An Algorithm for Incremental Nearest Neighbor Search in High-Dimensional Data Spaces*

Dong-Ho Lee, Hyung-Dong Lee, Il-Hwan Choi, and Hyoung-Joo Kim

OOPSLA Laboratory,
School of Computer Science and Engineering,
Seoul National University,
Shilim-Dong Gwanak-Gu, Seoul 151-742, KOREA
{dhlee,hdlee,ihchoi,hjk}@oopsla.snu.ac.kr

Abstract The SPY-TEC (Spherical Pyramid-Technique) [8] was proposed as a new indexing method for high-dimensional data spaces using a special partitioning strategy that divides a d-dimensional data space into $2d$ spherical pyramids. Although the authors of [8] proposed an efficient algorithm for processing hyperspherical range queries, they did not propose an algorithm for processing k-nearest neighbor queries that are frequently used in similarity search. In this paper, we propose an efficient algorithm for processing exact nearest neighbor queries on the SPY-TEC by extending the incremental nearest neighbor algorithm proposed in [10]. We also introduce a metric that can be used to guide an ordered best-first traversal when finding nearest neighbors on the SPY-TEC. Finally, we show that our technique significantly outperforms the related techniques in processing k-nearest neighbor queries by comparing it to the R*-tree, the X-tree, and the sequential scan through extensive experiments.

Keywords : *Similarity Search, High-Dimensional Index Technique, Nearest Neighbor Query, Incremental Nearest Neighbor Algorithm, Approximate Nearest Neighbor Algorithm, SPY-TEC*

1 Introduction

Feature-based similarity search has become an important search paradigm for various database applications such as multimedia retrieval, data mining, decision support, and statistical and medical 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 multidimensional index structure. Similarity search then corresponds to a hyperspherical range search, which returns all objects within a threshold level of similarity to the query objects, and a k-nearest neighbor search that returns the k most similar objects to the query object. One of the most popular applications using this technique is a content-based image indexing

* This research was partially funded by the 1999 BK21 IT area grant of the Ministry of Education in Korea.

and retrieval system [3–5, 12] which extracts several features (such as color, texture, shape, etc) from images, indexes the images based on those features, and supports similarity queries based on them. To support efficient similarity search in such a system, robust techniques to index high-dimensional feature spaces need to be developed because the feature vectors used are high-dimensional.

Initially, traditional multidimensional data structures (e.g., R-tree [1], kd-tree [11]), which were designed for indexing low-dimensional spatial data, were used for indexing high-dimensional feature vectors. However, recent research activities [19–21] reported the result that basically none of the querying and indexing techniques which provide good results on low-dimensional data also performs sufficiently well on high-dimensional data. Many researchers have called this problem the *"curse of dimensionality"* [9], and many database-related projects have tried to tackle it. As a result of these research efforts, a variety of new index structures [20, 22], cost models [21] and query processing techniques [18] have been proposed. However, most of the high-dimensional index structures are extensions of the R-tree or the kd-tree adapted to the requirements of high-dimensional indexing. Thus, all of these index structures are limited with respect to data space partitioning and suffer from specific drawbacks of the R-tree or the kd-tree.

For example, most of the R-tree-based index structures, such as the TV-tree [14], X-tree [22], SS-tree [6], and SR-tree [17], tend to have low fanouts and a high degree of overlap between bounding regions in higher dimensions. These degrade the performance of query processing in high-dimensional data spaces. Although the X-tree uses a modified R-tree node splitting algorithm to reduce overlap among the index nodes, it has the overhead of performing disk management operations to create and maintain variable sized nodes (so-called supernodes) produced by this modified splitting algorithm. Also, most of the kd-tree-based index structures, such as the KDB-tree [13], hB-tree [7], and LSDh-tree [2], suffer from such problems as no guaranteed utilization (e.g., KDB-tree) or require storage of redundant information (e.g., hB-tree). In addition to the above drawbacks, these index structures have the well-known drawbacks of multidimensional index structures, such as high costs for insert and delete operations and a poor support of concurrency control and recovery [8].

To overcome these drawbacks, in our earlier work, we proposed a new special space partitioning strategy, the SPY-TEC [8], which is optimized for similarity search in high-dimensional spaces, and proposed the algorithms for processing hyperspherical range queries on the data space partitioned by this strategy. The SPY-TEC first partitions 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 bases, and then cuts each spherical pyramid into several spherical slices. By this partitioning strategy of the SPY-TEC, we were able to transform the given d-dimensional data space into a 1-dimensional value. Thus, we could use a B^+-tree to store and access data items, and take advantage of all of the benefits of a B^+-tree, such as fast insert, update and delete operations, and good concurrency control and recovery. However, we could not

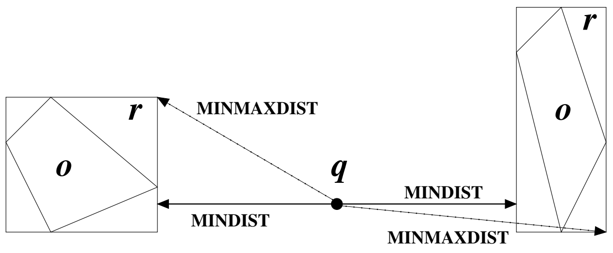


Fig. 1. An example of MinDist and MinMaxDist

propose an algorithm for processing nearest neighbor queries efficiently on the SPY-TEC.

In this paper, we introduce a new metric that can be used to guide an ordered best-first traversal when finding nearest neighbors on the SPY-TEC. Based on this new metric, we propose the incremental nearest neighbor algorithm on the SPY-TEC.

The rest of this paper is organized as follows. Section 2 discusses major algorithms related to nearest neighbor queries. Section 3 briefly reviews the structure of the SPY-TEC. Section 4 describes the incremental nearest neighbor algorithm on the SPY-TEC. Section 5 presents the results of an empirical study comparing our technique with the R*-tree, the X-tree and the sequential scan. Finally, we conclude our work and describe our future plans in Section 6.

2 Related Work

There are numerous algorithms for answering nearest neighbor or k-nearest neighbor queries that are motivated by the importance of these queries in fields including geographical information systems (GIS), document retrieval, pattern recognition, and learning theory [10]. Many of the above algorithms require specialized search structures, but some employ commonly used spatial structures. For example, algorithms exist for the k-d tree, quadtree-related structures, the R-tree, and others. Of these algorithms, there are two major approaches that provide a basis for our work. One was published by Roussopoulos, et al. [18] and we call it the *KNN algorithm* because it was intended for general nearest neighbor or $\underline{k}$-$\underline{n}$earest $\underline{n}$eighbor queries. The other algorithm was published by Hjaltason and Samet [10]. We call it the *INN algorithm* because it used the $\underline{i}$ncremental $\underline{n}$earest $\underline{n}$eighbor approach. Due to their importance for our work, these algorithms are presented in detail.

In the *KNN algorithm*, the authors proposed an approach for a nearest neighbor search in the R-tree. The key idea of their work is to maintain a global list (*ActiveBranchList*) of the candidate k nearest neighbors as the R-tree is traversed in a depth-first manner. The authors introduced two important distance functions, MinDist and MinMaxDist for ordering nodes that will be visited.

MinDist is the distance from the query point q to the closest point on the boundary of a bounding rectangle r of node n, while MinMaxDist is the distance from q to the closest corner of r that is "adjacent" to the corner farthest from q. Figure 1 shows two examples of the calculation of MinDist and MinMaxDist which are shown with a solid and a broken line, respectively. With these distance functions, the authors proposed three strategies for upward and downward pruning. In some sense, the two orderings represent the optimistic (MinDist) and the pessimistic (MinMaxDist) ordering choices because experiments reported in [18] showed that ordering the *ActiveBranchList* using MinDist consistently performed better than using MinMaxDist. Since MinDist represents the minimum distance from a query object q to a bounding rectangle r, it is the most optimistic ordering choice possible. Thus, it provides a means of pruning nodes from the search, given that a bound on the maximum distance is available. On the other hand, MinMaxDist is an upper bound on the distance of the object o nearest to q. Therefore, it should be clear that MinMaxDist by itself does not help in pruning the search, as objects closer to q could be found in elements of n at positions with higher MinMaxDist values [10].

In the *INN algorithm*, the authors proposed the incremental nearest neighbor algorithm that employs what may be termed best-first traversal. When finding k nearest neighbors to the query object using the *KNN algorithm*, k is known prior to the invocation of the algorithm. Thus, if the $(k+1)-th$ neighbor is needed, the k-nearest neighbor algorithm needs to be reinvoked for $(k+1)$ neighbors from scratch. To resolve this problem, the authors of the *INN algorithm* proposed the concept of *distance browsing* which is to obtain the neighbors incrementally (i.e., one by one) as they are needed. This operation means browsing through the database on the basis of distance. They showed through various experiments that the *INN algorithm* significantly outperforms the *KNN algorithm* for distance browsing queries and also usually outperforms the *KNN algorithm* when applied to the k-nearest neighbor problem for the R-tree. They also showed that the two pruning strategies proposed in [18] are only useful when finding the first nearest neighbor, and the one strategy that does not use MinMaxDist is sufficient when used in a combination of upward and downward pruning in their algorithm. This implies that MinMaxDist is not necessary for pruning in the incremental nearest neighbor approach.

To the best of our knowledge, the *INN algorithm* is one of the most efficient algorithms for finding the nearest neighbor or k nearest neighbors. However, this algorithm does not provide good results on high-dimensional data either, as we will show in our experimental evaluation. This is not a problem of the *INN algorithm* itself, but a problem of the spatial index structure (R-tree), which does not support efficient indexing or query processing structurally on a high-dimensional data space.

In this paper, we propose a new metric that can be used to guide an ordered best-first traversal when finding nearest neighbors on the SPY-TEC. We also propose an efficient incremental nearest neighbor algorithm based on this new metric on the SPY-TEC.

3 The SPY-TEC

In [20], Berchtold et al. proposed a special partitioning strategy (Pyramid-Technique) that divides the data space first into $2d$ pyramids, and then cuts each pyramid into several slices. They also proposed the algorithms 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 [3,5,9,23]. Thus, when processing hyperspherical range queries with the Pyramid-Technique, there is a drawback that exists in all index structures based on the bounding rectangle [8,9].

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. Although we

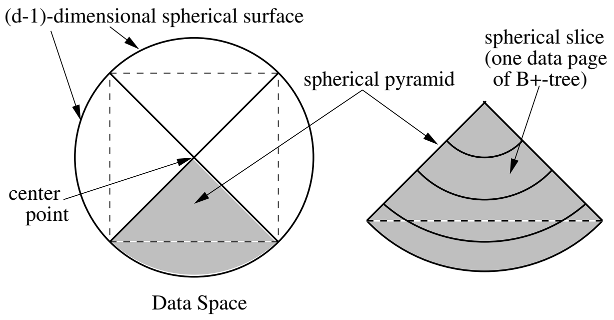

(d-1)-dimensional spherical surface
spherical slice (one data page of B+-tree)
spherical pyramid
center point
Data Space

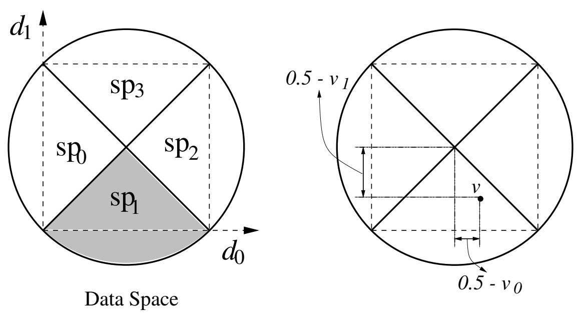
Fig. 2. Partitioning strategy of the SPY-TEC

have presented the basic idea and space partitioning strategy of the SPY-TEC in [8], we should explain it again briefly for better understanding of our incremental nearest neighbor algorithm on the SPY-TEC.

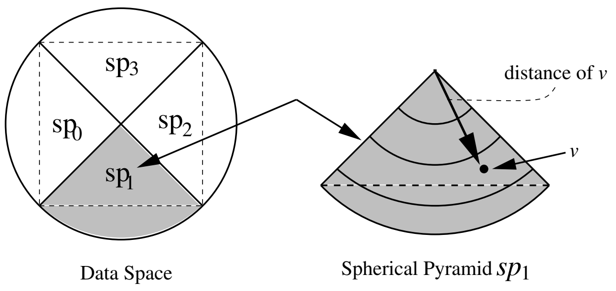
The SPY-TEC is to transform d-dimensional data points into one-dimensional values and then store and access the values using the B^+-tree. Also, we store a d-dimensional point *plus* the corresponding one-dimensional key as a record in the leaf nodes of the B^+-tree. Therefore, we do not need an inverse 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.

3.1 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, ..., 0.5) as their top and a (d-1)-dimensional curved



(a) Numbering of Spherical Pyramids



(b) Distance of a Point within its Spherical Pyramid

Fig. 3. The SPY-TEC

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 each slice corresponding to one data page of the B^+-tree. Figure 2 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 fans. Each of these spherical pyramids has the center point of the data space as its top and one curved line of the data space as its base. In the second step, each of these four 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 two-dimensional space, the base of the spherical pyramid is not a 1-dimensional curved line as in the example, but a $(d-1)$-dimensional spherical surface. As a sphere of dimension d has $2d$ $(d-1)$-dimensional spherical surface as a 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 than 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 Figure 3(a), 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 $d_0(0.3 = |0.5 - v^{'}_0| > |0.5 - v^{'}_1| = 0.1)$. Also, 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 of this procedure was presented in [8], we redefine it formally for better understanding of 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 a spherical pyramid sp_i.

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

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

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 containing v.

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. Figure 3(b) 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 image 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 Definition 1 and Definition 2, we are able to transform a d-dimensional point v into a one-dimensional value $(i \cdot ceil(\sqrt{d}) + d_v)$. In this one-dimensional value, i is the number of the spherical pyramid containing the point v, 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 containing v 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

$$spv_v = (i \cdot ceil(\sqrt{d}) + d_v)$$

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 $ceil(\sqrt{d})$ 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 $[i \cdot ceil(\sqrt{d}),\ (i \cdot ceil(\sqrt{d}) + 0.5\sqrt{d})]$. In order to make the sets of spherical pyramid values covered by any two spherical pyramids sp_i and sp_j be disjunct, we multiply i by $ceil(\sqrt{d})$. Without this multiplication of i by $ceil(\sqrt{d})$, the interval of every point within a spherical pyramid sp_i would be $[i,\ (i + 0.5\sqrt{d})]$. Thus, there might 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 four.

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 multiply the spherical pyramid number i by $ceil(\sqrt{d})$. Note further that this transformation is not injective. That is, two points v and v' may have the same spherical 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, the SPY-TEC does not require a bijective transformation [8].

3.2 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 corresponding data page of the B^+-tree. Update and delete operations can be done similarly.

The spherical pyramid values of points that all belong to the same spherical pyramid lies in the interval given by the minimum and maximum key values of the data pages. Thus, a single B^+-tree data page corresponds to a spherical slice of a spherical pyramid as shown in Figure 2(right). The page regions of the R-tree are (minimum) bounding rectangles, whereas the page regions of the SPY-TEC are spherical slices. Thus, in the rest of the paper, we call the spherical slice the *bounding slice (BS)*.

4 Incremental Nearest Neighbor Algorithm on the SPY-TEC

The algorithm proposed in [10] picks the node with the least distance in the set of all nodes that have yet to be visited when deciding what node to traverse

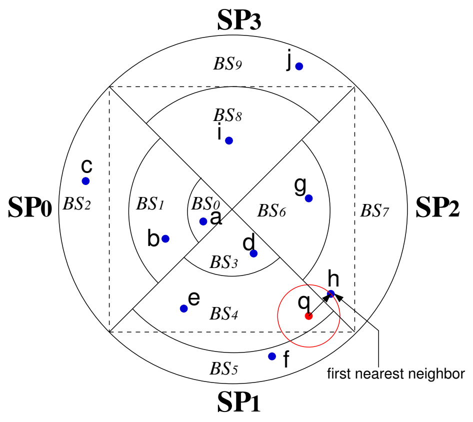


Fig. 4. An example of the SPY-TEC for a set of 10 points

next on the R-tree. This means that instead of using a stack or a plain queue to keep track of the nodes to be visited, it uses a priority queue where the distance from the query point is used as a key. In our algorithm, we also use a priority queue where the distance from the query point to the nodes or objects is used as a key.

4.1 Metrics for Nearest Neighbor Search

For the incremental nearest neighbor search on the SPY-TEC, we need the minimum possible distance from the query object to a node in the SPY-TEC. Figure 4 shows an example of the SPY-TEC in a two-dimensional data space. For the sake of simplicity, we assume that each bounding slice contains one object. In Figure 4, the query point falls within a bounding slice BS_4 in the spherical pyramid sp_1. As with most nearest neighbor algorithms, we must first visit the page (BS_4 in this example) containing the query point. Then, we visit the next page with the second smallest minimum distance from the query point. To do so, we must calculate the minimum possible distance from the query point to a spherical pyramid or a bounding slice. We first describe the process of calculating the minimum distance between the query point and a spherical pyramid, and then discuss the process of calculating the minimum distance between the query point and a bounding slice.

Lemma 1, which follows, measures the minimum distance $\text{MinDist}(q, sp_i)$ from the query point q to a spherical pyramid sp_i. For the sake of simplicity, we focus on the description of the case only for spherical pyramids sp_i where $i < d$. However, this lemma can be extended to all spherical pyramids in a straight-forward manner [8].

Lemma 1. (Minimum Distance from a Query Point to a Spherical Pyramid) Given a query point ($q = [q_0, q_1, ..., q_{d-1}]$), let sp_j ($j < d$) be the spherical pyramid containing a query point, and sp_i be the spherical pyramid

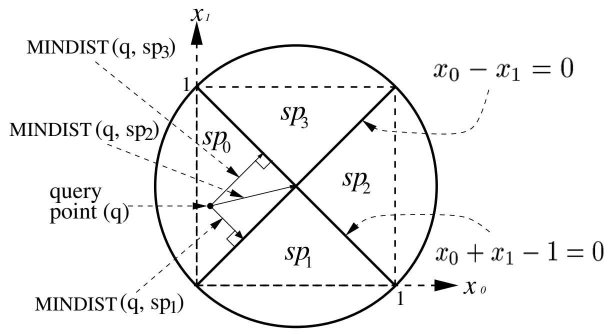


Fig. 5. The minimum distance from the query point to a spherical pyramid

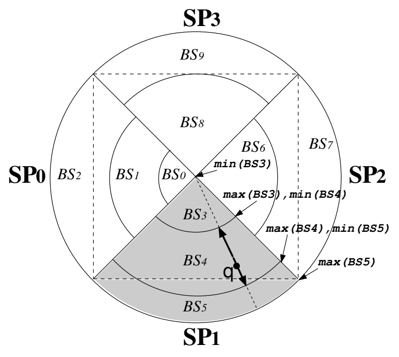
that will be examined for the minimum possible distance from q. The minimum distance from q to sp_i, $\textsc{MinDist}(q, sp_i)$, is defined as

$$\textsc{MinDist}(q, sp_i) = \begin{cases} 0 & \text{if } i = j \\ d_q & \text{if } |i-j| = d \\ \dfrac{|q_j - q_i|}{\sqrt{2}} & \text{if } i < d \\ \dfrac{|q_j + q_i - 1|}{\sqrt{2}} & \text{if } i > d \end{cases}$$

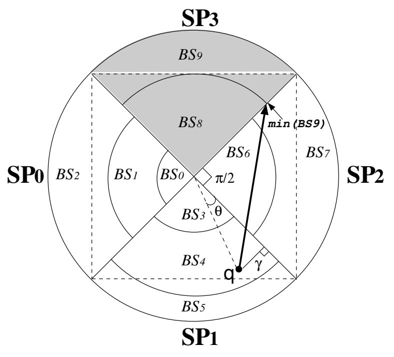
Proof : Given a point ($[q_0, q_1, ..., q_{d-1}]$) and a hyperplane ($k_0x_0 + k_1x_1 + ... + k_{d-1}x_{d-1} + C = 0$), the distance from the point to the hyperplane in Euclidean geometry is defined as

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

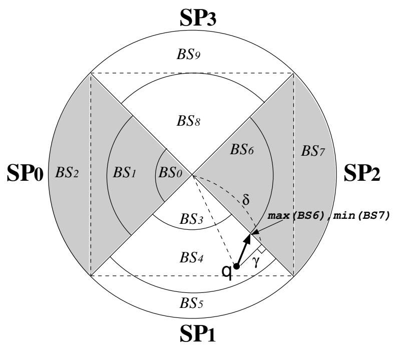
We are able to prove the case ($i > d$) and the case ($i < d$) using this formula.
1. If $i = j$, sp_i is the spherical pyramid containing the query point q. Therefore, $\textsc{MinDist}(q, sp_i) = 0$, which is less than or equal to the distance of q from any point in sp_i.
2. If $|i-j| = d$, sp_i is the spherical pyramid on the opposite side of sp_j. Therefore, the minimum distance of q from sp_i is the distance from q to the top of sp_i (the center of the data space). Thus, according to the notation of Definition 2, $\textsc{MinDist}(q, sp_i) = d_q$.
3. In formula (1), the index k_n and the constant C have discrete values [-1,0,1] because of unit space. If $i < d$, the 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 2-dimensional example of Figure 5. This formula can be extended to a 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 Figure 5, but a $(d-1)$-dimensional hyperplane, and the equation for this $(d-1)$-dimensional hyperplane has the



(a) Case 1



(b) Case 2



(c) Case 3

Fig. 6. The minimum distance from the query point to a bounding slice

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 minimum 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, $\textsc{MinDist}(q, sp_i) = |q_j - q_i|/\sqrt{2}$.
4. If $i > d$, the equation for the closest side plane of a spherical pyramid adjacent to the query point is $k_j x_j + k_i x_i - 1 = 0$ (refer to Figure 5). In this case, $k_j = 1$ and $k_i = 1$, because $i > d$. Thus, the minimum distance from the query point to the closest side plane of an adjacent spherical pyramid sp_i is $|q_j + q_i - 1|/\sqrt{2}$. Therefore, $\textsc{MinDist}(q, sp_i) = |q_j + q_i - 1|/\sqrt{2}$. □

Calculating the minimum distance from the query point to a bounding slice is more complex than the case of the minimum distance from the query point to a spherical pyramid. However, as depicted in Figure 6 and Lemma 2, we can present it easily by classifying into three cases.

Lemma 2. (Minimum Distance from a Query Point to a Bounding Slice) Given a query point (q), let sp_j be the spherical pyramid containing a query point, and BS_l be the bounding slice that belongs to a spherical pyramid sp_i. The minimum distance from q to a bounding slice BS_l, $\textsc{MinDist}(q, BS_l)$, is defined as
Case 1: ($i = j$: the case of BS_l belonging to the spherical pyramid that contains q.)

$$\textsc{MinDist}(q, BS_l) = \begin{cases} |d_q - max(BS_l)| & \text{if } d_q > max(BS_l) \\ 0 & \text{if } min(BS_l) \le d_q \le max(BS_l) \\ |d_q - min(BS_l)| & \text{if } d_q < min(BS_l) \end{cases}$$

Case 2: ($|i - j| = d$: the case of BS_l belonging to the spherical pyramid on the opposite side of q.)
Let α be the distance from the closest side plane of a spherical pyramid adjacent to q and θ ($\le \pi/4$) be the angle of a right-angled triangle which consists of two sides, α and d_q ($sin\theta = \frac{\alpha}{d_q}$),

$$\textsc{MinDist}(q, BS_l) = \sqrt{{d_q}^2 + min(BS_l)^2 - 2d_q min(BS_l) cos(\theta + \tfrac{\pi}{2})}$$

Case 3: ($otherwise$: the case of BS_l belonging to a spherical pyramid adjacent to q.)
Let δ be the length of the base line in a right-angled triangle which consists of two sides, α and d_q,

$$\textsc{MinDist}(q, BS_l) = \begin{cases} \sqrt{|\delta - max(BS_l)|^2 + \alpha^2} & \text{if } \delta > max(BS_l) \\ \alpha & \text{if } min(BS_l) \le \delta \le max(BS_l) \\ \sqrt{|\delta - min(BS_l)|^2 + \alpha^2} & \text{if } \delta < min(BS_l) \end{cases}$$

where;

$$min(BS_l) = \{d_v \mid (\forall v', \ v, v' \in BS_l : d_v \le d_{v'})\}$$
$$max(BS_l) = \{d_v \mid (\forall v', \ v, v' \in BS_l : d_v \ge d_{v'})\}$$

Proof: $min(BS_l)$ is d_v of the point v having the smallest value of the points belonging to BS_l, while $max(BS_l)$ is d_v of the point v having the largest value. We can prove each case by using $min(BS_l)$ and $max(BS_l)$.
1. If $min(BS_l) \le d_q \le max(BS_l)$, then q is inside BS_l. Therefore, $\textsc{MinDist}(q, BS_l) = 0$ because it is less than or equal to the distance of q from any point inside BS_l. If $d_q > max(BS_l)$, the distances of all of the points in BS_l from the center of the space are less than the distance of q from the center of the space. Therefore, $\textsc{MinDist}(q, BS_l)$ is the difference between d_q and d_v, where the point v is in BS_l and is farthest from the center of the space. That is, $\textsc{MinDist}(q, BS_l) = |d_q - max(BS_l)|$. Finally, if $d_q < min(BS_l)$, the distance of q from the center is less than the distances of all of the points in BS_l from the center. Therefore, $\textsc{MinDist}(q, BS_l)$ is the difference between d_q and d_v, where the point v is in BS_l and is closest to the center. That is, $\textsc{MinDist}(q, BS_l) = |d_q - min(BS_l)|$. In Figure 6(a), $\textsc{MinDist}(q, BS_4)$ is 0 because q is inside BS_4. Also, $\textsc{MinDist}(q, BS_3)$

is $|d_q - max(BS_3)|$ because the distances of all of the points in BS_3 are less than the distance of q. Finally, $\textsc{MinDist}(q, BS_5)$ is $|d_q - min(BS_5)|$ because the distance of q is less than the distances of all of the points in BS_5.
2. If $|i-j| = d$, sp_i is on the opposite side to the spherical pyramid containing q. In this case, the minimum distance from q to BS_l inside sp_i is the length of the base of a triangle which consists of two sides, such as d_q and $min(BS_l)$, and the angle between them as depicted in Figure 6(b). By using the *cosine rule* [15], we can get the length of the base of a triangle. First, the angle of the top of a spherical pyramid is $\pi/2$. Thus, the angle between d_q and $min(BS_l)$ is $(\theta + \pi/2)$ where $\theta = \arcsin(\alpha/d_q)$. Given the lengths of two sides (b and c) and the angle (A) between them, the cosine rule states : $a^2 = b^2 + c^2 - 2bc \cdot cosA$. Therefore, by the *cosine rule*, $\textsc{MinDist}(q, BS_l) = \sqrt{{d_q}^2 + min(BS_l)^2 - 2d_q min(BS_l) \cdot cos(\theta + \frac{\pi}{2})}$. Figure 6(b) shows this case in a two-dimensional example. $\textsc{MinDist}(q, BS_8)$ is d_q because $min(BS_8) = 0$.
3. In this case, sp_i is adjacent to sp_j which contains q. All sub-cases of this case are similar to those of **Case 1** except that the parameter for classifying each sub-case is not d_q, but δ. If $min(BS_l) \leq \delta \leq max(BS_l)$, $\textsc{MinDist}(q, BS_l)$ is the distance from q to the closest side plane of sp_i. That is, $\textsc{MinDist}(q, BS_l)$ $= \alpha$. This is similar to the sub-case $(min(BS_l) \leq d_q \leq max(BS_l))$ of **Case 1**. If $\delta > max(BS_l)$, $\textsc{MinDist}(q, BS_l)$ is the length of the hypotenuse in a right-angled triangle which consists of two sides, α and $|\delta - max(BS_l)|$. Therefore, $\textsc{MinDist}(q, BS_l) = \sqrt{|\delta - max(BS_l)|^2 + \alpha^2}$. Finally, if $\delta < min(BS_l)$, $\textsc{MinDist}(q, BS_l)$ is the length of the hypotenuse in a right-angled triangle which consists of two sides, α and $|\delta - min(BS_l)|$. Therefore, $\textsc{MinDist}(q, BS_l) = \sqrt{|\delta - min(BS_l)|^2 + \alpha^2}$. Figure 6(c) shows this case in a two-dimensional example. □

4.2 Algorithm Description

Algorithm 1 shows the algorithm for processing the nearest neighbor query. In lines 1∼4, the distances of each spherical pyramid from the query point are calculated by using Lemma 1, and then information about each spherical pyramid and its distance are inserted into the priority queue. Since the distance is used as a key in the priority queue, the spherical pyramid closest to the query point is at the head of the queue. The **while**-loop of lines 6 ∼ 21 is the main loop for the algorithm. In line 7, the first element in the head of the queue is dequeued and, according to the type of the element, appropriate operations will be performed. If the type of the element dequeued is a spherical pyramid, as depicted in lines 8 ∼ 12, the distances of each bounding slice in the spherical pyramid from the query point are calculated, and then information of each bounding slice and its distance are inserted into the queue by using Lemma 2. If the type is a bounding slice, as depicted in lines 13 ∼ 17, the distances of each object in the bounding slice from the query point are calculated, and then inserted into the queue. Finally, if the type is an object, it is reported as the next nearest

Algorithm 1 Processing the incremental nearest neighbor query

```
1: for $i = 0$ to $2d - 1$ do
2:   dist = MinDist($q, sp_i$); {Using Lemma 1}
3:   Enqueue(queue, $sp_i$, dist);
4: end for
5:
6: while not IsEmpty(queue) do
7:   Element = Dequeue(queue);
8:   if Element is a spherical pyramid then
9:     for each bounding slice in a spherical pyramid do
10:      dist = MinDist($q, BS_l$); {Using Lemma 2}
11:      Enqueue(queue, $BS_l$, dist);
12:    end for
13:  else if Element is a bounding slice then
14:    for each object in a bounding slice do
15:      dist = Dist_Query_to_Obj($q, object$);
16:      Enqueue(queue, $object$, dist);
17:    end for
18:  else {Element is a object}
19:    report element as the next nearest object
20:  end if
21: end while
```

neighbor object. The first reported object is naturally the nearest neighbor to the query point. If we control the number of reported nearest neighbors in the **while**-loop of Algorithm 1, we can easily process the k-nearest neighbor query.

4.3 Example

As an example, suppose that we want to find the first nearest neighbor to the query point q in the SPY-TEC given in Figure 4. Below, we show the steps of the algorithm and the contents of the priority queue. Table 1 shows these distances (*SP* means spherical pyramid and *BS* means bounding slice). When depicting the contents of the priority queue, the spherical pyramids and bounding slices are listed with their distances from the query point q, in order of increasing distance. The objects are denoted in bold letters (e.g., **a**). The algorithm starts by enqueueing $SP_0 \sim SP_3$, after which it executes the following steps:

1. Enqueue $SP_0 \sim SP_3$. Queue : {[SP_1,0], [SP_2,4], [SP_0,21], [SP_3,33]}
2. Dequeue SP_1, enqueue BS_3, BS_4, BS_5. Queue : {[BS_4,0], [BS_5,2], [SP_2,4], [BS_3,14], [SP_0,21], [SP_3,33]}
3. Dequeue BS_4, enqueue **e**. Queue : {[BS_5,2], [SP_2,4], [BS_3,14], [**e**,19], [SP_0,21], [SP_3,33]}
4. Dequeue BS_5, enqueue **f**. Queue : {[SP_2,4], [**f**,12], [BS_3,14], [**e**,19], [SP_0,21], [SP_3,33]}
5. Dequeue SP_2, enqueue BS_6, BS_7. Queue : {[BS_7,4], [BS_6,8], [**f**,12], [BS_3,14], [**e**,19], [SP_0,21], [SP_3,33]}

SP	Dist.
SP_0	21
SP_1	0
SP_2	4
SP_3	33

BS	Dist.
BS_0	21
BS_1	25
BS_2	29
BS_3	14
BS_4	0
BS_5	2
BS_6	8
BS_7	4
BS_8	33
BS_9	42

OBJ	Dist.
a	23
b	27
c	45
d	16
e	19
f	12
g	35
h	6
i	39
j	47

Table 1. Distances of spherical pyramids and bounding slices from the query point q in the SPY-TEC of Figure 4.

6. Dequeue BS_7, enqueue **h**. Queue : {[**h**,6], [BS_6,8], [**f**,12], [BS_3,14], [**e**,19], [SP_0,21], [SP_3,33]}
7. Dequeue **h**, report **h** as the first nearest neighbor.

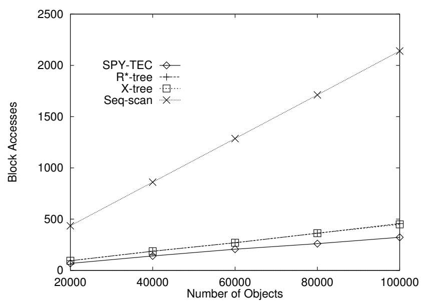
Since the elements in the priority queue are sorted in increasing order of distance, sp_1 containing the query point q is at the head of the queue. In line 7 of Algorithm 1, sp_1 is dequeued, and then BS_3, BS_4, and BS_5 in sp_1 are enqueued in increasing order of their distances from the query point. Now, BS_4 is at the head of the queue because it has the smallest distance. BS_4 is dequeued, and then the objects in BS_4 are enqueued. In this example, since we assume that only one object is contained in a bounding slice, the object **e** in BS_4 is enqueued. These operations are repeated until the user finds as many nearest neighbors as desired.

5 Experimental Evaluation

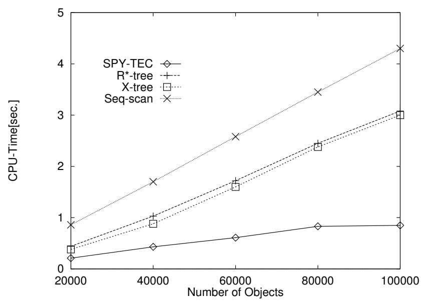
We performed various experiments to show the practical impact of the incremental nearest neighbor algorithm on the SPY-TEC and compared it to the R*-tree and the X-tree, as well as the sequential scan.

For clear comparison, we implemented the incremental nearest neighbor algorithm on the R*-tree and the X-tree using the algorithm proposed in [10]. All experiments were performed on a SUN SPARC 20 workstation with 128 MByte main memory and 10 GByte secondary storage. The block size used for our experiments was 4 KBytes. Due to lack of space, we show only the experiment using real data sets, although we performed various experiments using synthetic data sets and real data sets. For a more detailed results of various experiments, you can refer to [16].

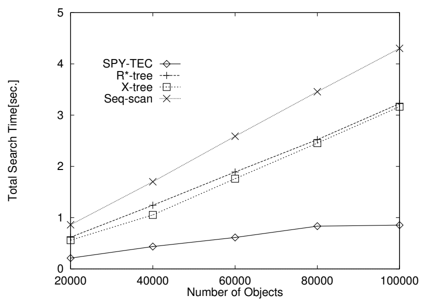
The real data consists of Fourier points [22] in 12-dimensional space. We performed 10-nearest neighbor queries with 100 query points that were selected from the real data itself, and varied the database size from 20,000 to 100,000.



(a) Block access



(b) CPU time



(c) Total search time

Fig. 7. Performance Behavior on Real Data

Figure 7 shows the result of the experiment using real data sets. In this experiment, the SPY-TEC, along with the R*-tree or the X-tree significantly outperform the sequential scan regardless of the database size. From this result, we found that the real data consists of well-formed clusters which are meaningful workloads for high-dimensional nearest neighbor queries. The speed-up of the SPY-TEC in the total search time ranges between 2.42 and 3.71 over the X-tree, between 2.85 and 3.78 over the R*-tree, and between 3.90 and 5.04 over the sequential scan. The performance behavior of the number of block accesses and of CPU time are analogous to that of the total search time. The index structures SPY-TEC, X-tree, and R*-tree significantly outperform the sequential scan in all cases, and the SPY-TEC also clearly yields a better performance than do the X-tree and the R*-tree.

6 Conclusions

The SPY-TEC is based on a special partitioning strategy which divides the d-dimensional data space first into $2d$ spherical pyramids, and then cuts each spherical pyramid into several bounding slices. In this paper, we proposed the incremental nearest neighbor algorithm on the SPY-TEC. We also introduced a metric that can be used to guide an ordered best-first traversal when finding nearest neighbors on the SPY-TEC. The metric (MinDist), the minimum possible distance of the query point from a spherical pyramid or a bounding slice, produces the most optimistic ordering possible when finding nearest neighbors on the SPY-TEC. We implemented the incremental nearest neighbor algorithm on the SPY-TEC and performed extensive experiments using synthetic data and real data sets to show the practical impacts of these algorithms. Through the experiments, we showed that the incremental algorithm on the SPY-TEC clearly outperforms that of the X-tree, the R*-tree, and the sequential scan.

For highly skewed data distributions or queries, the incremental nearest neighbor algorithm on the SPY-TEC may perform worse than those on other index structures. However, none of the index structure proposed so far can handle highly skewed data or queries efficiently [20]. We plan to address the problem of handling highly skewed data or queries in our future work. We also plan to study the parallel version of the nearest neighbor algorithm on the SPY-TEC using an efficient declustering technique that distributes the data onto the disks so that the data which has to be read when executing a query are distributed as equally as possible among the disks.

References

1. A. Guttman. “R-trees: a dynamic index structure for spatial searching”. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, June 1984.
2. A. Henrich. “The LSDh-Tree: An Access Structure for Feature Vectors”. *Proc. 14th Int. Conf on Data Engineering*, pages 362–369, 1998.
3. C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equiz. “Efficient and Effective Querying by Image Content”. *Journal of Intelligent Information System(JIIS)*, 3(3):231–262, July 1994.
4. B. C. Ooi, K. L. Tan, T. S. Chua, and W. Hsu. “Fast image retrieval using color-spatial information”. *The VLDB Journal*, 7(2):115–128, 1998.
5. C. E. Jacobs, A. Finkelstein, and D. H. Salesin. “Fast Multiresolution Image Query”. *Proc. of the 1995 ACM SIGGRAPH, New York*, 1995.
6. D. A. White and R. Jain. “Similarity Indexing with the SS-tree”. *Proc. 12th Int. Conf on Data Engineering*, pages 516–523, 1996.
7. D. B. Lomet and B. Salzberg. “The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance”. *ACM Transaction on Database Systems*, 15(4):625–658, 1990.
8. D. H. Lee and H. J. Kim. “SPY-TEC : An Efficient Indexing Method for Similarity Search in High-Dimensional Data Spaces”. *Data & Knowledge Engineering*, 34(1):77–97, 2000.
9. C. Faloutsos. “Fast Searching by Content in Multimedia Databases”. *Data Engineering Bulletin*, 18(4), 1995.
10. G. R. Hjaltason and H. Samet. “Distance Browsing in Spatial Databases”. *ACM Transaction on Database Systems*, 24(2):265–318, 1999.
11. J. Bentley. “Mutidimensional binary search trees used for associative searching”. *Communications of the ACM*, 18(9):509–517, 1975.

12. J. R. Smith and S.-F. Chang. "VisualSEEk: a fully automated content-based image query system". *ACM Multimedia 96, Boston, MA*, 1996.
13. J. T. Robinson. "The K-D-B-tree: a Search Structure for Large Multidimensional Dynamic Indexes". *Proc. ACM SIGMOD, Ann Arbor, USA*, pages 10–18, April 1981.
14. K.-I. Lin, H. V. Jagadish, and C. Faloutsos. "The TV-tree: An Index Structure for High-Dimensional Data". *The VLDB Journal*, 3(4):517–542, 1994.
15. L. Leithold. "Trigonometry". *Addison-Wesley*, 1989.
16. D. H. Lee and H. J. Kim. "An Efficient Nearest Neighbor Search in High-Dimensional Data Spaces". *Seoul National University, CE Technical Report (OOPSLA-TR1028), http://oopsla.snu.ac.kr/~dhlee/OOPSLA-TR1028.ps*, 2000.
17. N. Katayama and S. Satoh. "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries". *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 517–542, May 1997.
18. N. Roussopoulos, S. Kelley, and F. Vincent. "Nearest Neighbor Queries". *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, 1995.
19. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. "When Is "Nearest Neighbor" Meaningful ?". *Proc. 7th Int. Conf. on Database Teory*, pages 217–235, January 1999.
20. S. Berchtold, C. Böhm, and H.-P. Kriegel. "The Pyramid-Technique: Towards Breaking the Curse of Dimensionality". *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1998.
21. S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. "A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space". *ACM PODS Symposium on Principles of Database Systems, Tucson, Arizona*, 1997.
22. S. Berchtold, D. A. Keim, and H.-P. Kriegel. "The X-tree: An Indexing Structure for High-Dimensional Data". *Proc. 22nd Int. Conf. on Very Large Database*, pages 28–39, September 1996.
23. P.M. Kelly, T.M. Cannon and D.R. Hush. "Query by image example: the CANDID approach". *Proc. SPIE Storage and Retrieval for Image and Video Databases III*, 2420:238–248, 1995.