Significance-Sensitive Nearest-Neighbor Search for Efficient Similarity Retrieval of Multimedia Information

Norio Katayama and Shin'ichi Satoh

Research and Development Department
NACSIS (National Center for Science Information Systems)
3-29-1 Otsuka, Bunkyo-ku, Tokyo 112-8640, Japan
E-mail: {katayama,satoh}@rd.nacsis.ac.jp
Phone: +81-3-3942-6970 Fax: +81-3-5395-7064

Abstract

Nearest-neighbor search (NN-search) in the feature space is widely used for the similarity retrieval of multimedia information. Each piece of multimedia information is mapped to a vector in a multi-dimensional space where the distance between two vectors (typically, Euclidean distance between the heads of vectors) corresponds to the similarity of multimedia information. Once the feature space is obtained, the similarity retrieval of multimedia information is reduced to NN-search in the feature space. One of the important properties of the feature space is high dimensionality. It is very common to use 10 or higher dimensional space for the feature space. Recent research results in the database literatures reveal that a curious problem happens in high-dimensional space, which is not imagined in low-dimensional space. Since high-dimensional space has high degree of freedom, points could be so scattered that every distance between them might yield no significant difference. This problem affects the performance of NN-search. This paper shows how the problem affects the similarity retrieval of multimedia information, and how it can be circumvented with the introduction of a new NN-search algorithm: the significance-sensitive nearest-neighbor search.

1 Introduction

Nearest-neighbor search (NN-search) in the feature space is widely used for the similarity retrieval of multimedia information. Each piece of multimedia information is mapped to a vector in a multi-dimensional space where the distance between two vectors (typically, Euclidean distance between the heads of vectors) corresponds to the similarity of multimedia information. These vectors and the space are called feature vectors and feature space respectively. Once the feature space is obtained, the similarity retrieval of multimedia information is reduced to NN-search in the feature space. One of the typical examples of the feature space is the color histogram of an image which indicates how much portion of an image has a specific color (e.g., red, blue, green, etc.). One of the important properties of the feature space is high dimensionality. It is very common to use 10 or higher dimensional space for the feature space. For example, colors are often classified into 16 or more colors for color histograms. This generates 16 or more dimensional feature vectors.

In the database literatures, quite a few index structures have been proposed for the acceleration of NN-search: k-d-tree-like tree structures[1, 2, 3, 4, 5], a new space-partitioning method[6], etc. These index structures partition the feature space into the hierarchy of regions. The hierarchical structure permits NN-search operations to be completed with examining only some part of the feature space. This reduces CPU time and disk I/O of NN-search operations. Therefore, these index structures are widely used for the acceleration of the similarity retrieval of multimedia information.

Recent results in the database literatures reveal that a curious problem happens in high-dimensional space, which is not imagined in low-dimensional space. Since high-dimensional space has high degree of freedom, points could be so scattered that every distance between them might yield no significant difference. A typical example appears in the uniformly distributed points, i.e., points distributed uniformly in a unit hypercube. For example, Berchtold et al.[7] reported that most of the points are located near the surface of the cube, and Katayama et al.[4] reported that the distances among the points are very similar for any combination of them. Recently, Beyer et al.[8] showed that under a broad set of conditions (broader than uniform distribution) the distance to the nearest data point approaches the distance to the farthest data point as dimensionality increases. This problem, i.e., no significant difference in distances, affects the performance of NN-search. This paper shows how the problem affects

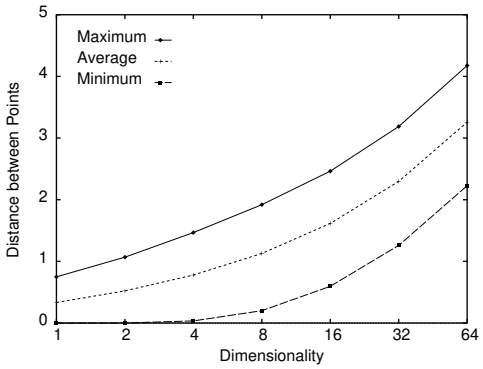


Figure 1: The maximum, the average, and the minimum of the distances between points uniformly distributed in a unit hypercube.

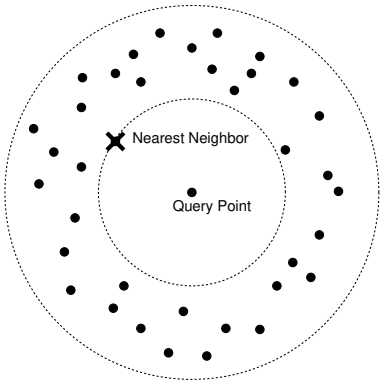


Figure 2: Insignificant nearest neighbor likely to occur in high-dimensional space.

the similarity retrieval of multimedia information, and how it can be circumvented with the introduction of a new NN-search algorithm: the significance-sensitive nearest-neighbor search.

This paper is organized as follows. Section 2 shows how insignificant nearest neighbors affect the similarity retrieval of multimedia information. Section 3 presents the new algorithm. The result of experiments are shown in Section 4. Section 5 contains conclusions.

2 Insignificant Nearest Neighbors in Feature Spaces

2.1 Insignificance of Nearest Neighbors

When we use NN-search, we expect that found neighbors are much closer than the others. However, this intuition is sometimes incorrect in high-dimensional space. For example, when points are uniformly distributed in a unit hypercube, the distance between two points is almost the same for any combination of two points. Figure 1 shows the minimum, the average, and the maximum of the distances among 100,000 points distributed uniformly in a unit hypercube. As shown in the figure, the minimum of the distances grows drastically as the dimensionality increases and the ratio of the minimum to the maximum increases up to 24% in 16 dimensions, 40% in 32 dimensions, and 53% in 64 dimensions. Thus, the distance to the nearest neighbor is only 53% of the distance to the farthest point in 64 dimensional space (Figure 2). In this case, we can consider the nearest neighbor to be insignificant, because the difference between the nearest neighbor and the others is negligible, i.e., the other points are as close to the query point as the nearest neighbor is. From the perspective of the similarity retrieval, when the nearest neighbor is insignificant, the nearest neighbor has almost the same similarity with the others and does not have significant similarity to the query.

As Figure 1 shows, insignificant nearest neighbors are more likely to occur as dimensionality increases. This characteristics can be verified with estimating the distance to k-th nearest neighbor. When N points are distributed uniformly within the hypersphere whose center is the query point, the expected distance to k-th nearest neighbor d_{kNN} is obtained as follows[9]:

$$E\{d_{kNN}\} \approx \frac{\Gamma(k+1/n)}{\Gamma(k)} \frac{\Gamma(N+1)}{\Gamma(N+1+1/n)} r, \qquad (1)$$

where n is the dimensionality of the space and r is the radius of the hypersphere. Then, the ratio of the $(k+1)$-NN distance to the k-NN distance is[9]:

$$\frac{E\{d_{(k+1)NN}\}}{E\{d_{kNN}\}} \approx 1 + \frac{1}{kn}. \qquad (2)$$

Thus, when points are distributed uniformly around the query point, it is expected that the difference between the k-th and $(k+1)$-th nearest neighbors decreases as dimensionality increases. This implies that insignificant nearest neighbors are more likely to occur in high-dimensional space than in low-dimensional space.

2.2 Harmful Influence of Insignificant NNs on Similarity Retrieval

Insignificant nearest neighbors have harmful influence on the similarity retrieval with the following respects:

- NN-search performance is degraded.

  When the nearest neighbor is insignificant, there exist many points that have almost the same similarity with the nearest neighbor. Since these

points are very strong candidates for the nearest neighbor, NN-search operations are forced to examine many points before determining the true nearest neighbor. This degrades the performance of NN-search operations.

- Meaningless results are answered.

  When the nearest neighbor is insignificant, NN-search operations answer the closest point among many strong candidates that have almost the same similarity with the nearest neighbor. This means that all of the candidates are either similar or dissimilar. Therefore, it is meaningless to choose the nearest neighbor from plenty of similar candidates.

These effects are extremely harmful to the retrieval systems with human-interaction. When the nearest neighbor is insignificant, the system forces users to wait until meaningless result is answered with very slow response. Thus, it is necessary to handle insignificant nearest neighbors appropriately in order to achieve efficient similarity retrieval of multimedia information.

3 NN-Search Algorithm

3.1 Definition of Insignificant Nearest Neighbors

In order to circumvent the harmful influence of insignificant nearest neighbors, we have devised a new nearest-neighbor search algorithm which quits the search and answers the partial result when it finds that a nearest neighbor is insignificant. We call this algorithm as the significance-sensitive nearest-neighbor search. This detects insignificant nearest neighbors according to the following definition:

Definition 1 *Let d_{NN} be the distance from the query point to a nearest neighbor. Then, the nearest neighbor is insignificant if more than or equal to N_c points exist within the range of d_{NN} to $R_p \times d_{NN}$ from the query point.*

Here, $R_p(> 1)$ and $N_c(> 1)$ are controllable parameters (Figure 3). R_p determines the proximity of the query point and N_c determines the congestion of the proximity. For example, we set 1.84 to R_p and 48 to N_c at the experiment described in Section 4. The way to find appropriate values for R_p and N_c will be discussed in Section 3.4 after the search algorithm has been described.

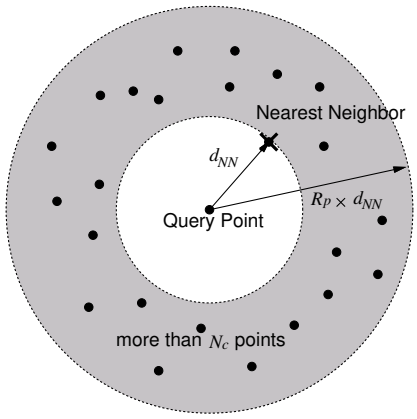


Figure 3: Definition of insignificant nearest neighbor.

3.2 NN-Search Algorithms for Multidimensional Index Structures

The significance-sensitive NN-search algorithm is designed for multidimensional index structures that splits the data space into the nested hierarchy of regions, e.g., the SS-tree[1], the VAMSplit R-tree[2], the X-tree[3], the SR-tree[4], and the LSD^h-tree[5], etc. Figure 4 illustrates the hierarchical structure of the R-tree. Each node of the tree structure corresponds to a region of the data space. Non-internal nodes store data points, while internal nodes store the information on child nodes, i.e., region specifications (e.g., location, size, shape, etc.) and pointers to child nodes. The hierarchical structure permits NN-search operations to be completed with examining only some part of the feature space. This reduces CPU time and disk I/O of NN-search operations. Therefore, these index structures are widely used for the acceleration of the similarity retrieval of multimedia information.

We designed the new algorithm with extending the NN-search algorithm presented by Hjaltason et al.[10] (This algorithm will be called the *basic* NN-search algorithm hereafter in order to be distinguished from the significance-sensitive NN-search algorithm). The basic NN-search algorithm finds the nearest neighbor of a given query point as follows. The search starts from the root node of the index structure. At every internal node, the distances from the query point to the child nodes are computed and the child nodes are enqueued into a priority queue in ascending order of the distance. Then, one node is dequeued from the priority queue and the search proceeds to the dequeued node. Thus, nodes are visited in ascending order of the distance from the query point. For example, in Figure 5 (a), regions will be searched in the order of Region 1, Region 2, and Region 3. On the other hand, every time a non-internal node is visited, the distances from the query

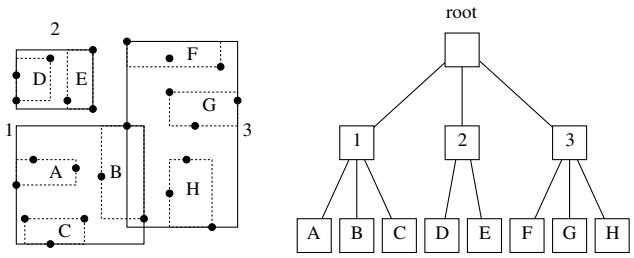
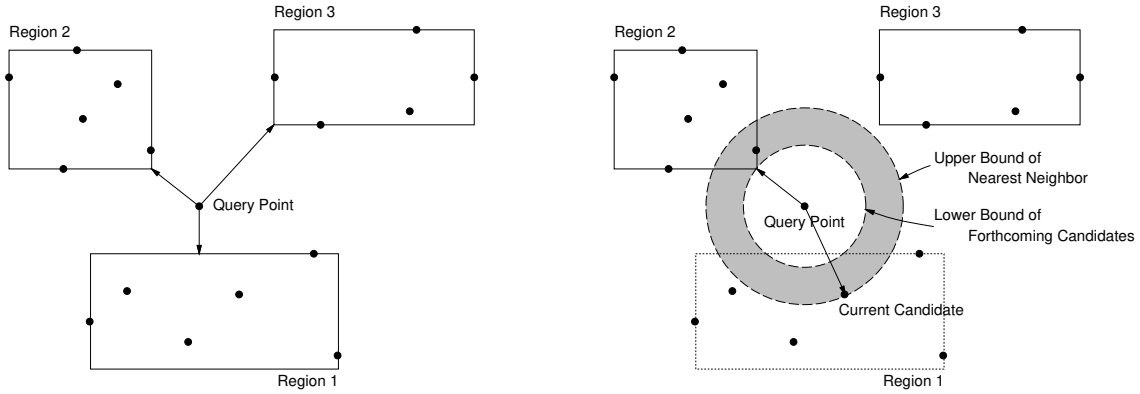


Figure 4: Hierarchical structure of the R-tree.



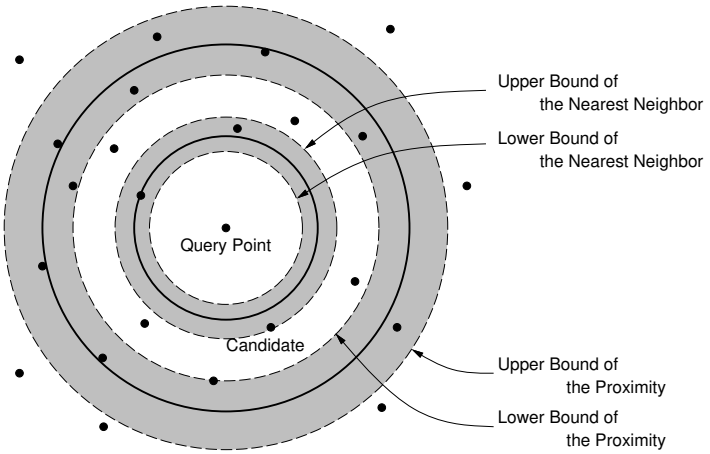
(a) Nodes are visited in ascending order of the distance from the query point.

(b) The current candidate gives the upper bound of the distance to the nearest neighbor, while the closest node among those not visited so far gives the lower bound of the distance to forthcoming candidates.

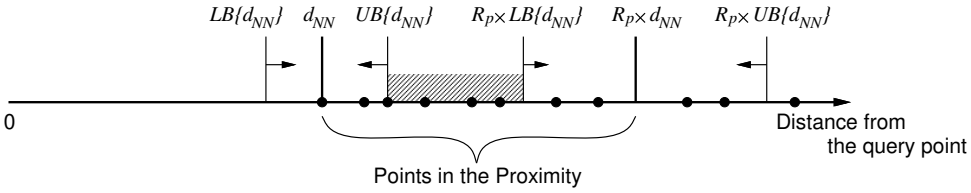
Figure 5: Nearest-neighbor search for multidimensional index structures.

point to the data points in the node are computed. If it is the first visit of a non-internal node, the closest point in the node is chosen for the candidate of the nearest neighbor; otherwise, the closest point in the node is compared with the latest candidate and the closer one is chosen for the new candidate. The distance to the candidate gives the upper bound of the distance to the nearest neighbor, because the nearest neighbor cannot be farther from the query point than the candidate is. On the other hand, the distance to the first node in the priority queue gives the lower bound of the distance to forthcoming candidates, because the first node of the priority queue is the closest among such nodes that are not visited so far. For example, Figure 5 (b) illustrates the case where Region 1 is visited and the closest point in Region 1 is chosen for the candidate. Region 2 and Region 3 are still in the priority queue. The candidate gives the upper bound of the distance to the nearest neighbor, while the distance to the first node of the priority queue, i.e., the distance to Region 2, gives the lower bound of the distance to forthcoming candidates. The search continues until the lower bound of the distance to forthcoming candidates exceeds the upper bound of the distance to the nearest neighbor, i.e., until no node in the priority queue is closer to the query point than the candidate. Finally, the nearest neighbor is obtained as the last candidate. Although the description above is the search finding the first nearest neighbor, it can be easily extended to the k-nearest neighbor search, i.e., finding k nearest neighbors.

While the basic NN-search algorithm finds the exact nearest neighbor, it can be modified to find an approximate answer with a given precision. This type of NN-search is called the approximate nearest neighbor search[11]. The approximate NN-search is achieved by the change of the termination condition. While the basic NN-search ends when the lower bound of the distance to forthcoming candidates exceeds the upper bound of the distance to the nearest neighbor, the ap-



(a) The lower and the upper bound of the proximity.



(b) The number of points in the proximity is more than or equal to the number of points in the range of $UB\{d_{NN}\}$ to $R_p \times LB\{d_{NN}\}$.

Figure 6: The bounds of the proximity can be obtained from the bounds of the nearest neighbor.

proximate NN-search ends when the ratio of the upper bound to the lower bound is reduced to less than or equal to $(1+\epsilon)$. Here ϵ is the controllable parameter of an error bound. At this point, the candidate is an approximate answer whose error in the distance from the query point is less than or equal to ϵ. Thus ϵ determines the precision of an approximate answer. The approximate nearest neighbor search is effective in improving the performance when it is not necessary to obtain exact answers.

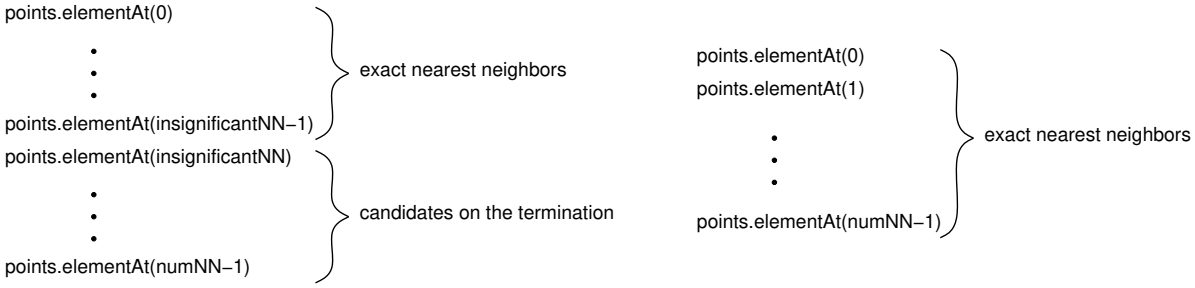
3.3 Significance-Sensitive NN-Search Algorithm

The significance-sensitive NN-search algorithm is an extension of the basic NN-search algorithm described above. The main idea of the extension is counting the number of points that are located within the proximity of the query point in the course of a search operation. The proximity is specified by the parameter R_p as in Definition 1. Then, the operation quits the search and answers the partial result if the congestion of the proximity is detected. The congestion is determined by the parameter N_c as in Definition 1, i.e., the case where the number of points in the proximity exceeds N_c is regarded as the congestion. In order to count points in the proximity, the algorithm takes advantage of the lower bound and the upper bound of the distance to the nearest neighbor d_{NN}. As stated in the description of the basic NN-search, the distance to the first node in the priority queue and the distance to the candidate of the nearest neighbor gives the lower and the upper bound of d_{NN} (Figure 5 (b)). These bounds also give the bounds of the proximity as shown in Figure 6. According to the definition of the proximity in Definition 1, the range of the proximity is $R_p \times d_{NN}$. When the lower and the upper bound of d_{NN} is $LB\{d_{NN}\}$ and $UB\{d_{NN}\}$ respectively, the lower bound and the upper bound of the range of the proximity can be obtained as $R_p \times LB\{d_{NN}\}$ and $R_p \times UB\{d_{NN}\}$. Since d_{NN} is obtained only at the end of the search, the algorithm uses $UB\{d_{NN}\}$ and $R_p \times LB\{d_{NN}\}$ to count points in the proximity. Because $d_{NN} \leq UB\{d_{NN}\}$ and $R_p \times LB\{d_{NN}\} \leq R_p \times d_{NN}$, the number of points in the proximity is more than or equal to the number of points that are located in the range of $UB\{d_{NN}\}$ to

```
void
searchNN(Point queryPoint, int numNN, double proximityThreshold, int congestionThreshold,
         PointVector *points_return, int *insignificantNN_return)
{
    PointQueue pointQueue = new PointQueue();
    NodeQueue nodeQueue = new NodeQueue();
    int i, numSignificantNN = 0, insignificantNN = -1;

    // visits the root node.
    visitNode(pointQueue, nodeQueue, queryPoint, numNN, getRootNodeOffset());
    // visits nodes until the first node in the queue is farther than the farthest active candidate.
    while ( nodeQueue.size() != 0 && insignificantNN < 0 &&
            (pointQueue.size() < numNN ||
             nodeQueue.elementAt(0).getDistance()
             <= pointQueue.elementAt(numNN-1).getDistance()) ) {
        // visits the first node in the node queue.
        off_t offset = nodeQueue.elementAt(0).getOffset();
        nodeQueue.removeElementAt(0);
        visitNode(pointQueue, nodeQueue, queryPoint, numNN, offset);
        // tests the significance of candidates.
        for ( i=numSignificantNN; i<pointQueue.size() && i<numNN; i++ ) {
            double pointDistance = pointQueue.elementAt(i).getDistance();
            if ( pointDistance < nodeQueue.elementAt(0).getDistance() ) {
                // this point is not a candidate but an exact answer.
                double proximityDistance = pointDistance * proximityThreshold;
                // tests whether this neighbor is insignificant.
                if ( i + congestionThreshold <= pointQueue.size() &&
                     pointQueue.elementAt(i + congestionThreshold - 1).getDistance()
                     <= proximityDistance ) {
                    insignificantNN = i;
                    break;
                }
                // tests whether this neighbor is significant.
                if ( proximityDistance < nodeQueue.elementAt(0).getDistance() ) {
                    numSignificantNN = i + 1;
                }
            } else {
                // this point is a candidate; then tests whether it is insignificant.
                if ( i + congestionThreshold <= pointQueue.size() &&
                     pointQueue.elementAt(i + congestionThreshold - 1).getDistance()
                     <= nodeQueue.elementAt(0).getDistance() * proximityThreshold ) {
                    insignificantNN = i;
                }
                break;
            }
        }
    }
    *points_return = new PointVector();
    for ( i=0; i<numNN && i<pointQueue.size(); i++ ) {
        (*points_return).addElement(pointQueue.elementAt(i).getPoint());
    }
    *insignificantNN_return = insignificantNN;
}
```

Figure 7: Significance-sensitive k-nearest-neighbor search algorithm.



(a) When an insignificant NN is detected.

(b) When no insignificant NN is detected.

Figure 8: Search result of the significance-sensitive k-nearest-neighbor search.

$R_p \times LB\{d_{NN}\}$ as shown in Figure 6 (b). Therefore, the congestion can be detected during the search operation by the test whether the number of points located in the range of $UB\{d_{NN}\}$ to $R_p \times LB\{d_{NN}\}$ is greater than or equal to N_c.

The program code in Figure 7 is the major part of the significance-sensitive k-NN-search algorithm written in C++. A query point is given to the first argument `queryPoint` and the number of nearest neighbors to be found is given to the second argument `numNN`. The third and the fourth argument, i.e., `proximityThreshold` and `congestionThreshold`, corresponds to the proximity parameter R_p and the congestion parameter N_c respectively. The search result is returned in the last two arguments: `*points_return` is the array of found nearest neighbors, `*insignificantNN_return` is the index to the insignificant nearest neighbor detected during the search. If no insignificant nearest neighbor is detected, `*insignificantNN_return` is set to -1. This routine tests the significance of nearest neighbors during the search. When the routine detects that one of `numNN` nearest neighbors is insignificant, it terminates the search and returns the partial result as shown in Figure 8 (a). The points, `points.elementAt(0)`, ... , `points.elementAt(insignificantNN-1)`, are exact answers, not candidates, because the test of the insignificance is only applied to exact answers and the closest candidate. When no insignificant nearest neighbor is detected, every point is the exact answer as shown in Figure 8 (b).

The class `PointQueue` and `NodeQueue` used in the routine are the priority queue of points and nodes respectively. An element of `PointQueue` stores a point and the distance to the point, while an element of `NodeQueue` stores a file offset of a node and the distance to the node. Both class has the following methods: `size()` returns the size of the queue, `elementOf(int n)` returns the n-th element in the queue, and `removeElementAt(int n)` removes the n-th element from the queue. The class `PointVector` is a variable-length array of the class `Point`. The function `visitNode()` used in the routine reads a node from the secondary storage and deploys the contents of the node into the queues, i.e., if the node is an internal node, it pushes the child nodes to the node queue; if the node is non-internal, it pushes the points in the node to the point queue.

In the perspective of the code optimization, the routine can be improved when redundant points and nodes are discarded from the queues. Points outside of the upper bound of the proximity of the `numNN`-th nearest neighbor, i.e., $R_p \times UB\{d_{\texttt{numNN}NN}\}$, are redundant points since they have no effect on the test of the significance. On the other hand, nodes outside of the upper bound of the `numNN`-th nearest neighbor, i.e., $UB\{d_{\texttt{numNN}NN}\}$, are redundant nodes since they need not be visited. Discarding redundant points and nodes shortens the size of queues. This reduces the computational cost of the search operation.

The routine in Figure 7 does not ensure that the returned exact nearest neighbors are significant. It terminates the search as soon as one insignificant nearest neighbor is found. Therefore, returned exact nearest neighbors may be significant or insignificant. In order to determine the significance of every exact nearest neighbors, the routine must be modified to continue the search until the significance of all nearest neighbors is determined. This modification is easy. However, the modified version requires more node visits and degrades the search performance. Therefore we employ the algorithm that terminates the search as soon as an insignificant nearest neighbor is found.

3.4 Optimal Parameter Settings

As shown above, the parameters R_p and N_c takes an essential role in the proposed algorithm. We present here the way to find optimal parameters with respect to the effective dimensionality[9]*. The distribution of the data may have less dimensionality than the data space. For example, when the distribution of the data is governed by a number of dominant dimensions, the effective dimensionality is given by the number of such dominant dimensions. In addition, when the data distribution is skewed, the effective dimensionality might not be consistent over the data set but vary from one local region to another. Therefore, the effective dimensionality is sometimes called the local dimensionality[9]. The parameter setting presented below consists of two steps. In the first step, the local dimensionality is related to the insignificance of nearest neighbors, then two control-points, (ν_c, ρ_c) and (ν_r, ρ_r), are determined in order to control what local dimensionality to be regarded as insignificant: ν_c is the cut-off dimensionality, ν_r the rejection dimensionality, ρ_c the rejection probability at ν_c, and ρ_r the rejection probability at ν_r. Then, in the second step, the parameters R_p and N_c are determined from these four parameters.

Firstly, we relate the insignificance of nearest neighbors with the local dimensionality by Equation (2). This equation holds when the local dimensionality is n [9]. Equation (2) indicates that the ratio of $d_{(k+1)NN}$ to d_{kNN} decreases monotonically as k increases. Therefore, the maximum of the ratio between two nearest neighbors is expected as follows:

$$\max_k \frac{E\{d_{(k+1)NN}\}}{E\{d_{kNN}\}} = \frac{E\{d_{2NN}\}}{E\{d_{1NN}\}} \approx 1 + \frac{1}{n}. \quad (3)$$

*The effective dimensionality is also called the intrinsic dimensionality[9].

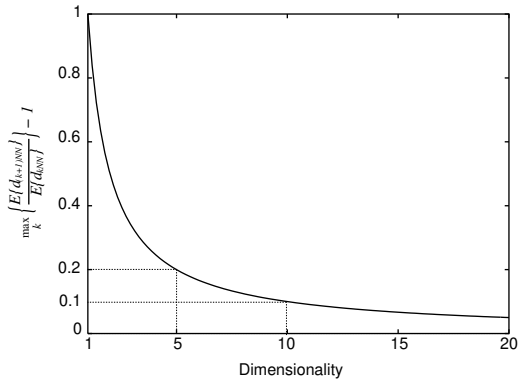


Figure 9: Maximum expected value of the relative difference between the k and the $k+1$ nearest neighbor.

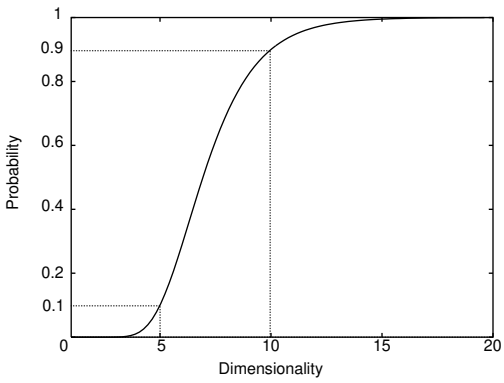


Figure 10: The rejection probability when R_p is 1.84471 and N_c is 48.0277.

This equation shows that the maximum expected value of the relative difference between the k and the $k+1$ nearest neighbor decreases monotonically as the local dimensionality increases (Figure 9). Only 20% or less difference is expected at the 5 dimensions, and 10% or less at 10 dimensions. Thus, the local dimensionality allows us to estimate the relative significance of nearest neighbors. We use this property to decide control-points (ν_c, ρ_c), and (ν_r, ρ_r) as is shown below.

Secondly, we relate the local dimensionality n with the parameters R_p and N_c. When points are distributed uniformly with the local dimensionality n, the probability that N_c points exist in the proximity specified by R_p is obtained as follows:

$$Pr\left\{N_c \text{ points in } R_p\right\} = \left(1 - {R_p}^{-n}\right)^{N_c} \tag{4}$$

The proposed algorithm determines the insignificance of a nearest neighbor when N_c points exist in the proximity specified by R_p. Therefore, this equation is the probability that the algorithm determines the insignificance at the local dimensionality n. Hence, we call this probability the *rejection probability*. The rejection probability increases monotonically as the local dimensionality increases and can be controlled easily by two control-points as follows. Suppose that we want to set the rejection probability to ρ_1 at the dimensionality ν_1 and ρ_2 at the dimensionality ν_2 ($0 < \rho 1 < \rho 2 < 1$ and $1 < \nu_1 < \nu_2$). Then, we can determine the parameters R_p and N_c by solving the following simultaneous equations:

$$(1 - {R_p}^{-\nu_1})^{N_c} = \rho_1 \tag{5}$$

$$(1 - {R_p}^{-\nu_2})^{N_c} = \rho_2 \tag{6}$$

By the elimination of N_c the following equation is obtained:

$$\frac{\log(1 - {R_p}^{-\nu_1})}{\log(1 - {R_p}^{-\nu_2})} = \frac{\log \rho_1}{\log \rho_2} \tag{7}$$

This equation cannot be arithmetically solved. However, since the left side of the equation increases monotonically as R_p increases, it can be easily solved with numerical methods, e.g., Newton's method. Once R_p is determined, N_c is obtained from R_p as follows:

$$N_c = \frac{\log \rho_1}{\log(1 - {R_p}^{-\nu_1})}. \tag{8}$$

Now, by Equation (7) and (8), we can determine R_p and N_c from the two control-points (ν_1, ρ_1) and (ν_2, ρ_2). We propose to set up two control-points as the pair of the cut-off point (ν_c, ρ_c) and the rejection point (ν_r, ρ_r) by analogy with the low pass filter. The range of $n < \nu_c$ is the significance band where the point distribution with the local dimensionality n should be regarded as significant. The range of $\nu_r < n$ is the insignificant band where the point distribution with the local dimensionality n should be regarded as insignificant. The range of $\nu_c < n < \nu_r$ is the transition band. We also propose to determine the value of ν_c, and ν_r from the maximum expected value of the relative difference between the k and the $k+1$ nearest neighbor (Equation (3) and Figure 9). For example, it should be reasonable to set the cut-off dimensionality to 5 and the rejection dimensionality to 10 since only 20% or less difference is expected at the 5 dimensions, and 10% or less at 10 dimensions. When we choose 0.1 as the probability at the cut-off dimensionality and 0.9 as the probability at the rejection dimensionality, two control-points are obtained as (5, 0.1) and (10, 0.9). Then, R_p and N_c can be computed from Equation (7) and (8) as 1.84471 and 48.0277 respectively. Figure 10

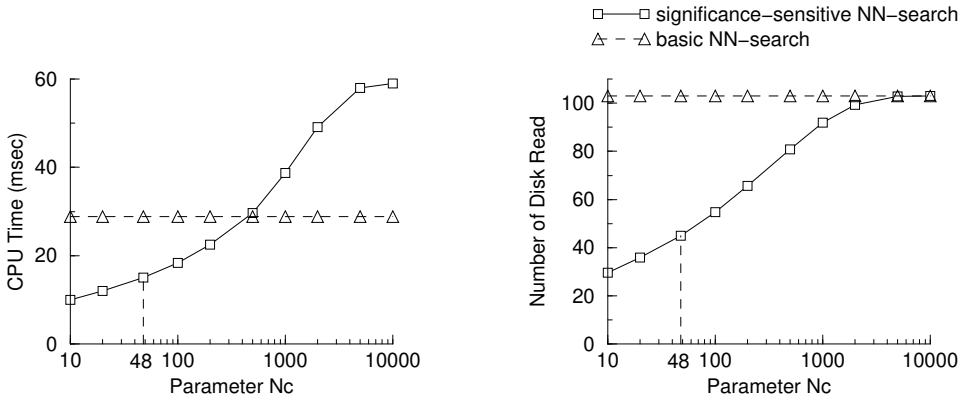


Figure 11: Performance of significance-sensitive nearest-neighbor search.

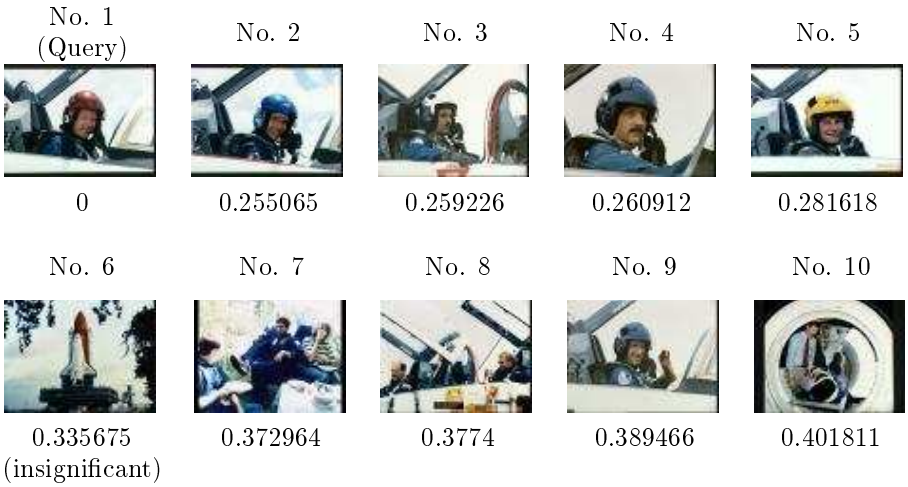


Figure 12: Example of the search result.

shows the rejection probability with these parameters, which is computed from Equation (4).

4 Experimental Evaluation

We evaluated the performance of the significance-sensitive nearest-neighbor search with applying it to the similarity retrieval of images. 40,745 images are collected from the photo and the video archives of NASA (Johnson Space Center, Keneddy Space Center, and NASA Shuttle Web). As for videos, scene changes are detected based on the transition of the color histogram and then representative images of each scene are extracted. The similarity of images is measured in terms of the color histogram. Munsell color system is used for the color space, which consists of hue, saturation, and intensity dimensions. It is divided into nine subspaces: black, gray, white, and six colors. For each image, histograms of four sub-regions, i.e., upper-left, upper-right, lower-left and lower-right, are calculated in order to take account of the composition of the image. Four histograms are concatenated to compose a 36-dimensional feature vector. Then, feature vectors are reduced to 20-dimensional vectors with the principal component analysis. Similarity of images is measured by Euclidean distance among these 20-dimensional vectors. Experiments are conducted on a Sun Microsystems workstation, Ultra 60 (CPU: UltraSPARC-II 360MHz, main memory: 512Mbytes, OS: Solaris 2.6). Programs are implemented in C++. The VAMSplit R-tree[2] is employed as the index structure. We measured the average performance of 1,000 random trials of finding 10 nearest neighbors. Query points are selected at random from the data set. Therefore, one of ten found neighbors is

a query point itself.

Figure 11 shows the result of the experiment. The horizontal axis indicates the parameter N_c which is changed from 10 to 10,000, while the vertical axis indicates the CPU time and the number of disk reads. The parameter R_p is set to 1.84471 according to the optimal setting described above. Changing the parameter N_c corresponds to changing the sensitivity of insignificant nearest neighbors. When the parameter N_c is small, e.g., 10, the detection is more sensitive. When the parameter N_c is large, e.g., 10,000, the detection is less sensitive and fall into no detection of insignificant nearest neighbors. With this setting, the significance-sensitive NN-search is equivalent to the basic NN-search. The result of the basic NN-search is also shown in the figure. Of course, the basic NN-search is independent of the parameter N_c. When N_c is 10,000, the CPU time of the significance-sensitive NN-search is larger than that of the basic NN-search even though their number of disk reads is identical. This is because the significance-sensitive NN-search requires more computational cost in counting the number of points in the proximity. As shown in Figure 11, both CPU time and disk I/O are remarkably reduced with the optimal parameter setting described in Section 3.4, i.e., when N_c is 48. CPU time is reduced by 48% and disk I/O is reduced by 56% compared with the basic NN-search. This result demonstrates the effectiveness of the significance-sensitivity nearest-neighbor search.

Figure 12 shows an example of the search result. The numerical value under each image is the distance from the query to the image. This result is obtained when R_p is 1.84471 and N_c is 48. The significance-sensitive NN-search detected that the image No. 6 is insignificant and answered exact nearest neighbors for No. $1 \sim 5$ and the candidates on the termination for No. $6 \sim 10$. The distance to the first node in the priority queue was 0.312595. This means that more than or equal to 48 points exist within the distance from 0.335675 to 0.576647. Basing on this result, we can consider the images No. $2 \sim 5$ as more significant nearest neighbors and the images No. $6 \sim 10$ as less significant nearest neighbors. This example shows that the significance-sensitive nearest-neighbor search allows us to classify the search result into more significant and less significant nearest neighbors. This capability should be advantageous to the interactive retrieval systems.

5 Conclusion

This paper shows that the insignificant nearest neighbors affect the efficiency of NN-search and that the efficiency can be improved with the employment of a new NN-search algorithm, the significance-sensitive nearest-neighbor search. When the search detects an insignificant nearest neighbor, it terminates the operation and answers the partial result. This reduces both CPU time and disk I/O, and enables us to classify the search result into more significant and less significant nearest neighbors. These capabilities are advantageous especially to the retrieval system with human-interaction. In our future work, we plan to conduct more experiments with various multimedia information in order to clarify the characteristics of the proposed search algorithm in further detail.

References

[1] D. A. White and R. Jain, "Similarity Indexing with the SS-tree," Proc. of the 12th Int. Conf. on Data Engineering, New Orleans, USA, pp.516–523, Feb. 1996.

[2] D. A. White and R. Jain, "Similarity Indexing: Algorithms and Performance," Proc. SPIE Vol.2670, San Diego, USA, pp.62–73, Jan. 1996.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," Proc. of the 22nd VLDB Conf., Bombay, India, pp.28–39, Sep. 1996.

[4] N. Katayama and S. Satoh, "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries," Proc. of the 1997 ACM SIGMOD, Tucson, USA (May 1997) pp. 369–380.

[5] A. Henrich, "The LSD^h-tree: An Access Structure for Feature Vectors," Proc. of the 14th Int. Conf. on Data Engineering, Orlando, USA, pp.362–369, Feb. 1998.

[6] S. Berchtold, C. Bohm, and H.-P. Kriegel, "The Pyramid-Technique: Towards Breaking the Curse of Dimensionality," Proc. ACM SIGMOD, Seattle, USA, pp.142–153, Jun. 1998.

[7] S. Berchtold, C. Bohm, B. Braunmuller, D. Keim, and H.-P. Kriegel, "Fast Parallel Similarity Search in Multimedia Databases," Proc. of the 1997 ACM SIGMOD, Tucson, USA (May 1997) pp. 1–12.

[8] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When Is "Nearest Neighbor" Meaningful?," Proc. of the 7th Int. Conf. on Database Theory, Jerusalem, Israel, pp.217–235, Jan. 1999.

[9] K. Fukunaga, "Introduction to Statistical Pattern Recognition (2nd ed.)," Academic Press, 1990.

[10] G. Hjaltason and H. Samet, "Ranking in Spatial Databases," 4th Int. Symp. on Spatial Databases, SSD'95, Portland, USA, pp.83–95, Aug. 1995.

[11] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An Optimal Algorithm for Approximate Nearest Neighbor Searching," Proc. of the 5th Ann. ACM-SIAM Symposium on Discrete Algorithms, pp.573–582, 1994.