in Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada, Jul. 2001, to appear

Fast Algorithm for Nearest Neighbor Search Based on a Lower Bound Tree

Yong-Sheng Chen†‡ Yi-Ping Hung†‡ Chiou-Shann Fuh‡

†Institute of Information Science, Academia Sinica, Taipei, Taiwan
‡Dept. of Computer Science and Information Engineering, National Taiwan University, Taiwan
Email: hung@iis.sinica.edu.tw

Abstract

This paper presents a novel algorithm for fast nearest neighbor search. At the preprocessing stage, the proposed algorithm constructs a lower bound tree by agglomeratively clustering the sample points in the database. Calculation of the distance between the query and the sample points can be avoided if the lower bound of the distance is already larger than the minimum distance. The search process can thus be accelerated because the computational cost of the lower bound, which can be calculated by using the internal node of the lower bound tree, is less than that of the distance. To reduce the number of the lower bounds actually calculated, the winner-update search strategy is used for traversing the tree. Moreover, the query and the sample points can be transformed for further efficiency improvement. Our experiments show that the proposed algorithm can greatly speed up the nearest neighbor search process. When applying to the real database used in Nayar's object recognition system, the proposed algorithm is about one thousand times faster than the exhaustive search.

1. Introduction

Nearest neighbor search is very useful in recognizing objects [12], matching stereo images [15], classifying patterns [9], compressing images [10], and retrieving information in database system [7]. In general, given a fixed data set P which consists of s sample points in a d-dimensional space, that is, $P = \{\mathbf{p}_i \in R^d | i = 1, \ldots, s\}$, preprocessing can be performed to construct a particular data structure. For each query point $\mathbf{q}$, the goal of the nearest neighbor search is to find in P the point closest to $\mathbf{q}$. The straightforward way to find the nearest neighbor is to exhaustively compute and compare the distances from the query point to all the sample points. The computational complexity of this exhaustive search is $O(s \cdot d)$. When s, d, or both are large, this process becomes very computational-intensive.

In the past, Bentley [2] proposed the k-dimensional binary search tree to speed up the the nearest neighbor search. This method is very efficient when the dimension of the data space is small. However, as reported in [13] and [3], its performance degrades exponentially with increasing dimension. Fukunaga and Narendra [8] constructed a tree structure by repeating the process of dividing the set of sample points into subsets. Given a query point, they used a branch-and-bound search strategy to efficiently find the closest point by using the tree structure. Djouadi and Bouktache [5] partitioned the underlying space of the sample points into a set of cells. A few cells located in the vicinity of the query point can be determined by calculating the distances between the query point and the centers of the cells. The nearest neighbor can then be found by searching only in these neighboring cells, instead of the whole space. Nene and Nayar [13] proposed a very fast algorithm to search for the nearest neighbor within a pre-specified distance threshold. From the first to the last dimension, their method can exclude the sample points that the distances from those sample points to the query point at the current dimension are larger than the threshold. Then, the nearest neighbor can be determined by examining the remaining candidates. Lee and Chae [11] proposed another elimination-based method. Based on triangle inequality, they used a number of anchor sample points to eliminate many distance calculations. Instead of finding the *exact* nearest neighbor, Arya et al. [1] proposed a fast algorithm which can find the *approximate* nearest neighbor within a factor of $(1 + r)$ of the distance between the query point and its exact nearest neighbor.

In this paper, we present a novel algorithm which can efficiently search for the exact nearest neighbor in Euclidean space. At the preprocessing stage, the proposed algorithm constructs a lower bound tree (LB-tree), in which each leaf node represents a sample point and each internal node represents a mean point in a space of smaller dimension. Given a query point, the lower bound of its distance to each sample point can be calculated by using the mean point of the internal node in the LB-tree. Calculation of the distance between the query and the sample points can be avoided if the lower

bound of the distance is already larger than the minimum distance between the query point and its nearest neighbor. Because the computational cost of the lower bound is less than that of the distance, the whole search process can be accelerated.

Furthermore, we adopt the following three techniques to reduce the number of the lower bounds actually calculated. The first one is the winner-update search strategy [4] for traversing the LB-tree. We adopt this search strategy to reduce the number of the nodes examined. Starting from the root node of the LB-tree, the node having the minimum lower bound is chosen and its children will then join the competition after their lower bounds having been calculated. The second one is the agglomerative clustering technique for the LB-tree construction. The major advantage of this technique is that it can keep the number of the internal nodes as small as possible while keeping the lower bound as tight as possible. The last technique we adopt is data transformation. By applying data transformation to each point, the lower bound of an internal node can be further tightened, and thus save more computation. We use two kind of data transformation in this work: wavelet transform and principal component analysis.

Our experiments show that the proposed algorithm can save substantial computation of the nearest neighbor search, in particular, when the distance of the query point to its nearest neighbor is relatively small compared with its distance to most other samples.

2. Multilevel structure and LB-tree

This section introduces the LB-tree, the essential data structure used in the proposed nearest neighbor search algorithm. We will first describe the multilevel structure of a data point. The multilevel structures of all the sample points can be used for the LB-tree construction.

2.1. Multilevel structure of each point

For a point $\mathbf{p} = [p_1, p_2, \ldots, p_d]$ in a d-dimensional Euclidean space, R^d, we define its multilevel structure, denoted by $\{\mathbf{p}^0, \mathbf{p}^1, \ldots, \mathbf{p}^L\}$, in the following way. At each level l, $\mathbf{p}^l$ comprises the first d^l dimensions of the point $\mathbf{p}$, where the integer d^l satisfies $1 \leq d^l \leq d$, $l = 0, \ldots, L$. That is, $\mathbf{p}^l = [p_1, p_2, \ldots, p_{d^l}]$, which will be referred to as the level-l projection of $\mathbf{p}$.

In this paper, we assume that the dimension of the underlying space, d, is equal to 2^L without loss of generality. Zero padding can be used to enlarge the dimension if d is not a power of 2. In the multilevel structure, the dimension at level l is set to be $d^l = 2^l$. In this way, a $(L+1)$-level structure of triangle shape can be constructed for point $\mathbf{p}$. Notice that level-L projection, $\mathbf{p}^L$, is the same as the point

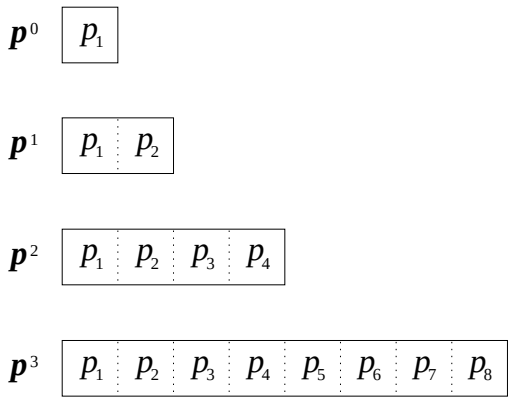

Figure 1. An example of the 4-level structure of the point $\mathbf{p}$, where $\mathbf{p} \in R^8$.

$\mathbf{p}$. Figure 1 illustrates an example of the 4-level structures, $\{\mathbf{p}^0, \ldots, \mathbf{p}^3\}$, where $d = 8$.

Given the multilevel structures of two points $\mathbf{p}$ and $\mathbf{q}$, we can derive the following inequality property.

Property 1 *The Euclidean distance between $\mathbf{p}$ and $\mathbf{q}$ is larger than or equal to the Euclidean distance between their level-l projections $\mathbf{p}^l$ and $\mathbf{q}^l$ for each level l. That is,*

$$\|\mathbf{p} - \mathbf{q}\|_2 \geq \|\mathbf{p}^l - \mathbf{q}^l\|_2, \; l = 0, \ldots, L.$$

From Property 1, the distance $\|\mathbf{p}^l - \mathbf{q}^l\|_2$ between the level-l projections can be considered as a lower bound of the distance $\|\mathbf{p} - \mathbf{q}\|_2$ between the points $\mathbf{p}$ and $\mathbf{q}$. Notice that the computational cost of the distance $\|\mathbf{p}^l - \mathbf{q}^l\|_2$ is less than that of the distance $\|\mathbf{p} - \mathbf{q}\|_2$. To be specific, the complexity of calculating the distance between level-l projections arises from $O(2^0)$ to $O(2^L)$ as l varies from 0 to L.

2.2. LB-tree for the data set

To construct an LB-tree, we need to use the multilevel structures of all the sample points $\mathbf{p}_i$, $i = 1, \ldots, s$, in the data set P. Suppose the multilevel structure has $L+1$ levels, that is, from level 0 to level L. Then, the LB-tree also has $L+1$ levels without considering the dummy root node having zero dimension. Each leaf node at level L in the LB-tree contains a level-L projection ${\mathbf{p}_i}^L$, which is exactly the same as the sample point $\mathbf{p}_i$. The level-l projections, ${\mathbf{p}_i}^l$, $i = 1, \ldots, s$, of all the sample points can be hierarchically clustered from level 0 to level $L-1$, as illustrated in Figure 2. More discussions on the LB-tree construction will be given in Section 4.

Let $\langle \mathbf{p} \rangle$ denote the node containing the point $\mathbf{p}$ in the LB-tree. Each cluster ${C^l}_j$ is represented by an internal node

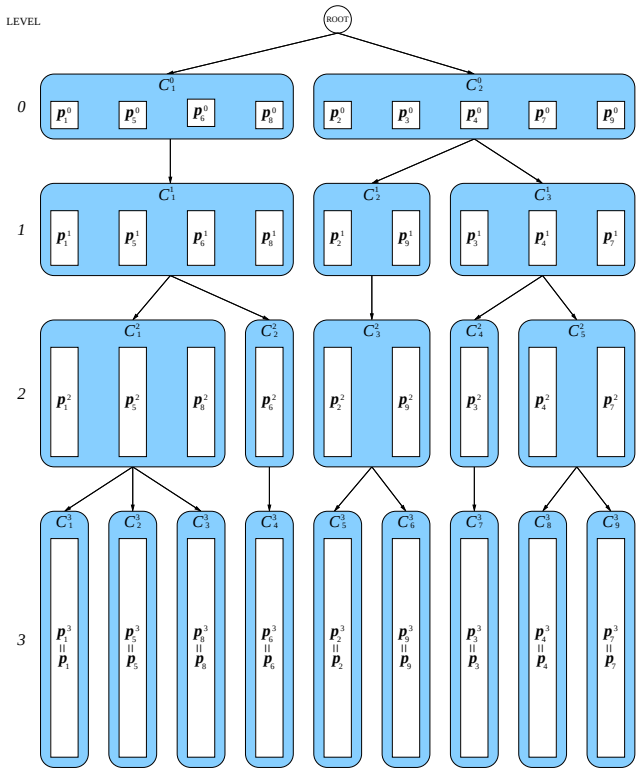

Figure 2. An example of hierarchical construction of the LB-tree. All the points in the same dark region are determined agglomeratively and are grouped into a cluster. Notice that each point is transposed to fit into the limited space.

$\langle \mathbf{m}^l{}_j \rangle$ at level l in the LB-tree, where $j = 1, \ldots, s^l$ and s^l denote the number of clusters at level l. As shown in Figure 3, the internal node $\langle \mathbf{m}^l{}_j \rangle$ contains the mean point $\mathbf{m}^l{}_j$ and the associated *radius*, $r^l{}_j$. The mean point $\mathbf{m}^l{}_j$ is the mean of all the level-l projections of the sample points contained in cluster $C^l{}_j$. The radius $r^l{}_j$ is the radius of the smallest hyper-sphere that centers at the mean point $\mathbf{m}^l{}_j$ and covers all the level-l projections in cluster $C^l{}_j$. This smallest hyper-sphere is called the *bounding sphere* of $C^l{}_j$. The radius $r^l{}_j$ can be calculated as the maximum distance from the mean point $\mathbf{m}^l{}_j$ to all the level-l projections in this cluster. In other words, the LB-tree has the following inequality property.

Property 2 *Given a sample point $\mathbf{p}^*$, the distance between its level-l projection, $\mathbf{p}^{*l}$, and its level-l ancestor, $\mathbf{m}^l{}_{j^*}$, is smaller than or equal to the radius of the bounding sphere of cluster $C^l{}_{j^*}$. That is,*

$$\|\mathbf{p}^{*l} - \mathbf{m}^l{}_{j^*}\|_2 \leq r^l{}_{j^*},\ l = 0, \ldots, L.$$

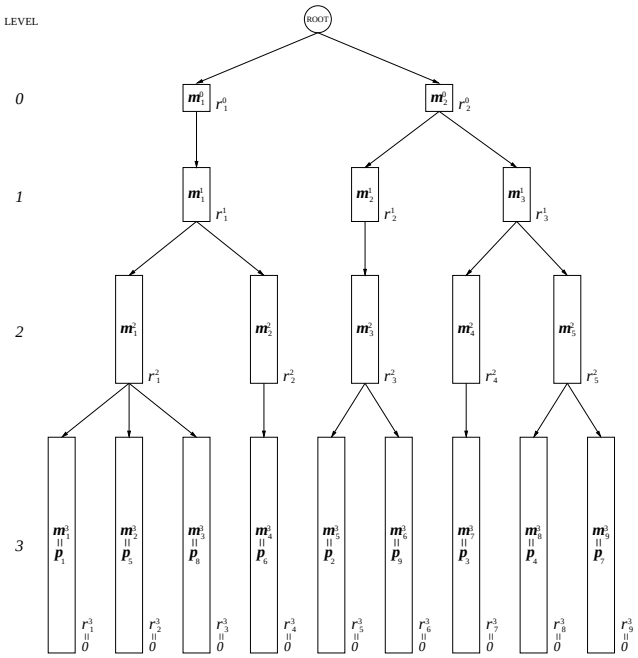

Figure 3. An example of the LB-tree.

Notice that each leaf node can be viewed as a cluster having only one point. In this case, the mean point is the sample point itself and the radius is zero.

Now, suppose we are given a query point $\mathbf{q}$. The first step is to form its multilevel structure as described in Section 2.1. Consider a sample point $\mathbf{p}^*$ and its corresponding leaf node $\langle \mathbf{p}^* \rangle$. Its ancestor at level l in the LB-tree can be found, and let it be denoted by $\langle \mathbf{m}^l{}_{j^*} \rangle$. As illustrated in Figure 4, we can derive the following inequality by using the triangle inequality and Properties 1 and 2:

$$\begin{aligned} \|\mathbf{p}^* - \mathbf{q}\|_2 &\geq \|\mathbf{p}^{*l} - \mathbf{q}^l\|_2 \\ &\geq \|\mathbf{m}^l{}_{j^*} - \mathbf{q}^l\|_2 - \|\mathbf{p}^{*l} - \mathbf{m}^l{}_{j^*}\|_2 \\ &\geq \|\mathbf{m}^l{}_{j^*} - \mathbf{q}^l\|_2 - r^l{}_{j^*}. \end{aligned} \quad (1)$$

Let $d_{LB}(\langle \mathbf{m}^l{}_{j^*} \rangle, \mathbf{q}^l)$ denote the LB-distance between the internal node $\langle \mathbf{m}^l{}_{j^*} \rangle$ and $\mathbf{q}^l$, which is defined below:

$$d_{LB}(\langle \mathbf{m}^l{}_{j^*} \rangle, \mathbf{q}^l) \equiv \|\mathbf{m}^l{}_{j^*} - \mathbf{q}^l\|_2 - r^l{}_{j^*}. \quad (2)$$

We then have the following inequality property.

Property 3 *Given a query point $\mathbf{q}$ and a sample point $\mathbf{p}^*$, the LB-distance between the level-l ancestor of $\mathbf{p}^*$ (that is, $\langle \mathbf{m}^l{}_{j^*} \rangle$) and the level-$l$ projection of $\mathbf{q}$ is smaller than or equal to the distance between $\mathbf{p}^*$ and $\mathbf{q}$. That is,*

$$d_{LB}(\langle \mathbf{m}^l{}_{j^*} \rangle, \mathbf{q}^l) \leq \|\mathbf{p}^* - \mathbf{q}\|_2,\ l = 0, \ldots, L.$$

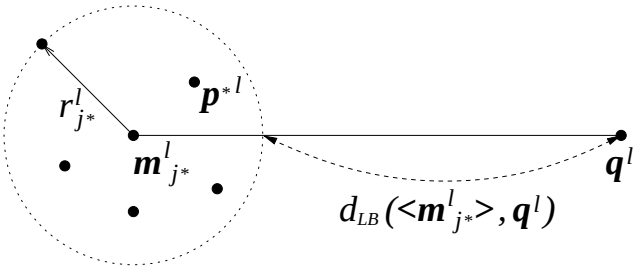

Figure 4. Illustration of the distance inequality of Equation (1).

From Property 3, we know that $d_{LB}(\langle \mathbf{m}^l{}_{j^*} \rangle, \mathbf{q}^l)$ can be viewed as a lower bound of the distance between $\mathbf{p}^*$ and $\mathbf{q}$. Notice that LB-distance is not a valid distance metric and a negative $d_{LB}(\langle \mathbf{m}^l{}_{j^*} \rangle, \mathbf{q}^l)$ implies that the query point $\mathbf{q}$ locates within the bounding sphere of $C^l{}_{j^*}$ centered at $\mathbf{m}^l{}_{j^*}$.

Because the internal node $\langle \mathbf{m}^l{}_{j^*} \rangle$ can have a number of descendants, $d_{LB}(\langle \mathbf{m}^l{}_{j^*} \rangle, \mathbf{q}^l)$ is not only the lower bound of the distance to $\mathbf{q}$ for any particular sample point $\mathbf{p}^*$, but also the lower bound of the distances to $\mathbf{q}$ for all the sample points in the cluster $C^l{}_{j^*}$ containing $\mathbf{p}^*$. Hence, we have the following property by using Property 3.

Property 4 *Let* $\mathbf{q}$ *be a query point and* $\hat{\mathbf{p}}$ *be a sample point. For any internal node* $\langle \mathbf{m}^l{}_j \rangle$ *of the LB-tree, if*

$$d_{LB}(\langle \mathbf{m}^l{}_j \rangle, \mathbf{q}^l) > \|\hat{\mathbf{p}} - \mathbf{q}\|_2,$$

then, for every descendant leaf node $\langle \mathbf{p}' \rangle$ *of* $\langle \mathbf{m}^l{}_j \rangle$*, we have*

$$\|\mathbf{p}' - \mathbf{q}\|_2 > \|\hat{\mathbf{p}} - \mathbf{q}\|_2.$$

From Property 4, we can see that if the LB-distance of the internal node $\langle \mathbf{m}^l{}_j \rangle$ is already larger than the distance from the sample point $\hat{\mathbf{p}}$ to the query point $\mathbf{q}$, all the descendant leaf node $\langle \mathbf{p}' \rangle$ of $\mathbf{m}^l{}_j$ can be eliminated in the contest. They have no chance to be the winner because there is already a better candidate, $\hat{\mathbf{p}}$, which is closer to $\mathbf{q}$.

3. Proposed algorithm

At the preprocessing stage, the proposed algorithm constructs an LB-tree of $L + 1$ levels by using the data set P. For each query point $\mathbf{q}$, the proposed algorithm uses the LB-tree to efficiently find its nearest neighbor, $\hat{\mathbf{p}}$, such that the Euclidean distance $\|\hat{\mathbf{p}} - \mathbf{q}\|_2$ is minimum. According to Property 4, if the LB-distance of an internal node $\langle \mathbf{m}^l{}_j \rangle$ is larger than the minimum distance, $\|\hat{\mathbf{p}} - \mathbf{q}\|_2$, all the descendant samples of the node $\langle \mathbf{m}^l{}_j \rangle$ can not be the nearest neighbor. Hence, the costly calculation of the distances between $\mathbf{q}$ and many sample points can be saved at the expense of calculating the less-expensive LB-distances.

However, the above saving requires the knowledge of the value $\|\hat{\mathbf{p}} - \mathbf{q}\|_2$, and we do not know which sample point is $\hat{\mathbf{p}}$ beforehand. In fact, $\hat{\mathbf{p}}$ is exactly the nearest neighbor which we would like to find. To achieve the same effect, we adopt the winner-update search strategy to compute the lower bounds from the root node toward the leaf nodes while traversing the LB-tree. Based on the following two observations, we calculate the LB-distances of the internal nodes from the top level to the bottom level. First, the computation of the LB-distance costs less at the upper level. Second, a node at the upper level has more descendants in general. Thus, more distance calculation can be saved if the LB-distance of an upper-level node is already larger than the minimum distance.

In the following, we will describe the winner-update search strategy, which is a best-first search strategy, that can greatly reduce the number of the LB-distances actually calculated. At the first iteration, the LB-distances between $\mathbf{q}^0$ and all the level-0 nodes in the LB-tree are calculated by using Equation (2). These level-0 nodes, $\langle \mathbf{m}^0{}_1 \rangle$, $\langle \mathbf{m}^0{}_2 \rangle$, ..., $\langle \mathbf{m}^0{}_{s^0} \rangle$, are used to construct a heap data structure and the root node of the heap, $\langle \hat{\mathbf{p}} \rangle$, is the node having the minimum LB-distance. Then, at the next iteration, the node $\langle \hat{\mathbf{p}} \rangle$ is deleted and its children are inserted into the heap. The LB-distances of these child nodes are calculated and the heap is rearranged to maintain the heap property. The node $\langle \hat{\mathbf{p}} \rangle$ is updated accordingly to be the new root node of the heap having the minimum LB-distance. Again, the node $\langle \hat{\mathbf{p}} \rangle$ is deleted and all its children are inserted. This procedure is repeated until the dimension of $\langle \hat{\mathbf{p}} \rangle$, $dim(\langle \hat{\mathbf{p}} \rangle)$, is equal to d. At this point, the node $\langle \hat{\mathbf{p}} \rangle$ is a leaf node containing a sample point and the distance $\|\hat{\mathbf{p}} - \mathbf{q}\|_2$ is the minimum in the heap. Since the LB-distances of other nodes in the heap, which are the lower bounds of the distances from all the other sample points to the query point $\mathbf{q}$, are already larger than $\|\hat{\mathbf{p}} - \mathbf{q}\|_2$, the nearest neighbor $\hat{\mathbf{p}}$ can be determined. The proposed algorithm is summarized below.

Proposed Algorithm for Nearest Neighbor Search

```
    /* Preprocessing Stage */
100 Given a data set P = {p_i ∈ R^d | i = 1, ..., s}
110   Construct the LB-tree of L + 1 levels for P

    /* Nearest Neighbor Search Stage */
120 Given a query point q
130   Construct the (L + 1)-level structure of q
140   Insert the root node of the LB-tree into an
        empty heap
150   Let ⟨p̂⟩ be the root node of the heap
160   while dim(⟨p̂⟩) < d do
```

170 Delete the node $\langle \hat{\mathbf{p}} \rangle$ from the heap
180 Calculate the LB-distances for all the children of $\langle \hat{\mathbf{p}} \rangle$
190 Insert all the children of $\langle \hat{\mathbf{p}} \rangle$ into the heap
200 Rearrange the heap to maintain the heap property that the root node is the one having the minimum LB-distance
210 Update $\langle \hat{\mathbf{p}} \rangle$ as the root node of the heap
220 **endwhile**
230 Output $\hat{\mathbf{p}}$

For conciseness of the pseudo code, the algorithm described above initializes the heap as the dummy root node of the LB-tree, instead of the level-0 nodes. This makes no difference because the dummy root node of the LB-tree is replaced immediately by its children, that is, all the level-0 nodes, at the first iteration of the loop.

4. Construction of LB-tree

A simple method of constructing the LB-tree is to directly use the multilevel structures of the sample points without clustering. That is, there are s internal nodes at each level l. Each internal node contains exactly one level-l projection, ${\mathbf{p}_i}^l$, $i = 1, \ldots, s$, and its radius is set to be zero. In the constructed LB-tree, each internal node has only one child node except the root node, which has s child nodes.

Recall that from Properties 3 and 4 the LB-distance for each internal node is the lower bound of the distances between $\mathbf{q}$ and all the descendant samples of this internal node. In order to obtain a tighter lower bound to skip more distance calculation, the LB-distance of each internal node should be as large as possible. Therefore, from Equation (2) the radius of the bounding sphere, ${r^l}_j$, should be as small as possible. From this point of view, we should adopt the simple construction method described above in which the radius of each internal node is zero. However, there would be too many internal nodes in this case, and the cost will be too high in terms of the memory storage and the computation of the LB-distance for these internal nodes. Hence, we would also like to keep the number of the internal nodes as small as possible. When constructing an LB-tree, consequently, we should consider the trade-off between the number of the internal nodes and their associated radius of the bounding sphere.

The construction of the LB-tree is performed in the preprocessing stage, and hence, its computational cost is not a major concern here. In this work, we use an agglomerative clustering technique [6] for constructing the LB-tree, in which both the number of the internal nodes and the associated radius are both small. As shown in Figure 2, the LB-tree is constructed hierarchically from top level to bottom level. At each level l, the level-l projections are agglomeratively grouped into clusters. The explanation of the detail is quite lengthy, and is not given here due to the limitation of space.

5. Data transformation

Data transformation can further improve the efficiency of the proposed algorithm for nearest neighbor search. Remember that the Euclidean distance calculated at level l in the LB-tree is actually the distance in the subspace of the first 2^l dimensions. If these dimensions are not discriminative enough (that is, the projections of the sample points on this subspace are too close to each other), the distances of different samples calculated at this subspace may be almost the same. Therefore, these distances computed in the 2^l-subspace will not help in the determination of the nearest neighbor. This problem can be alleviated by transforming the data points into another space such that the anterior dimensions are likely to be more discriminative than the posterior dimensions. The Euclidean distances calculated in either the transformed space or the original space should be the same, thus will not affect the final search result of the nearest neighbor but only efficiency. Furthermore, this transformation should not be too computation-expensive because the query points have to be transformed in the query process. The following lists the pseudo code for the data transformation, which are to be added to the algorithm given in Section 3:

105 Transform each sample point

125 Transform the query point $\mathbf{q}$

According to the contents of the data, we propose two types of data transformation. When the data point represents an autocorrelated signal, for example, an audio signal or an image block, wavelet transform with orthogonal basis [14] can be used. In this work, we adopt Haar wavelets to transform the data, and then each level in the multilevel structure represents the data in one of its multiple resolutions.

The second type of data transformation is the principal component analysis (PCA). Performing PCA can find a set of vectors ordered in their ability to account for the variation of data projected on those vectors. We can transform the data point onto the space spanned by this set of vectors such that the anterior dimensions are more discriminative than the posterior dimensions.

6. Experimental results

This section presents some experimental results obtained by using a computer-generated set of autocorrelated data (Section 6.1) and by using a real data set acquired from

a object recognition system (Section 6.2). These experiments were conducted on a PC with a Pentium III 700 MHz CPU. Instead of using the amount of distances calculated, we compare the efficiency of different algorithms by using the execution time. The reasons are: first, our algorithm has some overhead, including the insertion and deletion of elements in the heap, rearranging the heap, and updating the node $\langle \hat{\mathbf{p}} \rangle$; and second, the computational cost of the LB-distance of the node at some level differs from that of the node at another level.

6.1. Experiments on autocorrelated data

In this section, we show three experimental results to compare the performance of the proposed algorithm as the following three factors vary: the number of the sample points in the data set, s; the dimension of the underlying space, d; and the average of the minimum distances between the query points and their nearest neighbors, $\overline{\varepsilon_{min}}$. In these experiments, we randomly generate autocorrelated data points to simulate real signal. For each data point, its value of the first dimension is randomly generated from a uniform distribution with extent $[-1, 1]$. The value of each subsequent dimension is assigned as the value of the previous dimension added by a normally distributed noise with zero mean and variance 0.1. The value of each dimension beyond the extent $[-1, 1]$ is truncated. In order to see the influence of data transformation on the search efficiency for autocorrelated data, we construct two kinds of multilevel structures for each data point—one with Haar transform, and the other without any data transformation.

In the first experiment, we generated seven data sets of sample points with cardinalities $s = 800$, 1600, 3200, ..., 51200 by using the random process described above. The dimension of the underlying space, d, was 32. Another set containing $100,000$ query points were also generated by using the same random process. Nearest neighbor search was then performed for each query point. The mean query time is shown in Figure 5. The query time for the proposed algorithm has taken into account both the Haar transform, if applied, and the search process. To demonstrate the influence of the agglomerative clustering and the Haar transform on the search efficiency, we show the mean query time resulted by using different versions of the proposed algorithm. The first version adopts the agglomerative clustering but not the data transformation, and is denoted by "+clustering-Haar". The second version adopts the Haar transform but not the agglomerative clustering (that is, use the simplest method for LB-tree construction mentioned in Section 4), and is denoted by "-clustering+Haar". The third version adopts both the agglomerative clustering and the Haar transform, and is denoted by "+clustering+Haar". In this experiment, the proposed algorithm (the "+clustering+Haar" version) is 11.7 and 52.8 times faster than the exhaustive search algorithm, when s is 800 and $51,200$, respectively. When s increases (from 800 to $51,200$), there are more sample points scattered in the fixed space. Therefore, the average of the minimum distances, $\overline{\varepsilon_{min}}$, decreases (from 0.91 to 0.53). According to Property 4, the LB-distance is more likely to be larger than the minimum distance of the query point $\mathbf{q}$ to its nearest neighbor $\hat{\mathbf{p}}$ when the minimum distance is smaller. That is, more distance calculations can be avoided if $\overline{\varepsilon_{min}}$ is smaller. This is the reason why the speedup factor increases as s increases.

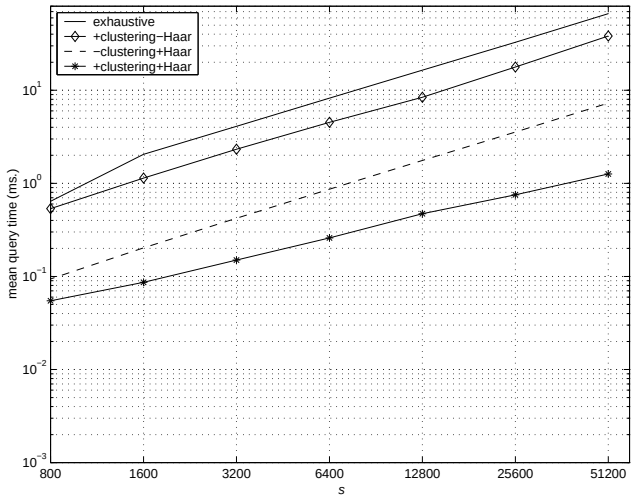

Figure 5. Mean query time for different sizes, s, of the sample point set ($d = 32$).

In the second experiment, we generated eight data sets of $10,000$ sample points, each set generated with a different dimension, $d = 2$, 4, 8, ..., 256. Also, eight corresponding sets of $100,000$ query points, with dimensions $d = 2, 4, 8, \ldots, 256$, were generated by using the same random process. As shown in Figure 6, the proposed algorithm (the "+clustering+Haar" version) apparently outperforms the exhaustive search. It is interesting to note that, for autocorrelated data, our algorithm does not suffer the curse of dimensionality that k-dimensional binary search tree algorithm suffers, as reported in [13, 3]. In fact, the computational speedup of the proposed algorithm (over the exhaustive algorithm) scales up from 5.5 to 63.5 (the "+clustering+Haar" version) as d increases from 2 to 256. When d increases, the level number of the multilevel structure and of the constructed LB-tree also increases. By using Haar transform, the anterior dimensions contain more significant components of the autocorrelated data. Consequently, the lower bound of the distance calculated at the upper level can be tighter in this way. Distance calculation for more

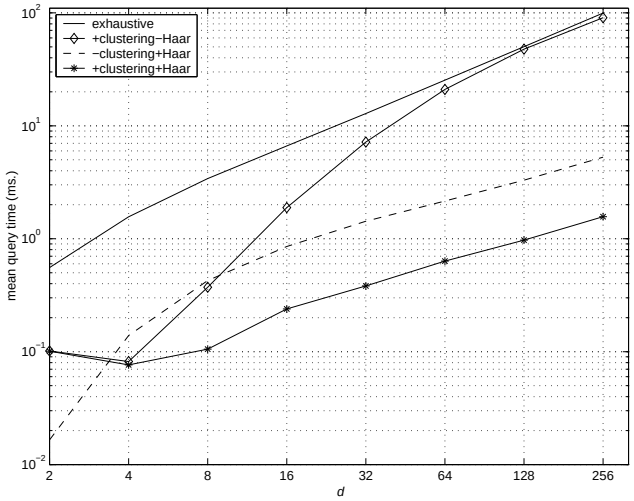

Figure 6. Mean query time for different dimension of the underlying space, d. ($s = 10,000$)

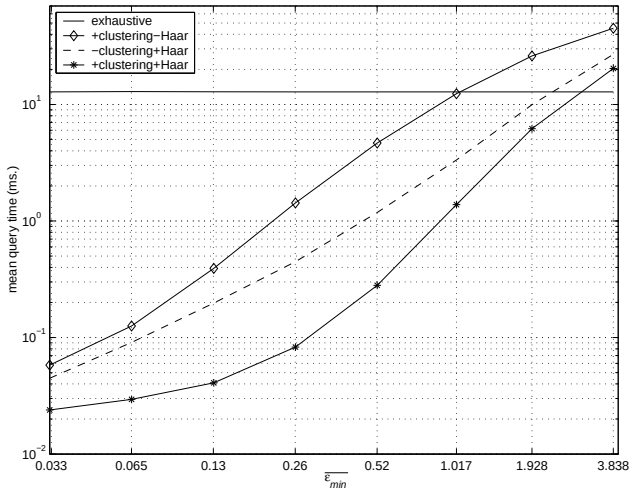

Figure 7. Mean query time for different mean of the minimum distances, $\overline{\varepsilon_{min}}$. ($s = 10,000$, $d = 32$)

sample points therefore can be avoided by calculating only a few LB-distances of their ancestors on the upper level, except for a few tough competitors. If Haar transform is not applied (i.e., the "+clustering-Haar" version), each dimension of the data point is equally significant. Thus the LB-distance at the lower level (which requires more computation) needs to be calculated to determine the nearest neighbor, which then degrades the performance. Also, the agglomerative clustering from top to down is more effective if data transformation is applied so that the anterior dimensions contain more significant components. For the "+clustering-Haar" version, there are more internal nodes, comparing to that for the the "+clustering+Haar" version, thus will reduce its efficiency. When d increases, this phenomenon becomes more amplified, thus the speedup factor for the non-transform version drops dramatically but not for the transform version.

The third experiment shows how the proposed algorithm performs with respect to $\overline{\varepsilon_{min}}$. We generated a data set of $10,000$ sample points in a space of dimension $d = 32$. Then, each sample point was used for generating a query point by adding to each coordinate a uniformly distributed noise with extent $[-e, e]$. When e is large, the distance between the query point and its nearest neighbor tends to be large also. In this experiment, we generated eight sets of $10,000$ query points, each with a different e, $e = 0.01, 0.02, 0.04, \ldots, 1.28$. The mean query time versus the mean of the minimum distances, $\overline{\varepsilon_{min}}$, for different versions of our algorithm is shown in Figure 7. When e increases from 0.01 to 1.28, $\overline{\varepsilon_{min}}$ increases from 0.033 to 3.838. The computational cost of the proposed algorithm increases because the LB-distance is less likely to be larger than the minimum distance when the minimum distance of the nearest neighbor is already very large. Thus, less distance calculation can be saved. In this case, the speedup factor of the proposed algorithm (the "+clustering+Haar" version), compared with the exhaustive algorithm, decreases from 537 to 0.63. Notice that the noise extent, $[-1.28, 1.28]$, is larger than the data extent, $[-1, 1]$, when the speedup factor becomes 0.63. For most applications, $\overline{\varepsilon_{min}}$ is usually relatively small. Therefore, the phenomenon that the proposed algorithm does not outperform the exhaustive search algorithm, as shown on the right part in Figure 7, is not likely to happen for most applications.

6.2. Experiments on an object recognition database

The experiments described in this section adopted the same database used in [12, 13]. This database was generated from $7,200$ images of 100 objects. For each object, 72 images were taken at different poses. Each of these $7,200$ images is of size 128×128. Each image was represented in vector form and was normalized to unit length. These normalized vectors can be used to compute an eigenspace of dimension 35. Then, each vector can be compressed from $16,384$ dimensions to 35 dimensions by projecting onto the eigenspace. In the eigenspace, the manifold for each object can be constructed by using the 72 vectors belonging to the object. Each of the 100 manifolds was sampled to obtain 360 vectors for each object. All the $s = 36,000$ sampled vectors constitute the data set, here each sample point has

dimension $d = 35$.

The set of query points can be generated by first uniformly sampling the manifolds and then adding random noise to each coordinate. We first sampled each of the 100 manifolds at $3,600$ equally spaced positions and then added to each coordinate a uniformly distributed noise with extent $[-0.005, 0.005]$. In this way, we can have a set of $360,000$ query points.

The proposed algorithm and the exhaustive search algorithm were used for performing the nearest neighbor search, and the mean query time is 0.046 ms and 50.048 ms, respectively. In this case, our algorithm is $1,088$ times faster than the exhaustive search algorithm. This performance is roughly 18 times faster than the result attained by Nene and Nayar [13], in which their method is about 61 times faster than the exhaustive search. The construction time of the LB-tree using those $36,000$ sample points of dimension 35 is $11,679$ seconds and the number of clusters, s^l, at level l of the LB-tree is $s^l = 20, 245, 2456, 4684, 5716, 7019, 36000$, $l = 0, 1, \ldots, 6$. Although the construction time is still acceptable in this case, more work should be done to improve the efficiency of the LB-tree construction when dealing with a larger sample point set.

7. Conclusions

In this paper, we have proposed a fast algorithm for nearest neighbor search. This algorithm adopts the winner-update search strategy to traverse an LB-tree created by using agglomerative clustering. For further speedup of the search process, some kind of data transformation, such as Haar transform (for autocorrelated data) and PCA (for general object recognition data), is applied to sample points and query points.

According to our experiments, the proposed algorithm dramatically speed up the search process, in particular, when the distance of the query point to its nearest neighbor is relatively small compared with its distance to most other samples. In many object recognition applications, a query point of an object is close to the sample points belonging to the same object, but is far from the sample points of other objects. This makes the proposed algorithm more appealing and practical. In this paper, we have applied our algorithm to the object recognition database used in [12, 13]. The proposed algorithm is about one thousand times faster than the exhaustive search, which is about eighteen times faster than the result achieved in [13].

We believe that the proposed algorithm can be very helpful in content-based retrieval from a large image or audio database, such as [7, 16], where each sample point represents an autocorrelated signal. In this kind of applications, the dimension d and the number of sample points s are both large. Our algorithm can provide very efficient search for a given query image or audio.

References

[1] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] S. Berchtold, D. A. Keim, H.-P. Kriegel, and T. Seidl. Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):45–57, 2000.

[4] Y.-S. Chen, Y.-P. Hung, and C.-S. Fuh. Fast block matching algorithm based on the winner-update strategy. *IEEE Transactions on Image Processing*, to appear.

[5] A. Djouadi and E. Bouktache. A fast algorithm for the nearest-neighbor classifier. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):277–282, 1997.

[6] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, New York, second edition, 2001.

[7] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC sytstem. *IEEE Computer*, 28(9):23–32, 1995.

[8] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, 24:750–753, 1975.

[9] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):607–616, 1996.

[10] C.-H. Hsieh and Y.-J. Liu. Fast search algorithms for vector quantization of images using multiple triangle inequalities and wavelet transform. *IEEE Transactions on Image Processing*, 9(3):321–328, 2000.

[11] E.-W. Lee and S.-I. Chae. Fast design of reduced-complexity nearest-neighbor classifiers using triangular inequality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):567–571, 1998.

[12] H. Murase and S. K. Nayar. Visual learning and recognition of 3-D objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.

[13] S. A. Nene and S. K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):989–1003, 1997.

[14] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Wellesley, Massachusetts, 1996.

[15] C. Tomasi and R. Manduchi. Stereo matching as a nearest-neighbor problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):333–340, 1998.

[16] H. D. Wactlar, T. Kanade, M. A. Smith, and S. M. Stevens. Intelligent access to digital video: Informedia project. *IEEE Computer*, 29(5):46–52, 1996.