Pattern Recognition 43 (2010) 2351–2358

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Fast exact k nearest neighbors search using an orthogonal search tree

Yi-Ching Liaw *, Maw-Lin Leou, Chien-Min Wu

Department of Computer Science and Information Engineering, Nanhua University, Chiayi, Taiwan 622, ROC

A R T I C L E   I N F O

Article history:
Received 29 July 2009
Received in revised form
4 January 2010
Accepted 7 January 2010

Keywords:
k nearest neighbors
Fast algorithm
Principal axis search tree
Orthonormal basis

A B S T R A C T

The problem of k nearest neighbors (kNN) is to find the nearest k neighbors for a query point from a given data set. In this paper, a novel fast kNN search method using an orthogonal search tree is proposed. The proposed method creates an orthogonal search tree for a data set using an orthonormal basis evaluated from the data set. To find the kNN for a query point from the data set, projection values of the query point onto orthogonal vectors in the orthonormal basis and a node elimination inequality are applied for pruning unlikely nodes. For a node, which cannot be deleted, a point elimination inequality is further used to reject impossible data points. Experimental results show that the proposed method has good performance on finding kNN for query points and always requires less computation time than available kNN search algorithms, especially for a data set with a big number of data points or a large standard deviation.

1. Introduction

The k nearest neighbors' (kNN) problem is to find the nearest k neighbors for a query point from a given data set. This problem occurs in many scientific and engineering applications including pattern recognition [1], object recognition [2], data clustering [3,4], function approximation [5], vector quantization [6,7], and pattern classification [8,9].

The intuitive method of finding the nearest k neighbors for a query point $\boldsymbol{Q}$ from a data set $S=\{\boldsymbol{X_1}, \boldsymbol{X_2}, \ldots, \boldsymbol{X_n}\}$ of n data points is to compute n distances between the query point and all data points in the data set. This method is known as the full search algorithm (FSA). In general, the squared Euclidean distance is used to measure the distance between two points, for the query point $\boldsymbol{Q}=[q_1, q_2, \ldots, q_d]^T$ with dimension d and a data point $\boldsymbol{X_i}=[x_{i1}, x_{i2}, \ldots, x_{id}]^T$ from data set S, the distance between these two points is defined as below:

$$D(X_i, Q) = \sum_{j=1}^{d}(x_{ij}-q_j)^2 \tag{1}$$

It is obviously to see that the finding process of k nearest neighbors for a query point using FSA is very time consuming. To reduce the computational complexity of the kNN finding process, many algorithms [10–26] were proposed. These algorithms can be categorized into two classes.

An algorithm in the first class creates a search tree for storing data points in a more efficient way and the pre-created tree structure is searched in the kNN finding process using a branch and bound search strategy. Fukunaga and Narendra [10] applied hierarchical clustering technique to decompose data points. The result of decomposition is represented by a tree (referred to as Ball tree) and a branch and bound search algorithm was applied to reduce the number of distance computations. In general, the Ball tree has good performance on high dimensional problems [11]. The performance of Ball tree is highly influenced by the clustering algorithm. To further improve the performance of Ball tree, five Ball tree construction algorithms were delivered by Omohundros [12]. Friedman et al. [13] presented a balanced *k-dimensional* tree (referred to as *k-d* tree) based on spatial decomposition and the refined version of the *k-d* tree method was given by Sproull [14]. The *k-d* tree performs better than the Ball tree when the dimension is small. Kim and Park [15] created a multiple branch search tree using the ordered partition method and a branch and bound search algorithm to find nearest neighbors. Mico et al. [16] proposed a different branch and bound search algorithm using a pre-stored distance table to reject more impossible nodes from a pre-created binary tree. McNames [17] introduced a fast kNN search method based on a principal axis search tree (PAT). Wang and Gan [18] integrated the PAT algorithm and projected clusters to reduce the query time for high-dimensional data sets. Chen et al. [19] used a lower-bound tree (LB tree) and winner update search method to speed up the nearest neighbors search problem.

The other class finds the nearest neighbors without using a tree structure. Cheng et al. [20] proposed the min–max method to decrease the number of multiplication operations. Bei and Gray [21] presented the partial distortion search (PDS) algorithm, which allows the early termination of a distance calculation between the query point and a data point. Ra and Kim [22]

* Corresponding author.
E-mail address: ycliaw@mail.nhu.edu.tw (Y.-C. Liaw).

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.01.003

utilized the equal-average nearest neighbor search algorithm to eliminate impossible data points using the difference between mean values of the query point and a data point. Tai et al. [23] applied the projection values of data points to eliminate unlike data points. In reference [24], Nene and Nayar proposed a projection value based search method to find nearest neighbors within a given distance around a query point. Lu et al. [25] used the mean value, variance, and norm of a data point to reject unlikely data points. Lai et al. [26] applied projection values and triangle inequality to speed up the *kNN* finding process (referred to as FKNNUPTI).

Among available methods, the PAT algorithm always has good performance for many types of benchmark data sets [17,18,26]. Using the PAT method, a principal axis search tree is off-line created according to projection values of data points onto principal axes of tree nodes. The principal axis of a node is evaluated using data points in the node and the principal component analysis (PCA) [17,28] method.

The *kNN* finding process for a query point using the PAT method is to delete impossible nodes from the tree using projection values of boundary points and a node elimination criterion. Once a node is determined to be deleted, data points belong to that node are impossible to be the *kNN* of the query point and distance calculations between the query point and those data points can be excluded. If a leaf node cannot be rejected, distances between the query point and all data points in this leaf node must be computed. In general, more than 90% distance calculations can be saved using the PAT method [26].

To find the *kNN* for a query point using the PAT method, boundary points and their projection values onto principal axes of nodes should be evaluated. For a large data set or a search tree with high depth, the computation time for evaluating such information may not be ignored. To reduce the computation time for evaluating such information and to eliminate more impossible data points, a novel method is presented in this paper. The proposed method creates an orthogonal search tree (OST) for a data set using an orthonormal basis evaluated from the data set and a similar search tree construction process as that presented in the PAT method. In the proposed method, an orthogonal vector is chosen from the orthonormal basis for a node instead of using the principal axis of a node to divide data points of a node into several groups. For a node in an OST, the orthogonal vector selected by the node is perpendicular to those orthogonal vectors chosen by ancestors of the node. Since orthogonal vectors chosen by nodes in a path of an OST are mutually orthogonal and the number of orthogonal vectors is small, our proposed method requires no boundary points and only little computation time on evaluating projection values in the *kNN* finding process. To further reduce the computation load, a point elimination inequality is also presented in our method to reject unlikely data points, which cannot be deleted using the node elimination inequality. Through applying the orthonormal basis and two effective inequalities in our method, computation time can be effectively reduced by comparing with five *kNN* search methods [10,13,17,19,26] through using data sets from different kinds of data distributions and real data.

The rest of this paper is organized as follows. In Section 2, the PAT algorithm is briefly reviewed. Our proposed method is presented and described in detail in Section 3. Experimental results and conclusions are given in Sections 4 and 5, respectively.

2. The principal axis search tree algorithm

The PAT algorithm [17] includes two processes that are the principal axis search tree construction process and the *k* nearest neighbors searching process. The principal axis search tree construction process is to partition a data set into distinct subsets using the PCA method and a tree structure. The *kNN* searching process is to find the *k* nearest neighbors for a query point from the constructed PAT. These two processes are introduced in the following sections.

2.1. Principal axis search tree construction process

The PCA technique is a method to obtain the principal axis from a set of data points. The projection value of a data point onto the principal axis is the major feature that distinguishes the data point from others. In general, data points with closer projection values will have smaller distances. The PCA technique is applied here to obtain the principal axis of data points and projection values of data points onto the principal axis are used to partition data points into groups.

In the beginning of the PAT construction process, there is only one node (the root node) presented in the PAT and the node contains all available data points. Let n_c be the number of child nodes in a node. The next step is to partition data points in the node into n_c child nodes.

The partition process of a node is started from checking the number of data points in the node. Let n_p be the number of data points in a node. If $n_p < n_c$, data points in this node should not be divided and this node is a terminal node (leaf node). Otherwise, this node is an internal node and data points in the node must be divided into n_c groups. In case of data points in a node to be divided, the principal axis of this node should be evaluated using data points in the node and recorded for the node. Projection values of data points in the node onto the principal axis of the node are then evaluated and used for arranging data points into an increasing sequence of projection values. After that, n_c child nodes are created and each contains about n_p/n_c data points with a continuous range of projection values. Besides, child nodes of a node should be sorted in ascending order of their projection values and the minimum and maximum projection values for data points in each child node should be stored in the node.

The above partition process is applied to the root node and recursively applied to its child nodes till every node in the tree all have been partitioned or should not be partitioned. After the PAT creation process, a list of data points will be recorded for a leaf node. For an internal node, a principal axis, a list of child nodes, and the minimum and maximum projection values for every child node must be stored.

2.2. k nearest neighbors search using the principal axis search tree

Given a query point $\boldsymbol{Q}$, our task is to find the *k* nearest neighbors of $\boldsymbol{Q}$ from the PAT. The searching process is started from finding a leaf node, which has the most similar projection values to those of $\boldsymbol{Q}$. To find the leaf node, we first evaluate the projection value of $\boldsymbol{Q}$ onto the principal axis of the root node. Since child nodes of a node are sorted by projection values, the child node with the range of projection values containing $\boldsymbol{Q}$'s projection value can be found in $O(\log_2 n_c)$ using the binary search algorithm. The similar process is recursively applied to the child node, which has the range of projection values containing $\boldsymbol{Q}$'s projection value, till a leaf node is reached. Once the leaf node is found, the partial distortion search (PDS) [21] method is applied to find the *kNN* for $\boldsymbol{Q}$ from the leaf node.

After the leaf node with closest projection values to those of $\boldsymbol{Q}$ is searched, the searching process turns to check ancestors of the leaf node from the parent node of the leaf node up to the root node. Let $\boldsymbol{N_a}$ be an ancestor of the leaf node. Every child node of

$\boldsymbol{N_a}$ except the one with the range of projection values containing $\boldsymbol{Q}$'s projection value should be checked to see if it can be rejected in the *kNN* searching process of $\boldsymbol{Q}$. If a child node cannot be rejected in the checking process, its child nodes must be recursively checked.

Before introducing the node checking process, the lower bound and the boundary point should be defined for a node first. Since a node has explicit ranges of projection values onto principal axes of its ancestors, a node will be bounded by a hypercube and the coordinate axes of the hypercube are principal axes of the node's ancestors. Consequently, the lower bound of a node can be defined as the distance from the query point $\boldsymbol{Q}$ to the boundary of the hypercube and the boundary point is a point on the hypercube's boundary with the least distance to $\boldsymbol{Q}$. In the beginning of the checking process for $\boldsymbol{N_a}$, the lower bound and boundary point of $\boldsymbol{N_a}$ are set as 0 and $\boldsymbol{Q}$, respectively. The checking process for the node $\boldsymbol{N_a}$ and its descendants is described in the following.

Let $\boldsymbol{N}$ be a node whose child nodes are going to be checked, d_{LB}, $\boldsymbol{B}$, and $\boldsymbol{P}$ be the lower bound, the boundary point, and the principal axis of $\boldsymbol{N}$, respectively, b be the projection value of $\boldsymbol{B}$ onto $\boldsymbol{P}$, and p^m_{min} and p^m_{max} individually represent the minimum and maximum projection values for data points in mth child node of $\boldsymbol{N}$. The checking sequence of child nodes in $\boldsymbol{N}$ is in ascending order of the differences between their projection values and b and the lower bound of a child node is used to check whether a child node could be deleted. The evaluation of lower bound for a child node depends on its projection value. For a child node m with p^m_{max} less than b, the lower bound of the child node is evaluated using the following equation:

$$d^m_{LB} = d_{LB} + (b - p^m_{max})^2 \tag{2}$$

For a child node m having p^m_{min} greater than b, the lower bound of the node is defined as

$$d^m_{LB} = d_{LB} + (p^m_{min} - b)^2 \tag{3}$$

Once the lower bound of a child node is determined, the node elimination criterion (inequality (4)) could be used to check if the child node can be eliminated or not:

$$d_k \le d^m_{LB} \tag{4}$$

where d_k is the distance between $\boldsymbol{Q}$ and the candidate of its kth nearest neighbor. If inequality (4) is satisfied for a child node m whose p^m_{max} is less than b, the child node and those child nodes with projection values less than p^m_{max} can be eliminated at a time. In case of inequality (4) is satisfied for a child node whose p^m_{min} is greater than b, the child node and those child nodes with projection values greater than p^m_{min} can be rejected.

For a child node cannot be rejected using the node elimination criterion and the node is a leaf node, distances between $\boldsymbol{Q}$ and all data points in the node must be computed with PDS applied. If a child node m cannot be rejected and the node is an internal node, its child nodes must be further checked. In such a case, the boundary point $\boldsymbol{B^m}$ of child node m is evaluated using Eqs. (5) or (6), where Eq (5) is used for a child node with p^m_{max} less than b and Eq. (6) is applied when a child node with p^m_{min} greater than b:

$$B^m = B - (b - p^m_{max})P \tag{5}$$

$$B^m = B + (p^m_{min} - b)P \tag{6}$$

It is noted that one boundary point and one projection value is required in the checking process of a node. In the worst case, the number of boundary points and projection values to be evaluated for a query point is close to the number of internal nodes in the PAT. That is, for a big PAT, the computation time to evaluate boundary points and projection values will be large.

3. The orthogonal search tree algorithm

To reduce the computation time on evaluating boundary points and projection values in the *kNN* searching process for a query point, we use the orthonormal basis of a data set to build the search tree. In this paper, the orthonormal basis of a data set is evaluated using the PCA [28] and consists of d orthogonal vectors, where d is the dimension of the data set.

Let $O=\{V_1, V_2, \ldots, V_d\}$ be the orthonormal basis of a data set and V_i denote the ith orthogonal vector in O, $\boldsymbol{X}=[x_1, x_2, \ldots, x_d]^T$ and $\boldsymbol{Y}=[y_1, y_2, \ldots, y_d]^T$ be two points, $\{p_1, p_2, \ldots, p_d\}$ and $\{q_1, q_2, \ldots, q_d\}$ be projection values of $\boldsymbol{X}$ and $\boldsymbol{Y}$ onto orthogonal vectors $\{V_1, V_2, \ldots, V_d\}$, respectively. Since O is the orthonormal basis of the data set, the distance between $\boldsymbol{X}$ and $\boldsymbol{Y}$ can be computed as follows:

$$D(X,Y) = \sum_{i=1}^{d}(x_i - y_i)^2 = \sum_{i=1}^{d}(p_i - q_i)^2 = \sum_{i=1}^{j}(p_i - q_i)^2 + \sum_{i=j+1}^{d}(p_i - q_i)^2 \tag{7}$$

where $1 \le j < d$. From Eq. (7), we can have the following inequality:

$$D(X,Y) \ge \sum_{i=1}^{j}(p_i - q_i)^2 \tag{8}$$

According to the triangle inequality, we have that

$$\sqrt{\sum_{i=j+1}^{d}(p_i - q_i)^2} \ge \sqrt{\sum_{i=j+1}^{d} p_i^2} - \sqrt{\sum_{i=j+1}^{d} q_i^2} \tag{9}$$

Using inequality (7) and inequality (9), we can easily derive the following inequality:

$$D(X,Y) \ge \sum_{i=1}^{j}(p_i - q_i)^2 + \left(\sqrt{\sum_{i=j+1}^{d} p_i^2} - \sqrt{\sum_{i=j+1}^{d} q_i^2}\right)^2 \tag{10}$$

Here, inequalities (8) and (10) will be applied for pruning impossible nodes and data points, respectively, in our proposed fast *kNN* search algorithm. Now, we would like to introduce the orthogonal search tree (OST) construction process and the *kNN* search method using the OST.

3.1. Orthogonal search tree construction process

In the beginning of the OST building process, the root node of the OST is first created and contains all data points from the data set. Besides, a residual length should be defined for each data point and is used in checking if a data point in a leaf node is impossible to be the *kNN* of the query point and can be rejected in the *kNN* finding process. The initial value of the residual length for a data point is the length (norm) of the data point. For a data point $\boldsymbol{X_i}=[x_{i1}, x_{i2}, \ldots, x_{id}]^T$, the length of $\boldsymbol{X_i}$ is equal to $\sqrt{x_{i1}^2 + x_{i2}^2 + \cdots + x_{id}^2}$. After the root node is created, data points in the root node should be partitioned into n_c child nodes. The partition process of a node is described in the following.

Let N_c be a node to be partitioned, n_p be the number of data points in N_c, and l be the level of N_c (the level of the root node is 1). If $n_p < n_c$ or $l > d$, data points in N_c should not be divided. Otherwise, data points must be partitioned into n_c groups. To partition data points in N_c into groups, an orthogonal vector is selected from the orthonormal basis for N_c. Let O_a be the set of orthogonal vectors chosen by ancestors of N_c, and O^c_a be the complement of O_a in O

$$V_c = \arg\max_{V_i \in O_a^c}\{\mathrm{Var}(\text{projection values of data points in } N_c \text{ onto } V_i)\} \tag{11}$$

where Var(.) is the variance operation.

Once the orthogonal vector of N_c is determined, the orthogonal vector of N_c is recorded and projection values of data points in N_c onto V_c are evaluated. After that, a similar partition method as that appears in the PAT building process is utilized to divide data points into n_c subsets. Before putting data points into child nodes, residual lengths of data points in N_c must be updated. Let r_i be the residual length of a data point $\boldsymbol{X_i}$ in N_c and p_i be the projection value of $\boldsymbol{X_i}$ onto the orthogonal vector of N_c. The new residual length of $\boldsymbol{X_i}$ is $\sqrt{r_i^2-p_i^2}$. After the partition process, each child node contains about n_p/n_c data points with a continuous range of projection values, child nodes of N_c are sorted in ascending order according to their projection values, and the minimum and maximum projection values for each child node are stored in N_c.

The above partition process is applied to the root node and repeated applied to its child nodes till all nodes are partitioned or should not be partitioned. After the OST is built for a data set, data points in the data set and their corresponding residual lengths are stored in leaf nodes. For an interior node, a number to identify its orthogonal vector, a list of child nodes, and the minimum and maximum projection values for every child node are recorded. The algorithm for building an OST for a data set is given below.

3.1.1. *The OST construction algorithm*

(1) Determine the orthonormal basis for the data set and set the residual length of each data point as its length (norm).
(2) Create the root node for the OST, assign all data points to the root node, and push the root node to an initially empty stack ***SN***.
(3) Pop one node from stack ***SN*** and set that node as the current node.
(4) If the number of data points in current node is less than n_c, or the level of current node greater than d, current node is a leaf node and go to step (5). Otherwise, current node is an internal node and following steps must be executed.
  (4.1) Determine the orthogonal vector of current node, evaluate projection values of data points in current node onto the selected orthogonal vector, arrange data points in ascending order according to their projection values, and update the residual length for each data point.
  (4.2) Partition data points in current node into n_c distinct subsets according to their projection values and create n_c child nodes each contains one subset of data points. Record the orthogonal vector of current node, n_c child nodes, and the minimum and maximum projection values for each child node.
  (4.3) Push every child node of current node into stack ***SN***.
(5) If stack ***SN*** is not empty, go to step (3). Otherwise terminate the OST building process.

3.2. *k nearest neighbors search using the orthogonal search tree*

To find the nearest k neighbors for a query point $\boldsymbol{Q}$ from the OST, a leaf node having the most similar projection values to those of $\boldsymbol{Q}$ must be found first. The process for finding such a leaf node is similar to that of the PAT method. Once the leaf node is found, the PDS is applied to search the *kNN* of $\boldsymbol{Q}$ from the leaf node cooperating with a point elimination inequality, which will be introduced later in this section.

After that, the searching process turns to check ancestors of the leaf node from the parent node of the leaf node up to the root node. The checking order of nodes in the OST method is also similar to that of the PAT method, but the checking process for rejecting nodes from the OST is simpler because no boundary points are required. Let $\boldsymbol{N_a}$ be a node whose child nodes are going to be checked, the lower bound of $\boldsymbol{N_a}$ be 0, and the initial residual length of the query point $\boldsymbol{Q}=[q_1, q_2, \ldots, q_d]^T$ is $r_q=\sqrt{q_1^2+q_2^2+\cdots+q_d^2}$. The checking process for the node $\boldsymbol{N_a}$ and its descendants is described in the following.

Let $\boldsymbol{N}$ be a node, whose child nodes are going to be checked, d_{LB} and $\boldsymbol{P}$ be the lower bound and the orthogonal vector of $\boldsymbol{N}$, respectively, q be the projection value of $\boldsymbol{Q}$ onto $\boldsymbol{P}$, and p_{min}^m and p_{max}^m individually be the minimum and maximum projection values for data points in mth child node of $\boldsymbol{N}$. The checking order of child nodes in $\boldsymbol{N}$ is in ascending order according to the differences between projection values of child nodes and q and the lower bound of a child node is used to check whether a child node could be deleted. The lower bound of a child node depends on its projection value. For a child node m with p_{max}^m less than q, the lower bound of the child node is evaluated using the following equation:

$$d_{LB}^m = d_{LB}+(q-p_{max}^m)^2 \tag{12}$$

For a child node m having p_{min}^m greater than q, the lower bound of the node is defined as below:

$$d_{LB}^m = d_{LB}+(p_{min}^m-q)^2 \tag{13}$$

Once the lower bound of a child node is determined, the node elimination inequality as defined in Eq. (4) could be used to check if the child node can be eliminated or not. If the child node cannot be rejected using inequality (4), the updated residual length $r_q=\sqrt{r_q^2-q^2}$ of $\boldsymbol{Q}$ is delivered to this child node and the node checking process is recursively applied to this child node. For a child node cannot be rejected and it is a leaf node, data points in this node could be further checked using the following point elimination inequality:

$$d_k \le d_{LB}+(r_i-r_q)^2 \tag{14}$$

where r_i is the residual length of a data point X_i in the leaf node.

If a data point cannot be rejected using the point elimination inequality, the distance between the data point and $\boldsymbol{Q}$ is computed with PDS applied. It is noted that only projection values of $\boldsymbol{Q}$ onto orthogonal vectors should be evaluated in the *kNN* searching process in our method. Since there are merely d orthogonal vectors could be used for an OST, at most, only d projection values will be computed for a query point. Now, we would like to present our proposed *kNN* search algorithm. In convenience, the *kNN* search algorithm is presented in the form of a function. The prototype of the *kNN* search function is

searchOST(N, Q, dLB, rq)

where the four parameters in function *searchOST* represent the node to be searched, the query point, the lower bound of N, and the residual length of Q, respectively. For a query point Q and an OST with a root node N_r, the *kNN* finding process for the query point Q is initiated by calling *searchOST*$(N_r, Q, 0, l_q)$, where l_q is the length of Q. The pseudo-codes for the *searchOST* function are listed as follows:

```
Function searchOST($N$, $Q$, $d_{LB}$, $r_q$)
(1)  If N is a leaf node
(2)      For each $X_i$ in node $N$
```

```
(3)           Evaluate $d=d_{LB}+(r_i-r_q)^2$
(4)           If ($d_k > d$)
                  Compute $D(Q, X_i)$ with PDS applied
                  Update $d_k$ and the $kNN$ set.
(5)   Else
(6)      If projection value $q$ of $Q$ onto orthogonal vector of $N$ is
         not available
(7)           Evaluate the value of $q$
(8)      Set $r_q=\sqrt{r_q^2-q^2}$
(9)      Let $N_{middle}$ be a child node of $N$, which has the closest
         projection value to $q$ and $middle$ be the sequence
         number of $N_{middle}$.
(10)     searchOST($N_{middle}$, $Q$, $d_{LB}$, $r_q$)
(11)     Let $left=middle-1$ and $right=middle+1$.
(12)     If ($left < 1$) $d_{left}=\infty$
         Else $d_{left}=q-p_{max}^{left}$
(13)     If ($right > n_c$) $d_{right}=\infty$
         Else $d_{right}=p_{min}^{right}-q$
(14)     If ($d_{left} <= d_{right}$)
              $d=d_{left}^2+d_{LB}$
              If ($d_k \le d$) go to step (15)
              searchOST($N_{left}$, $Q$, $d$, $r_q$);
              Left=left−1
              If ($left < 1$) $d_{left}=\infty$
              Else $d_{left}=q-p_{max}^{left}$
         Else
              $d=d_{right}^2+d_{LB}$
              If ($d_k \le d$) go to step (15)
              searchOST($N_{right}$, $Q$, $d$, $r_q$);
              right=right+1
              If ($right > n_c$) $d_{right}=\infty$
              Else $d_{right}=p_{min}^{right}-q$
(15)     Terminate the $kNN$ searching process and return the
         $kNN$ set.
```

4. Experimental results

To evaluate the performance of the proposed algorithm, the uniform Markov source [26,29], the auto-correlated data [19], the clustered Gaussian data [19,27], one set of real images [26], and the Statlog data set from reference [30] are used. The proposed algorithm OST is compared with the full search algorithm (FSA), Ball tree [10], *k-d* tree [13], PAT algorithm [17], LB tree [19], and FKNNUPTI method [26] in terms of the average number of distance calculations per data point and total computation time. Except the FSA and LB tree, the PDS method [21] is applied in the process of distance computation. Since the winner update strategy is used in LB tree, the number of distance calculation is not available for LB tree.

In the experiment, search trees were constructed with n_c=16 for Ball tree, PAT, and OST, where n_c is the number of child nodes. For *k-d* tree, each bucket contains 40 points. For LB tree, the LBT+HAAR method [19] is used. All computing was performed on a personal computer with Intel Core 2 Duo E8500 3.16 GHz CPU and memory of 2 GB. All programs were implemented as console applications of Microsoft Visual C++ 2008 Express and executed under Windows XP Professional SP3. The preprocessing time for constructing the search tree, evaluating the principal axes of nodes, computing the orthonormal basis of a data set, and computing the projection values of data points is not included in the computation time.

Example 1. Data sets are from the uniform Markov source.

In this example, several data sets with dimensions ranging from 4 to 256 are first generated from the uniform Markov sequence of values in the range of −1 to 1. Let α be the correlation coefficient of the uniform Markov sequence. The uniform Markov sequence $\{y_i\}$ is generated using the following equation:

$$y_{i+1}=\alpha y_i+u \tag{15}$$

where u is a random number from uniform distribution and y_0=0.

In this example, the value of α is set as 0.9 for every data sets and each data set contains 10,000 data points. The total computation time and number of distance calculations per data point are calculated using 10,000 query points. Query points are also obtained from the uniform Markov sequence.

Tables 1 and 2 list the number of distance calculations per data point and the total computation time for using *kNN* search algorithms to find 3 nearest neighbors of 10,000 query points from data sets with dimensions ranging from 4 to 256, respectively. From these two tables, we may find that our method outperforms other methods in case of the number of dimensions less than 128 in terms of total computation time. When the number of dimensions is greater than 64, no algorithms can get significant improvement on FSA. In such a case, the PDS method [21] may be the best choice.

Fig. 1 gives the results for six methods, each method was executed on a data set of 10,000 data points with various k and dimension=16. The result of this experiment is influenced by the updating scheme of the *kNN* set. In our experiments, the *kNN* of a query point is stored in an array for all methods except the LB tree. To update the *kNN* of a query point during the *kNN* searching process, a point in *kNN* with maximum distance is searched in linear time. From Fig. 1, we can see that all methods have similar growing rate with increasing k.

To see the influence of the number of data points in a data set on the performance of *kNN* search algorithms, data sets with various numbers of data points and with dimension=16 are used and the result is given in Fig. 2. From Fig. 2, we can see that the proposed method has the best performance for all cases. It is

Table 1
Average number of distance calculations per data point to find 3 nearest neighbors for query points from data sets with dimensions ranging from 4 to 256.

Algorithm	4	8	16	32	64	128	256
FSA	10,000	10,000	10,000	10,000	10,000	10,000	10,000
Ball tree	193	584	1435	2773	5424	9529	11,996
k-d tree	175	863	3178	7776	9991	10,000	10,000
PAT	55	242	587	1178	2685	6030	9382
FKNNUPTI	17	128	409	958	2349	5625	9175
OST	32	202	521	1056	2458	5854	9412

Table 2
Total computation time (in ms) to find 3 nearest neighbors for query points from data sets with dimensions ranging from 4 to 256.

Algorithm	4	8	16	32	64	128	256
FSA	2453	4531	8703	16,969	34,328	67,390	135,250
Ball tree	197	662	1897	5391	16,956	53,056	132,197
k-d tree	63	362	1719	6378	15,678	36,022	87,622
PAT	53	200	581	1700	6222	25,806	88,062
LB tree	437	563	1204	2672	5578	18,031	51,703
FKNNUPTI	81	272	697	1722	5497	22,103	76,519
OST	38	153	472	1366	5072	23,553	82,356

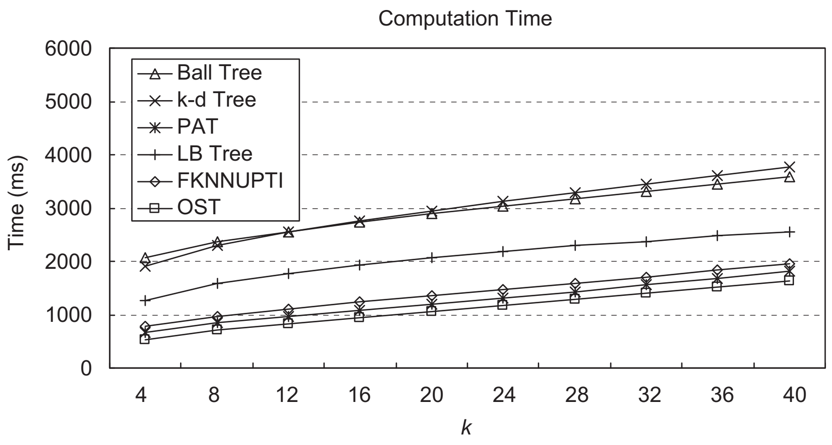

Fig. 1. Total computation time (in ms) to find k nearest neighbors for 10,000 query points from data sets of 10,000 data points and with dimension=16.

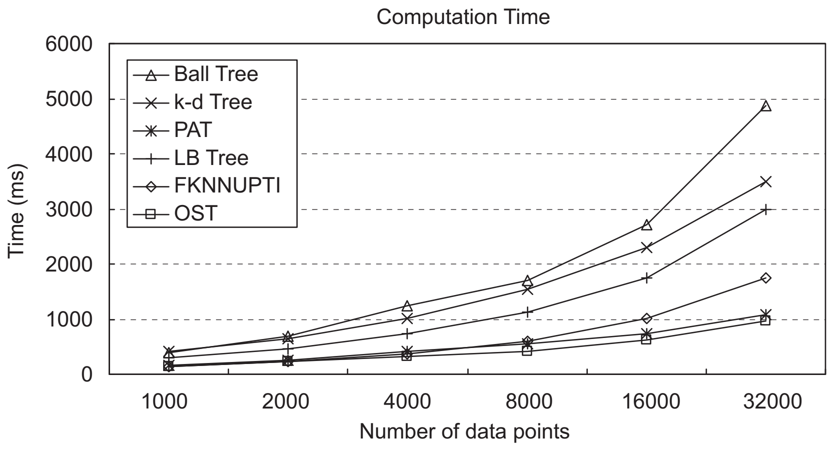

Fig. 2. Total computation time (in ms) to find 3 nearest neighbors for 10,000 query points from data sets of various numbers of data points and with dimension=16.

noted that our proposed method has less growing rate with increasing the number of data points than other methods.

Example 2. Data sets are from the auto-correlated data.

Several auto-correlated data sets with dimensions ranging from 4 to 256 are generated and used in this example. The generation method of an auto-correlated data point is started from obtaining the value of its first dimension from a uniform distribution of values in $[1,-1]$. The values of successive dimensions of the data point are calculated by adding the value of its first dimension to a value from a zero mean normal distribution data source with the standard deviation of 0.1. In this example, each data set contains 10,000 data points. The total computation time and average number of distance calculations per data point are computed using 10,000 query points, which are generated using the same method.

Tables 3 and 4 give the total computation time and average number of distance calculations per data point for using *kNN* search algorithms to find 3 nearest neighbors for 10,000 query points from data sets of various dimensions, respectively. From Table 4, we can see that the OST method takes the least computation time than others when the number of dimensions less than 128. In case of the number of dimensions is greater than

Table 3
Average number of distance calculations per data point to find 3 nearest neighbors for query points from auto-correlated data with various dimensions.

Dimension	4	8	16	32	64	128	256
FSA	10,000	10,000	10,000	10,000	10,000	10,000	10,000
Ball tree	200	417	631	957	1241	1687	2299
k-d tree	172	624	1413	3152	5839	8897	9990
PAT	45	210	296	436	575	791	1173
FKNNUPTI	15	81	149	271	402	611	970
OST	41	175	262	396	522	719	1067

Table 4
The total computation time (in ms) to find 3 nearest neighbors for query points from auto-correlated data with various dimensions.

Dimension	4	8	16	32	64	128	256
FSA	2453	4531	8703	16,969	34,328	67,390	135,250
Ball tree	209	463	878	1966	4169	10,337	25,963
k-d tree	63	272	778	2534	7191	18,841	46,225
PAT	56	166	297	666	1463	3616	10,365
LB tree	297	422	813	1235	1562	2016	2875
FKNNUPTI	50	144	259	562	1184	2834	7887
OST	44	137	244	553	1191	2988	8384

64, our proposed method may still have good performance on this kind of data sets.

Example 3. data sets are from the clustered Gaussian data.

In this example, data sets are obtained from the clustered Gaussian data source with dimension=32. To generate a clustered Gaussian data set, cluster centers are first chosen from a uniform distribution of values in $[-1,1]$. After the cluster centers are determined, data points for a cluster are obtained from a Gaussian distribution centered at the cluster center. In this example, 100 cluster centers are generated for each data set and each cluster contains 100 data points. That is, each data set consists of 10,000 data points. The total computation time and average number of distance calculations per data point are evaluated using 10,000 query points. Each query point is generated using the same method as to generate the data set.

Tables 5 and 6 give the total computation time and average number of distance calculations per data point for using seven algorithms to find 3 nearest neighbors of 10,000 query points from data sets of various standard deviations, respectively. From Tables 5 and 6, we may find that the LB Tree and FKNNUPTI methods are highly influenced by the standard deviation of a data set. Our proposed method has less sensitivity to standard deviation and takes least computation time then other methods.

Example 4. Data sets are codebook generated using 6 real images.

In this example, data sets consist of 2048–16,384 data points with dimension 16 are generated from six gray images ("Lena," "Peppers," "Baboon," "Airplane," "Island," and "Parrot") using GLA [31]. The image size of these images is 512×512. Every data point is a non-overlapping 4×4 pixel block obtained from these images and each component of a data point has a value in the range of from 0 to 255. All query points are selected randomly from six training images.

Tables 7 and 8 give the number of distance calculations per data point and total computation time to find the nearest neighbors for 10,000 query points. From Table 8, we can see that the FKNNUPTI and the OST methods always spend less computation time than the other methods. Comparing with the FKNNUPTI method, the OST method requires less computation time when the number of data points is large enough.

Example 5. Using the Statlog data set.

The final example uses the Statlog data set [30], which consists of spectral values from a satellite image, to estimate the performance of *kNN* search algorithms. The Statlog data set includes 6435 data points of dimension 36. The value for each component of a data point is in the range of 0–255. The average query time and number of distance calculations per data points are calculated using 10,000 query points. Each query point is obtained using the mean value of four data points selected randomly from the Statlog data set.

The total computation time and number of distance calculations per data point to find three nearest neighbors for seven algorithms are listed in Table 9. From this table, we can see that the OST method has the best performance than the other methods both in terms of distance calculations and computation time. Compared with the second best method, our proposed method can effectively reduce the total computation time and number of distance calculations by about 34% and 39%, respectively.

Table 5
Average number of distance calculations per data point to find 3 nearest neighbors for query points from data sets with various standard deviations.

Standard deviation	0.02	0.04	0.06	0.08	0.10
FSA	10,000	10,000	10,000	10,000	10,000
Ball tree	266	278	300	323	351
k-d tree	185	251	379	607	927
PAT	111	129	157	208	290
FKNNUPTI	117	158	217	444	735
OST	104	119	157	220	340

Table 6
Total computation time (in ms) to find 3 nearest neighbors for query points from data sets with various standard deviations.

Standard deviation	0.02	0.04	0.06	0.08	0.10
FSA	16,969	16,969	16,969	16,969	16,969
Ball tree	616	650	728	766	850
k-d tree	219	247	300	394	525
PAT	228	262	322	397	503
LB tree	281	469	735	1219	1968
FKNNUPTI	362	587	828	1141	1412
OST	203	225	266	312	394

Table 7
Average number of distance calculations to find the nearest neighbor from data sets with various sizes.

Codebook size	2048	4096	8192	16384
FSA	2048	4096	8192	16,384
Ball tree	297	523	957	1595
k-d tree	343	489	680	871
PAT	103	79	136	215
FKNNUPTI	20	30	44	56
OST	57	55	89	141

Table 8
Total computation time (in ms) to find the nearest neighbor from data sets with various sizes.

Codebook size	2048	4096	8192	16,384
FSA	1782	3562	7109	14,219
Ball tree	466	847	1597	2691
k-d tree	169	238	331	419
PAT	69	128	144	172
LB tree	156	234	422	719
FKNNUPTI	44	69	112	159
OST	53	84	103	134

Table 9
Total computation time (in ms) and average number of distance calculations to find three nearest neighbors from the Statlog data set.

	Total computation time	Number of distance calculations
FSA	12,250	6435
Ball tree	2475	1030
k-d tree	2800	3357
PAT	669	354
LB tree	4594	–
FKNNUPTI	887	484
OST	441	216

5. Conclusions

In this paper, we have presented a novel fast exact *kNN* search algorithm using the orthogonal search tree to speed up the *kNN* searching process. The proposed method creates an orthogonal search tree for a data set using an orthonormal basis evaluated from the data set. To find the *kNN* for a query point from an orthogonal search tree, projection values of the query point onto orthogonal vectors and a node elimination inequality are applied for pruning unlikely nodes. For a node, which cannot be deleted, a point elimination inequality is used to reject impossible data points. Experimental results show that the proposed method always spends less computation time to find the *kNN* for a query point than the other methods, especially for a data set with a big number of data points or a large standard deviation. It is noted that the proposed method will find the same results as those of the full search algorithm.

Acknowledgement

This work was supported by a Grant from National Science Council of Taiwan, ROC under Grant no. NSC-98-2221-E-343-008.

References

[1] S. Theodoridis, K. Koutroumbas, Pattern Recognition, 2nd edition, Academic Press, 2003.
[2] H. Murase, S.K. Nayar, Visual learning and recognition of 3D objects from appearance, International Journal of Computer Vision 14 (1) (1995) 5–24.
[3] V. Roth, J. Laub, M. Kawanabe, J.M. Buhmann, Optimal cluster preserving embedding of nonmetric proximity data, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (12) (2003) 1540–1551.
[4] Y.C. Liaw, Improvement of the fast exact pairwise-nearest-neighbor algorithm, Pattern Recognition 42 (5) (2009) 867–870.
[5] C.Y. Chen, C.C. Chang, R.C.T. Lee, A near pattern-matching scheme based on principal component analysis, Pattern Recognition Letters 16 (1995) 339–345.
[6] Y.C. Liaw, J.Z.C. Lai, W. Lo, Image restoration of compressed image using classified vector quantization, Pattern Recognition 35 (2) (2002) 329–340.
[7] A. Gersho, R.M. Gray, Vector Quantization and Signal Compression, Kluwer Academic Publishers, Boston, MA, 1991.
[8] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory 13 (1) (1967) 21–27.
[9] N. García-Pedrajas, D. Ortiz-Boyer, Boosting k-nearest neighbor classifier by means of input space projection, Expert Systems with Applications 36 (7) (2009) 10570–10582.
[10] K. Fukunaga, P.M. Narendra, A branch and bound algorithm for computing k-nearest neighbors, IEEE Transactions on Computers C-24 (7) (1975) 750–753.
[11] T. Liu, A.W. Moore, A. Gray, New algorithms for efficient high-dimensional nonparametric classification, Journal of Machine Learning Research 7 (2006) 1135–1158.
[12] S.M. Omohundro, Five balltree construction algorithms, Technical Report 89-063, (1989).
[13] J.H. Friedman, J.L. Bentley, R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Transactions on Mathematical Software 3 (3) (1977) 209–226.
[14] R.F. Sproull, Refinements to nearest-neighbor searching in k-dimensional trees, Algorithmica 6 (1991) 579–589.
[15] B.S. Kim, S.B. Park, A fast k nearest neighboring finding algorithm based on the ordered partition, IEEE Transactions on Pattern Analysis and Machine Intelligence 8 (6) (1986) 761–766.
[16] L. Mico, J. Oncina, R.C. Carrasco, A fast branch and bound nearest neighbor classifier in metric spaces, Pattern Recognition Letters 17 (7) (1996) 731–739.
[17] J. McNames, A fast nearest-neighbor algorithm based on a principal axis search tree, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (9) (2001) 964–976.
[18] B. Wang, J.Q. Gan, Integration of projected clusters and principal axis trees for high-dimensional data indexing and query, Lecture Notes in Computer Science 3177 (2004) 191–196.
[19] Y.-S. Chen, Y.-P. Hung, T.-F. Ten, C.-S. Fuh, Fast and versatile algorithm for nearest neighbor search based on a lower bound tree, Pattern Recognition 40 (2) (2007) 360–375.
[20] D.-Y. Cheng, A. Gersho, B. Ramamurthi, Y. Shoham, Fast search algorithms for vector quantization and pattern matching, IEEE International Conference on Acoustics, Speech, and Signal Processing 1 (1984) 9.11.1–9.11.4.
[21] C.D. Bei, R.M. Gray, An improvement of the minimum distortion encoding algorithm for vector quantization, IEEE Transactions on Communications 33 (10) (1985) 1132–1133.
[22] S.W. Ra, J.K. Kim, Fast mean-distance-ordered partial codebook search algorithm for image vector quantization, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 40 (9) (1993) 576–579.
[23] S.C. Tai, C.C. Lai, Y.C. Lin, Two fast nearest neighbor searching algorithms for image vector quantization, IEEE Transactions on Communications 44 (12) (1996) 1623–1628.
[24] S.A. Nene, S.K. Nayar, A simple algorithm for nearest neighbor search in high dimensions, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (9) (1997) 989–1003.
[25] Z.-M. Lu, S.-C. Chu, K.-C. Huang, Equal-average equal-variance equal-norm nearest neighbor codeword search algorithm based on ordered Hadamard transform, International Journal of Innovative Computing, Information and Control 1 (1) (2005) 35–41.
[26] J.Z.C. Lai, Y.C. Liaw, J. Liu, Fast k-nearest-neighbor search based on projection and triangular inequality, Pattern Recognition 40 (1) (2007) 351–359.
[27] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, A. Wu, An efficient k-means clustering algorithm: analysis and implementation, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (7) (2002) 881–892.
[28] I.T. Jolliffe, Principal component analysis, Springer, NY, 2002.
[29] W.H. Cooley, P.R. Lohnes, Multivariate data analysis, Wiley, New York, 1971.
[30] Statlog (Landsat Satellite) data set, 〈http://archive.ics.uci.edu/ml〉.
[31] Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design, IEEE Transactions on Communications 28 (1) (1980) 84–95.

About the Author – YI-CHING LIAW was born in Taiwan, in 1970. He received his B.S., M.S., and Ph.D. degrees in Information Engineering and Computer Science all from Feng-Chia University, Taiwan, in 1992, 1994, and 2004, respectively. From 1999 to 2004, he was an engineer with Industrial Technology Research Institute, Hsinchu, Taiwan. In 2005, he joined the Department of Computer Science and Information Engineering, Nanhua University, Chiayi, Taiwan as an assistant professor. Since 2008, he has been an associate professor at the same department. His current research interests are in data clustering, fast algorithm, image processing, video processing, and multimedia system.

About the Author – CHIEN-MIN WU was born in Taiwan, ROC, in 1966. He received the B.S. degree in automatic control engineering from the Feng-Jea University, Taichung, Taiwan, in 1989, the M.S. degree in electrical and information engineering from Yu-Zu University, Chung-Li, Taiwan, in 1994, and the Ph.D. degree in electrical engineering from National Chung Cheng University, Chia-Yi, Taiwan, in 2004. In July 1994, he joined the Technical Development Department, Philips Ltd. Co., where he was a Member of the Technical Staff. Currently, he is also a faculty member of the Department of Computer Science and Information Engineering, NanHua University, Dalin, Chia-Yi, Taiwan, ROC. His current research interests include ad hoc wireless network protocol design, IEEE 802.11 MAC protocols, and fast algorithm.

About the Author – MAW-LIN LEOU was born in Taiwan, in 1964. He received the B.S. degree in communication engineering from National Chiao-Tung University, Taiwan in 1986, the M.S. degree in electrical engineering from the National Taiwan University, Taiwan in 1988, and the Ph.D. degree in electrical engineering from the National Taiwan University in 1999. From 1990 to 1992, he was with the Telecommunication Labs, Ministry of Transportation and Communications in Taiwan, as an Assistant Researcher. From 1992 to 1999, he has been an instructor in the Department of Electronic Engineering, China Institute of Technology, Taiwan. From 1999 to 2005, he has been an associate professor in the Department of Electronic Engineering, Nan-Jeon Institute of Technology, Taiwan. Since 2005, he is a faculty member in the Department of Computer Science and Information Engineering, NanHua University, Taiwan. His current research interests include adaptive arrays, adaptive signal processing, wireless communication, and fast algorithm.