.)

Dimension	Radius(ε)	Average Result Set Size
8	0.26	20.91
12	0.48	20.84
16	0.69	21.00
20	0.89	19.78
24	1.07	21.75

(d) Average Result Set Size

Fig. 9. Performance behavior over data space dimension.

data space. From this point, the page accesses are growing linearly with the index size. In the case of the X-tree, a phenomenon analogous to that of the R*-tree has appeared except that its efficiency rapidly decreases in 16-dimensional data space. However, the SPY-TEC and Pyramid-Technique have no rapid deterioration of the performance even though their performance decreases slowly with increasing dimension. We also observed that the SPY-TEC clearly outperforms the Pyramid-Technique and the sequential scan in all cases.

The speed-up in CPU time is analogous to the speed-up in page accesses, but is higher than it. Finally, in total elapsed time, the SPY-TEC performs hyperspherical range queries 1.23 times faster than the Pyramid-Technique, 3.18 times faster than the R*-tree, 2.69 times faster than the X-tree, and 1.32 times faster than the sequential scan when the dimension is 24.

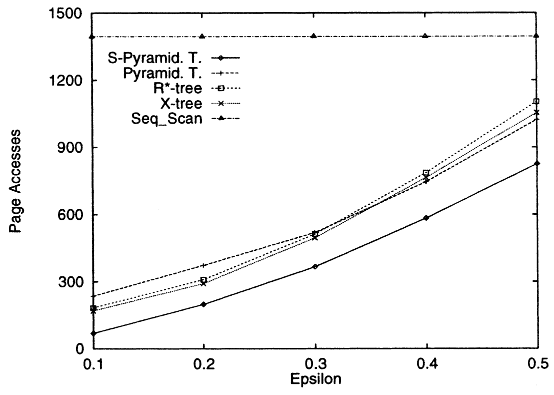
Through our second experiment, we observed that the performance of the R*-tree or the X-tree is efficient when the dimension is lower than 10, but rapidly decreases with going to higher dimension so that the sequential scan yields better performance than them. We also found out that our new method outperforms the R*-tree or the X-tree when the dimension is higher than 10, and outperforms the Pyramid-Technique and the sequential scan in all cases.

6.2. Evaluation using real data sets

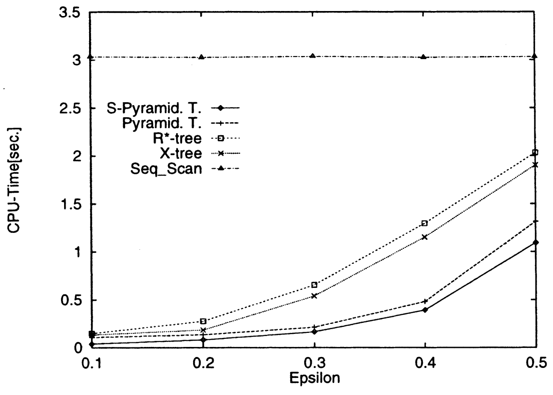
To demonstrate the practical impact of our technique for real data sets, we performed an experiment using feature vectors used for content-based image retrieval. We extracted feature vectors from 56,230 images using the wavelet transform. Thus, our real data sets contains 56,230 points in 16-dimensional data space and each point is composed of normalized wavelet coefficients.

We varied the radii of the query circles from 0.1 to 0.5 and measured the average result set size, the number of page access, CPU time, and the total elapsed time absorbed to process hyper-spherical range queries.

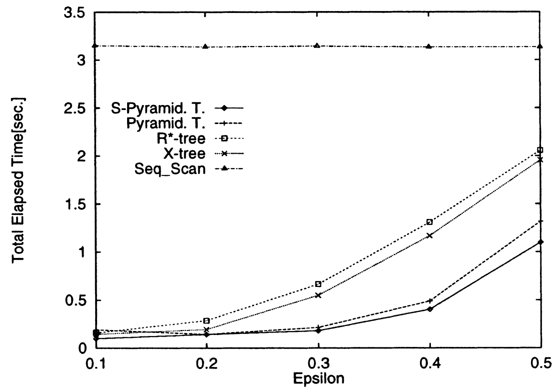
We found that the real data sets are more clustered than the uniform data distribution through (d) of Fig. 10. As depicted in Fig. 10, the SPY-TEC clearly outperforms the other index structures for any radius of query circles used in this experiment.



(a) Page Accesses



(b) CPU Time(sec.)



(c) Total Elapsed Time(sec.)

Radius(ε)	Average Result Set Size
0.1	10.21
0.2	10.43
0.3	11.32
0.4	23.97
0.5	183.69

(d) Average Result Set Size

Fig. 10. Performance behavior on real data.

The sequential scan take about 3.2 s to process queries regardless of the radius of query circles. This result is deserved because the sequential scan has to access all pages and all objects stored in database regardless of the radius. We found out that index-based query processing such as the R*-tree and X-tree outperforms the sequential scan on real data sets.

When the radius of query circle is 0.1, the SPY-TEC performs hyperspherical range queries 1.92 times faster than the Pyramid-Technique, 1.59 times faster than R*-tree, 1.45 times faster than the X-tree, and 31.5 times faster than the sequential scan. And, when the radius of query circle is 0.5, the SPY-TEC performs hyperspherical range queries 1.20 times faster than the Pyramid-Technique, 1.87 times faster than R*-tree, 1.78 times faster than the X-tree, and 2.85 times faster than the sequential scan.

Through the experiment on real data sets, we could observe that the SPY-TEC clearly outperforms the Pyramid-Technique and other index structures for the reasonable radii of query circles when processing hyperspherical range queries.

7. Conclusions

In this paper, we proposed the SPY-TEC, a new indexing technique for similarity search which is very frequently used in applications such as content-based multimedia retrieval.

The SPY-TEC is based on a special partitioning strategy which is optimized for hyperspherical range queries. This technique transforms d-dimensional data space to one-dimensional data space and manages a B^+-tree to store and access the transformed one-dimensional values as the Pyramid-Technique does, and therefore, we are able to use all of the advantages of a B^+-tree. We also showed that the SPY-TEC outperforms other related techniques including the Pyramid-Technique when processing hyperspherical range queries. For highly skewed data distributions or queries, the SPY-TEC may perform worse than other index structures. However, none of the index structure proposed so far can handle highly skewed data or queries efficiently [14]. We plan to address the problem of handling highly skewed data or queries in our future work.

A few cost models [13,12] for processing hyperspherical range queries in high-dimensional spaces have been proposed. Although these models considered boundary effects which occur frequently when processing hyperspherical queries in high-dimensional data spaces, we could not take advantage of these models directly to tackle the cost model of the SPY-TEC, because they assumed that the shape of the index pages was a rectangle or a sphere. We have found out that we need a new cost model for the SPY-TEC, which is a very complex problem. Therefore, we plan to study a new cost model for the SPY-TEC. Also, we plan to develop an efficient algorithm for another similarity query, i.e., the k-nearest neighbor query with the SPY-TEC in our future work.

Acknowledgements

We are thankful to Dr. Stefan Berchtold in AT&T Bell laboratory for providing the X-tree code, the Pyramid-Technique code, and helpful comments.

References

[1] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, W. Equiz, Efficient and effective querying by image content, Journal of Intelligent Information System 3 (3) (1994) 231–262.

[2] C.E. Jacobs, A. Finkelstein, D.H. Salesin, Fast multiresolution image query, in: Proceedings of the 1995 ACM SIGGRAPH, 1995.

[3] D.A. White, R. Jain, Similarity indexing: algorithms and performance, in: Proceedings of the SPIE Storage and Retrieval for Image and Video Databases IV, vol. 2670, 1996, pp. 62–75.

[4] D.A. White, R. Jain, Similarity indexing with the SS-tree, in: Proceedings of the 12th International Conference on Data Engineering, 1996, pp. 516–523.

[5] C. Faloutsos, Fast searching by content in multimedia databases, Data Engineering Bulletin 18 (4) (1995) 31–40.

[6] I. Kamel, C. Faloutsos, Hilbert R-tree: An improved R-tree using fractals, in: Proceedings of the 20th International Conference on Very Large Database, September 1994, pp. 500–509.

[7] K.-I. Lin, H.V. Jagadish, C. Faloutsos, The TV-tree: an index structure for high-dimensional data, VLDB Journal 3 (4) (1994) 517–542.

[8] G.E. Martin, The Foundations of Geometry and the Non-Euclidean Plane, Springer, Berlin, 1996.

[9] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, May 1990, pp. 322–331.

[10] N. Katayama, S. Satoh, The SR-tree: an index structure for high-dimensional nearest neighbor queries, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, May 1997, pp. 517–542.

[11] K. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft, When is "Nearest Neighbor" meaningful? in: Proceedings of the Seventh International Conference on Database Theory, January 1999, pp. 217–235.

[12] R. Weber, H.-J. Schek, S. Blott, A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, in: Proceedings of the 24th International Conference on Very Large Database, August 1998, pp. 194–205.

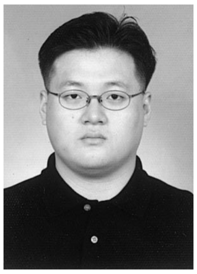
[13] S. Berchtold, C. Böhm, D.A. Keim, H.-P. Kriegel, A cost model for nearest neighbor search in high-dimensional data space, ACM PODS Symposium on Principles of Database Systems, 1997, pp. 78–86.

[14] S. Berchtold, C. Böhm, H.-P. Kriegel, The Pyramid-Technique: towards breaking the curse of dimensionality, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1998, pp. 142–153.

[15] S. Berchtold, D.A. Keim, H.-P. Kriegel, The X-tree: an indexing structure for high-dimensional data, in: Proceedings of the 22nd International Conference on Very Large Database, September 1996, pp. 28–39.

[16] S. Berchtold, D. Keim, H.-P. Kriegel, T. Seidl, Fast nearest neighbor search in high-dimensional spaces, in: Proceedings of the 14th International Conference on Data Engineering, 1998, pp. 209–218.

[17] P.M. Kelly, T.M. Cannon, D.R. Hush, Query by image example: the CANDID approach, in: Proceedings of the SPIE Storage and Retrieval for Image and Video Databases III, vol. 2420, 1995, pp. 238–248.



Dong-Ho Lee received his BS degree from Hong-Ik University, and MS degree in computer engineering from Seoul National University, Seoul, Korea, in 1995 and 1997, respectively. He is currently enrolled in a Ph.D. program in computer engineering at Seoul National University. His research interests include high-dimensional index techniques, content-based retrieval system, multimedia databases, and object-oriented databases.

Hyoung-Joo Kim received his BS degree in computer engineering from Seoul National University, Seoul, Korea, in 1982 and his MS and Ph.D. in computer engineering from University of Texas at Austin in 1985 and 1988, respectively. He was an assistant professor of Georgia Institute of Technology, and is currently a professor in the Department of Computer Engineering at Seoul National University. His research interests include object-oriented databases, multimedia databases, HCI, and computer-aided software engineering.