Information Processing Letters 83 (2002) 115–123

Information Processing Letters

www.elsevier.com/locate/ipl

An efficient algorithm for hyperspherical range query processing in high-dimensional data space

Dong-Ho Lee [a,*], Shin Heu [b], Hyoung-Joo Kim [a]

[a] *Department of Computer Engineering, Seoul National University, San 56-1, Shilim-dong, Gwanak-gu, Seoul, 151-742 Republic of Korea*

[b] *Department of Computer Science, Hanyang University, Seoul, Republic of Korea*

Received 7 January 2000; received in revised form 25 August 2001

Communicated by F.Y.L. Chin

Keywords: Algorithms; Similarity search; Range query; High-dimensional index structure

1. Introduction

Recently, similarity search on high-dimensional feature vectors has become an important search paradigm for various multimedia retrieval applications. The technique used is to map the data items as points into a high-dimensional feature space. The feature space is usually indexed using a multi-dimensional index structure. Similarity search then corresponds to the hyperspherical range query that returns all objects within a threshold level of similarity to the query object [1]. The hyperspherical range query for similarity search can be defined as follows:

Definition 1 (*Hyperspherical range query*). Given a query point (Q) and a threshold (ε) of similarity, find the points in the database (DB) which have a distance (D) smaller than or equal to ε from Q. More formally:

$$\mathit{HypersphericalRangeQuery}(DB, Q, \varepsilon, D) = \{P \in DB \mid D(Q, P) \leqslant \varepsilon\}.$$

One of the most popular applications using this technique is a content-based image retrieval system [2] (so-called CBIR system) which extracts several features from images, indexes them based on those features and supports similarity queries based on them. Examples of feature spaces indexed in a CBIR system include color histograms, color moments, texture vectors, shape descriptions, etc. To support efficient similarity search in such a system, robust techniques to index high-dimensional feature spaces needs to be developed because features used are usually high-dimensional points.

However, basically none of the querying and indexing techniques which provide good results on low-dimensional data also perform sufficiently well on high-dimensional data [7,8]. Many researchers have called this problem the "*curse of dimensionality*" [2] and have tried to tackle it. As a result of these research efforts, a variety of new index structures, such as the TV-tree [3], X-tree [9], SS-tree [1] and SR-tree [6], have been proposed. However, most of these index structures are extensions of multi-dimensional index structures adapted to the requirements of high-dimensional indexing [8]. Thus, all of these index structures are limited with respect to the data space

[*] Corresponding author.

E-mail addresses: dhlee@oopsla.snu.ac.kr (D.-H. Lee), shinheu@cse.hanyang.ac.kr (S. Heu), hjk@oopsla.snu.ac.kr (H.-J. Kim).

PII: S0020-0190(01)00318-0

partitioning and suffer from the well-known drawbacks of multi-dimensional index structures, such as high costs for insert, delete and search operations [8].

To overcome these drawbacks, in this paper, we propose a special partitioning strategy, the Spherical Pyramid-Technique (so-called SPY-TEC), which divides the d-dimensional data space first into $2d$ spherical pyramids, and then cuts the single spherical pyramid into several spherical slices. This partition provides a transformation of d-dimensional data space into 1-dimensional value. Therefore, we can use the B^+-tree to manage fast insert, update and delete operations on high-dimensional data. We also propose the algorithm for processing hyperspherical range queries on the space partitioned by this strategy.

2. The SPY-TEC

In [8], Berchtold et al. proposed a special partitioning strategy, the Pyramid-Technique, which divides the d-dimensional data space first into $2d$ pyramids and then cuts the single pyramid into several slices. They also proposed the algorithm for processing hypercubic range queries on the space partitioned by this strategy. However, the shape of queries used in similarity search is not a hypercube, but a hypersphere [2]. Thus, when processing hyperspherical range queries with the Pyramid-Technique, there is a drawback which exists in all index structures based on the bounding rectangle [2,4].

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 1-dimensional values and then store a d-dimensional point *plus* the corresponding 1-dimensional key as a record in the leaf nodes of the B^+-tree. 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.

2.1. Data space partitioning

The SPY-TEC partitions the data space in two steps: In the first step, 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 spherical 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. 1 shows the data space partitioning of the SPY-TEC in a 2-dimensional example. First, the 2-dimensional data space has been divided into 4 spherical pyramids. In the second step, each of these 4 spherical pyramids is split again into several data pages which are shaped like the annual rings of a tree. Given a d-dimensional space instead of the 2-dimensional space, the base of the spherical pyramid is not a 1-dimensional curved line, but a $(d-1)$-dimensional spherical surface. As a sphere of dimension d has $2d$ $(d-1)$-dimensional spherical surface, we obviously obtain $2d$ spherical pyramids [8].

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 containing the point v. If v_i is greater or equal to 0.5, then the spherical pyramid containing the point v is $sp_{(i+d)}$. If it is smaller than 0.5, the spherical pyramid containing the point v is sp_i. As depicted in Fig. 2(I), the value of $|0.5 - v_1|$ of a 2-dimensional point v 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

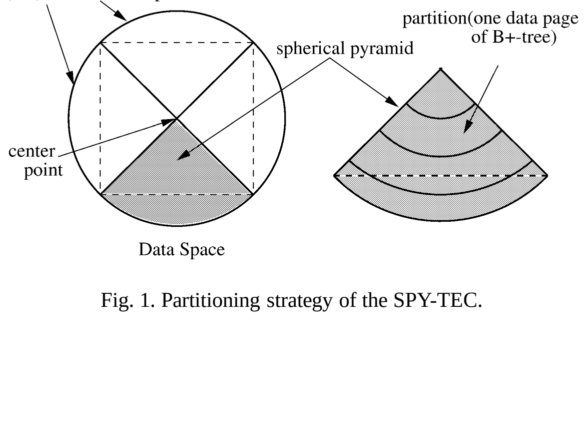
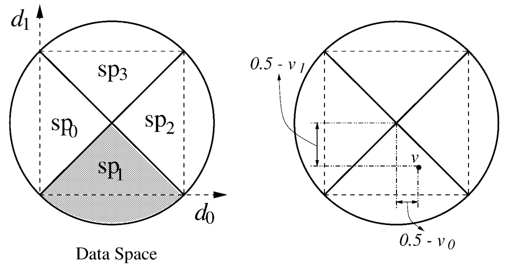
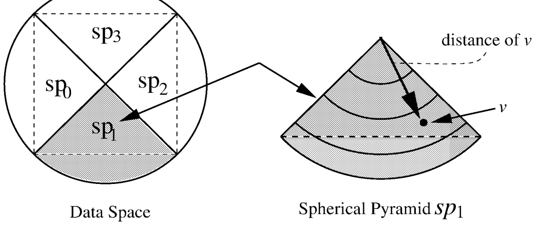

(d-1)-dimensional spherical surface
spherical pyramid
partition(one data page of B+-tree)
center point
Data Space

Fig. 1. Partitioning strategy of the SPY-TEC.



(I) Numbering of Spherical Pyramids



(II) Distance of a Point within it's Spherical Pyramid

Fig. 2. The SPY-TEC.

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 [8], we redefine it formally to help understanding the partitioning strategy of the SPY-TEC.

Definition 2 (*Spherical pyramid of a point v*). A d-dimensional point v is defined to be located in spherical pyramid sp_i.

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

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

In Definition 2, $j_{\max}$ 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 containing v. In order to transform d-dimensional data into a 1-dimensional value, we have to determine the location of a point v within its spherical pyramid. By Definition 3, we can determine the location of a point (refer to Fig. 2(II)).

Definition 3 (*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 2 and 3, we are able to transform a d-dimensional point v into a 1-dimensional value $(i \cdot \lceil\sqrt{d}\rceil + d_v)$. More formally:

Definition 4 (*Spherical pyramid value of a point v*). Given a d-dimensional point v, let sp_i be the spherical pyramid containing v according to Definition 2, and d_v be the distance of v according to Definition 3. Then, the spherical pyramid value spv_v of v is defined as

$$spv_v = \big(i \cdot \lceil\sqrt{d}\rceil + d_v\big).$$

If we consider the same example point $v' = (0.8, 04)$, the distance of v' is $\sqrt{0.1}$ by Definition 3. Therefore, the spherical pyramid value of the point v' is $(2 \cdot \lceil\sqrt{2}\rceil + \sqrt{0.1})$ according to Definition 4.

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 $\lceil\sqrt{d}\rceil$ 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 $[i \cdot \lceil\sqrt{d}\rceil, (i \cdot \lceil\sqrt{d}\rceil + 0.5\sqrt{d})]$. 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 $\lceil\sqrt{d}\rceil$. If we would not multiply i by $\lceil\sqrt{d}\rceil$, 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 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 $\lceil\sqrt{d}\,\rceil$.

2.2. *Index creation*

It is a very simple task to build an index using the SPY-TEC. 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.

3. Query processing

The geometric correspondence of hyperspherical range queries for similarity search is a hypersphere having the query point as the center point and ε as the radius of the query circle as depicted in a 2-dimensional example of Fig. 3.

We process a hyperspherical range query in two main steps: In the first step, we have to determine which spherical pyramids intersect the query circle, and then in the second step, we have to determine the ranges inside the spherical pyramids. In Fig. 3, the query circle intersects sp_0 and sp_1. Therefore, we have to determine the range $[d_{\text{low}}, d_{\text{high}}]$ inside sp_0 and sp_1, respectively. Points lying between d_{low} and d_{high} are candidates for a further investigation. Some of the

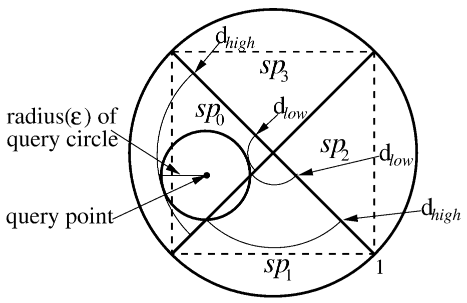


Fig. 3. Two-dimensional example of hypersphere range queries.

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.

The following Lemma 1 corresponds to the first step. 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 straightforward manner.

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 *as the radius of the query circle, let* j ($j < d$) *be the number of a spherical pyramid containing a query point, 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*

(1) ($i = j$): *In this case, the spherical pyramid* sp_i *always intersects the query circle because* sp_j *is a spherical pyramid into which the query point falls.*

(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.*

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

(3) ($i < d$):

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

(4) ($i \geqslant d$):

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

Proof. Due to lack of space, we only show each case by using a 2-dimensional example of Fig. 4 instead of the formal proof. First, the spherical pyramid sp_0 intersects the query circle because the query point falls into sp_0 (case (1)). And, sp_1 also intersects the query circle because the distance (α) from the query point to the closest side plane of sp_1 is smaller than ε (case (3)). sp_2 does not intersect the query circle because ε is smaller than the distance (β) from the query point to the center of the data space (case (2)). Finally, sp_3 also does not intersect the query circle because the distance (α) from the query point to the

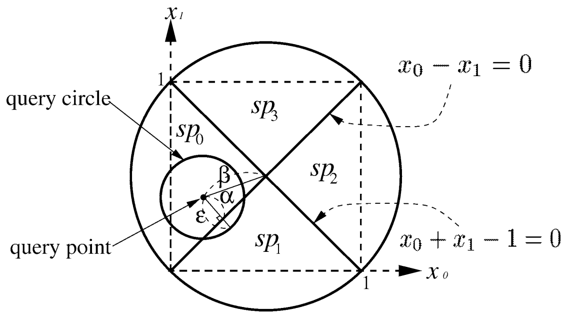


Fig. 4. Intersection of a spherical pyramid and a query circle.

closest side plane of sp_3 is greater than ε (case (4)). For the formal proof, you can refer to [4]. □

The following Lemma 2 corresponds to the second step. The basic idea of 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* [5].

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

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

(1.1) ($i = j$): *This case corresponds to* (a) *of Fig.* 5(I). *Let β be the distance from the center point to the query point and ε be the radius of the query circle.*

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

(1.2) ($|i - j| = d$): *This case corresponds to* (c) *of Fig.* 5(I). *Let α be the maximum value of the distances from the query point to the closest side plane of all 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.*

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

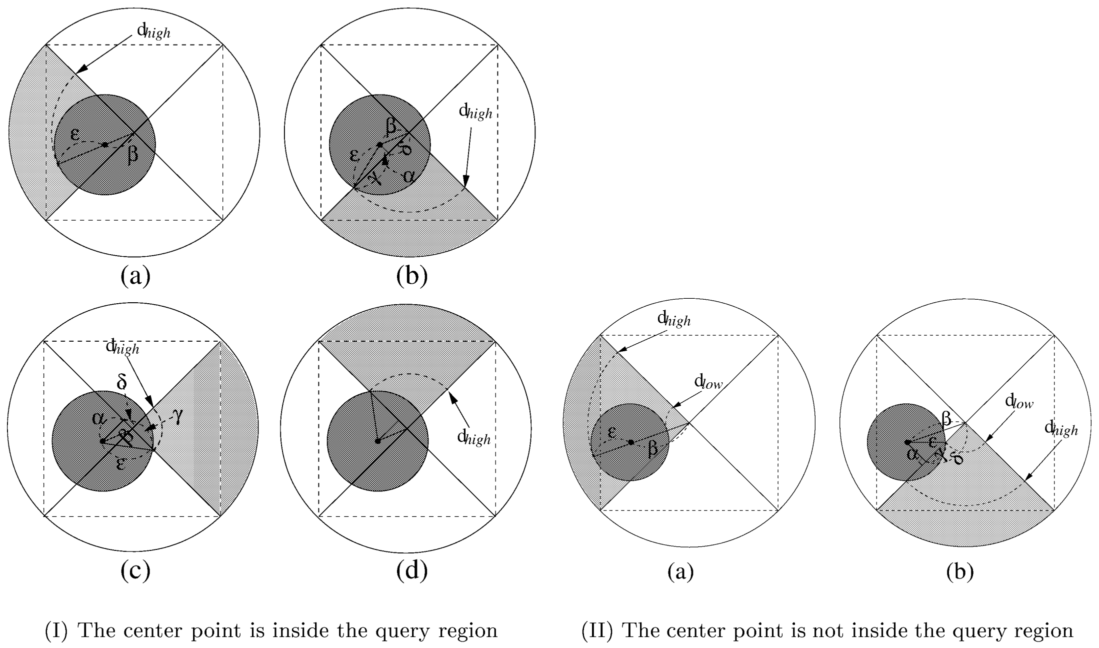


Fig. 5. Interval of intersection of query and spherical pyramid.

(1.3) (otherwise): *This case corresponds to* (b) *or* (d) *of Fig.* 5(I). *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* (1.2).

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

(2) *The case in which the center point of the data space is not inside the query region.*

(2.1) $(i = j)$: *This case corresponds to* (a) *of Fig.* 5(II).

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

(2.2) $(i \neq j)$: *This case corresponds to* (b) *of Fig.* 5(II).

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

Proof. Due to lack of space, we would prove only (1.1). Subcase (1.1) corresponds to (a) of Fig. 5(I). Let v be the point which lies inside the query region and an intersected spherical pyramid sp_i. Then, $0 \leqslant d_v \leqslant \beta + \varepsilon$. Therefore, $d_{\text{low}} = 0 \leqslant d_v \leqslant d_{\text{high}} = \beta + \varepsilon$. The rest of the cases of Lemma 2 can be proved analogously. For the formal proof, you can refer to [4], too. □

With Lemmas 1 and 2, we can process the hyperspherical range query by Algorithm 1. In line 5 of Algorithm 1, we test that a spherical pyramid sp_i intersects the query circle having *qp* as the query point and ε as the radius of the query circle. If sp_i intersects the query circle, we determine the interval $[d_{\text{low}}, d_{\text{high}}]$ in line 6. Using this interval, we perform 1-dimensional interval query on the B^+-tree in line 7. 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 that this task consumes a lot of CPU time. Thus, before starting the refinement step, we perform the filtering step, which tests whether or not the ith 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. In line 9, the function named *Inside_DiameterOfCircle* corresponds to the filtering step. And, in line 10, the function named *Point_In_Circle* corresponds to the refinement step.

```
 1: INPUT: $qp$ ← query point, $\varepsilon$ ← range
 2: OUTPUT: set of points satisfying the query (result)
 3:
 4: for $i = 0$ to $2d - 1$
 5:   if Intersect($sp[i], qp, \varepsilon$) then   /* Using Lemma 1 */
 6:     Determine_Interval($sp[i], qp, \varepsilon, d_{\text{low}}, d_{\text{high}}$);   /* Using Lemma 2 */
 7:     cs = Btree_Interval_Query($i \cdot \lceil\sqrt{d}\,\rceil + d_{\text{low}}, i \cdot \lceil\sqrt{d}\,\rceil + d_{\text{high}}$);
 8:     for $c$ = cs.first to cs.end do
 9:       if Inside_DiameterOfCircle($c, qp, \varepsilon$) then   /* filtering step */
10:         if Point_In_Circle($c, qp, \varepsilon$) then   /* refinement step */
11:           result ← $c$;
12:         endif
13:       endif
14:     endfor
15:   endif
16: endfor
```

Algorithm 1. Processing the hyperspherical range query.

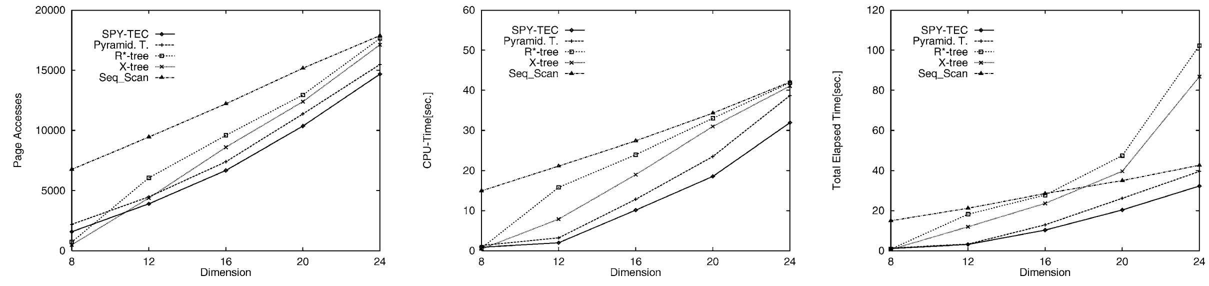
4. Performance evaluation

To show the practical impact of the SPY-TEC, we compared it to other index structures such as the Pyramid-Technique, the R*-tree, the X-tree and the sequential scan. We performed our experiments using both real and synthetic data sets on the SUN SPARC 20 workstation with 128 MByte main memory and 10 GByte secondary storage. The page size is 4 KByte. We used 100 query points which were selected from the data set itself. Thus, the result was evaluated as the average of 100 random trials.

In our first experiment, we measured the number of page access, CPU time and the total elapsed time absorbed in processing hyperspherical range queries on uniformly distributed points while we varied the data space dimension. For this experiment, we created 5 files with the dimensionalities 8, 12, 16, 20, and 24 and set the database size to 500,000 points.

Fig. 6 shows the result of our first experiment. For a consistent condition of the experiment, we chose the radii of query circles so that the average result set sizes would be very similar (about 20) through preliminary experiments. Fig. 6(d) show the radii of the query circles used in this experiment and their average result set size for each dimension. We observed that the R*-tree is more efficient than our technique, including the Pyramid-Technique, and the sequential scan in 8-dimensional data spaces, but rapidly decreases with increasing dimension up to the 12-dimensional 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 the 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.

In our second experiment, we measured the performance behavior of the query processing on real data sets. For this experiment, we extracted feature vectors from 56,230 images using the wavelet transform. Thus, our real data sets contain 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

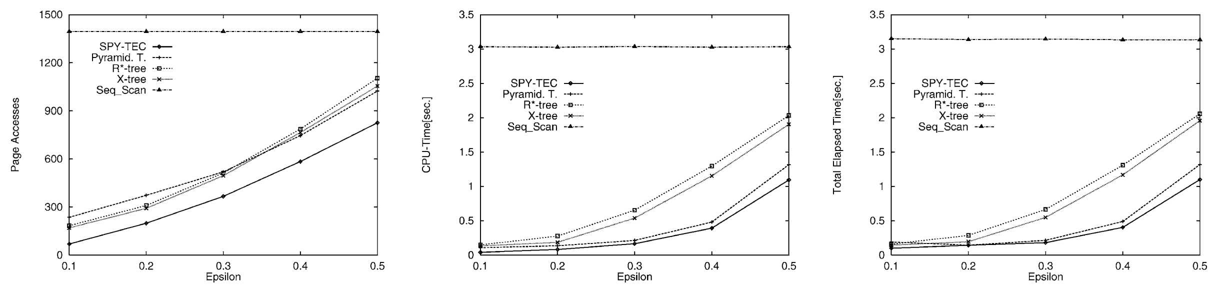


(a) (b) (c)

Dimension (radius)	8 (0.26)	12 (0.48)	16 (0.69)	20 (0.89)	24 (1.07)
Average result set size	20.91	20.84	21.00	19.78	21.75

(d)

Fig. 6. Performance behavior on uniformly distributed data. (a) Page accesses, (b) CPU time (sec), (c) total time (sec), (d) average result set size.



(a) (b) (c)

Radius (ε)	0.1	0.2	0.3	0.4	0.5
Average result set size	10.21	10.43	11.32	23.97	183.69

(d)

Fig. 7. Performance behavior on real data. (a) Page accesses, (b) CPU time (sec), (c) total time (sec), (d) average result set size.

time and the total elapsed time absorbed in processing hyperspherical range queries.

Fig. 7(d) shows the radii of the queries used in the experiment on real data and the average result set size. As depicted in Fig. 7, the SPY-TEC clearly outperforms the other index structures for any radius of query circles used in this experiment. The sequential scan take about 3.2 sec 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 the 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 the 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 the R*-tree, 1.78 times faster than the X-tree, and 2.85 times faster than the sequential scan.

Range query processing on a B^+-tree can be performed much more efficiently 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 [8]. Through the experiments on real data sets, we could observe that the SPY-TEC clearly outperforms other related index structures for the reasonable radii of query circles when processing hyperspherical range queries.

5. 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. We also showed that the SPY-TEC outperforms other related techniques including the Pyramid-Technique when processing hyperspherical range queries in high-dimensional data space.

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 [8]. We plan to address the problem of handling highly skewed data or queries in our future work. We also 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. This research was partially supported by the 1999 BK21 IT area grant of Korea.

References

[1] D.A. White, R. Jain, Similarity indexing with the SS-tree, in: Proc. 12th Internat. Conf. on Data Engineering, 1996, pp. 516–523.

[2] C. Faloutsos, Fast searching by content in multimedia databases, Data Engrg. Bull. 18 (4) (1995) 31–40.

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

[4] D.-H. Lee, H.-J. Kim, SPY-TEC: An efficient indexing method for similarity search in high-dimensional data spaces, SNU OOPSLA Technical Report, Seoul, 1999; http://oopsla.snu.ac.kr/~dhlee/spy-tec.ps.

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

[6] N. Katayama, S. Satoh, The SR-tree: An index structure for high-dimensional nearest neighbor queries, in: Proc. ACM SIGMOD Internat. Conf. on Data Management of Data, May 1997, pp. 517–542.

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

[8] S. Berchtold, C. Böhm, H.-P. Kriegel, The Pyramid-Technique: Towards breaking the curse of dimensionality, in: Proc. ACM SIGMOD Internat. Conf. on Management of Data, 1998, pp. 142–153.

[9] S. Berchtold, D.A. Keim, H.-P. Kriegel, The X-tree: An indexing structure for high-dimensional data, in: Proc. 22nd Internat. Conf. on Very Large Database, September 1996, pp. 28–39.