Digital Signal Processing 20 (2010) 1494–1501

Contents lists available at ScienceDirect

Digital Signal Processing

www.elsevier.com/locate/dsp

# Fast *k*-nearest neighbors search using modified principal axis search tree

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

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

ARTICLE INFO

*Article history:*
Available online 2 February 2010

*Keywords:*
*k*-nearest neighbors
Fast algorithm
Principal axes
Search tree

ABSTRACT

The problem of *k*-nearest neighbors (*kNN*) is to find the nearest *k* neighbors for a query point from a given data set. Among available methods, the principal axis search tree (PAT) algorithm always has good performance on finding nearest *k* neighbors using the PAT structure and a node elimination criterion. In this paper, a novel *kNN* search algorithm is proposed. The proposed algorithm stores projection values for all data points in leaf nodes. If a leaf node in the PAT cannot be rejected by the node elimination criterion, data points in the leaf node are further checked using their pre-stored projection values to reject more impossible data points. Experimental results show that the proposed method can effectively reduce the number of distance calculations and computation time for the PAT algorithm, especially for the data set with a large dimension or for a search tree with large number of data points in a leaf node.



## 1. Introduction

The problem of *k*-nearest neighbors (*kNN*) 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], and vector quantization [6,7].

The intuitive method of finding the nearest *k* neighbors for a query point $\mathbf{Q}$ from a data set $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$ of $n$ data points is to compute $n$ distances between the query point and all data points in the data set. Such 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 $\mathbf{Q} = [q_1, q_2, \ldots, q_d]$ with dimension $d$ and a data point $\mathbf{X}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]$ from the data set, the distance between these two points is defined as follows:

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

It is obviously to see that the process of finding nearest *k* neighbors for a query point using FSA is very time consuming. To reduce the computational complexity of the *kNN* finding process, many algorithms [8–17] were proposed. Among these methods, the principal axis search tree (PAT) algorithm has steadily performance for many types of benchmark data sets [13–15]. Using the PAT method, a search tree is off-line created according to the projection values of data points onto principal axes of tree nodes. The *kNN* finding process for a query point using the PAT method is to delete impossible nodes from the search tree using a node elimination criterion. Once a node is determined to be deleted, data points belonging to that node are impossible to be the *kNN* of the query point and distance calculations between the query point and those

* Corresponding author.
*E-mail address:* ycliaw@ms1.hinet.net (Y.-C. Liaw).

1051-2004/$ – see front matter 
doi:10.1016/j.dsp.2010.01.009

data points can be omitted. If a leaf node cannot be rejected, distances between the query point and all data points in the leaf node must be computed.

In this paper, a modified PAT (MPAT) algorithm is proposed to improve the computation time for the PAT method. The proposed method stores projection values for all data points in leaf nodes. In case a leaf node cannot be rejected using the node elimination criterion, data points in the leaf node are further checked using their projection values to avoid more unnecessary distance calculations. Through this additional checking step, the computation time and number of distance calculations can be effectively reduced.

This paper is organized as follows. In Section 2, the PAT algorithm is briefly reviewed. The modified PAT algorithm 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 [13] includes two processes that are the principal axis search tree construction process and the $k$-nearest neighbors search process. The principal axis search tree construction process is to partition a data set into distinct subsets using the principal component analysis (PCA) technique [18] and a tree structure. The *kNN* search process is to find the nearest $k$ neighbors for a query point from the constructed PAT. These two processes are described in the following subsections.

### 2.1. Principal axis search tree construction process

The PCA technique is a method for extracting the principal axis from a set of data points. The projection value of a data point onto the principal axis is the major feature to distinguish the data point from others. In general, data points with closer projection values will have smaller distance. To divide data points into distinct subsets, the PCA technique is applied here to extract the principal axis of data points. Once the principal axis of data points is available, projection values of data points onto the principal axis are evaluated and used for partitioning data points into several groups and each group contains data points of similar projection values.

To build the PAT for a given data set, we first create the root node for the PAT and assign all data points to it. That is, in the beginning, there is only one node in the PAT and that 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 root 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 the node should not be divided. Otherwise, data points must be partitioned into $n_c$ groups according to their projection values. In case data points in a node must be divided, the principal axis of data points in the node should be evaluated and recorded for the node. Projection values of data points onto the principal axis are then computed 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. Finally, child nodes of the node are sorted in ascending order according to their projection values and the minimum and maximum projection values of data points in a child node are stored in the node.

The above partition process is applied to the root node and recursively applied to each child node of the root node till terminal nodes (leaf nodes) all with numbers of data points less than $n_c$. After the PAT creation process, each leaf node records a list of data points. While 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 process

Given a query point $\mathbf{Q}$, our task is to find the nearest $k$ neighbors for $\mathbf{Q}$ from the PAT. The search process of *kNN* is started from finding a leaf node which has the most similar projection values to those of $\mathbf{Q}$. To find the leaf node, the projection value of $\mathbf{Q}$ onto the principal axis of the root node is first evaluated. Since child nodes of a node are sorted by projection values, the child node with the range of projection values containing $\mathbf{Q}$'s projection value can be found in $O(\log_2 n_c)$ using a binary search algorithm. The similar process is recursively applied to the child node which has the range of projection values containing $\mathbf{Q}$'s projection value, till a leaf node is reached. Once the leaf node is found, the partial distortion search (PDS) [9] is applied to find the *kNN* for $\mathbf{Q}$ from the leaf node.

After the leaf node with the most similar projection values to those of $\mathbf{Q}$ is searched, the finding process turns to check ancestors of the leaf node upward from the parent node of the leaf node to the root node. Let $\mathbf{N}_a$ be an ancestor of the leaf node. Every child node of $\mathbf{N}_a$ except the one with the range of projection values containing $\mathbf{Q}$'s projection value should be checked to see if it can be rejected in the *kNN* search process of $\mathbf{Q}$. Let $q_a$ be the projection value of $\mathbf{Q}$ onto the principal axis of $\mathbf{N}_a$. The checking sequence of $\mathbf{N}_a$'s child nodes is in increasing order of differences between their projection values and $q_a$. That is, the child node with the range of projection values closest to $q_a$ is first checked and the one with the range of projection values farthest to $q_a$ will be checked last. If a child node cannot be rejected in the checking process, its child nodes must be recursively checked.

Before introducing the node checking process, we would like to define the lower bound and the boundary point for a node first. Since a node has explicit range of projection values onto principal axes of its parent nodes, a node can be seen as a hypercube. The lower bound of a node can then be defined as the distance from **Q** to the boundary of the node and the boundary point for **Q** to a node is the projection point of **Q** onto the boundary of the node.

In the beginning of the checking process for $\mathbf{N}_a$, the lower bound and boundary point of $\mathbf{N}_a$ are set as 0 and **Q**, respectively. The checking process for the node $\mathbf{N}_a$ and its descendants is described as follows.

Let **N** be a node whose child nodes are going to be checked, $d_{\mathrm{LB}}$, **B**, and **P** be the lower bound, the boundary point, and the principal axis of **N**, respectively, $b$ be the projection value of **B** onto **P**, and $p^m_{\min}$ and $p^m_{\max}$ individually represent the minimum and maximum projection values for data points in $m$th child node of **N**. The checking sequence of child nodes in **N** is in ascending order of 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 lower bound of 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 can be evaluated using the following equation:

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

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

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

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,

$$d_k \leqslant d^m_{\mathrm{LB}}, \tag{4}$$

where $d_k$ is the distance between **Q** and the candidate of its $k$th 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 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 child node is a leaf node, distances between **Q** and all data points in the child node are computed with PDS applied. If a child node $m$ cannot be rejected and the child node is an internal node, its child nodes should be further checked. In such a case, the boundary point $\mathbf{B}^m$ of child node $m$ is evaluated using Eq. (5) or Eq. (6):

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

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

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 $b$.

## 3. Modified principal axis search tree algorithm

The *kNN* finding process for a query point using the PAT method is to reject impossible nodes from a PAT using the node elimination criterion. If a leaf node cannot be rejected, distances between the query point and all data points in the leaf node must be computed.

To speed up the search process of the PAT method, if a leaf node cannot be deleted using the node elimination criterion, our attempt is to check whether a data point in the leaf node is impossible to be the *kNN* of the query point and could be rejected instead of computing distances between the query point and all data points in the leaf node directly. To achieve this goal, projection values for every data point in leaf nodes should be stored during the PAT creation process. Data points in a leaf node should also be sorted in ascending order according to their projection values.

Once projection values for data points in leaf nodes are available, an inequality similar to inequality (4) can be applied to reject impossible data points. Let $\mathbf{N}_l$ be a leaf node whose data points are going to be checked, $d_{\mathrm{LB}}$, **P**, and **B** be the lower bound, the principal axis, and the boundary point of $\mathbf{N}_l$'s parent, respectively, $b$ be the projection value of **B** onto **P**, and $p^l_i$ denote the projection value of a data point $\mathbf{X}_i$ in $\mathbf{N}_l$ onto **P**. A data point in $\mathbf{N}_l$ can be checked using the following inequality to see whether the data point can be rejected or not,

$$d_k \leqslant d_{\mathrm{LB}} + \left(b - p^l_i\right)^2. \tag{7}$$

The above data point rejection inequality can be used jointly with inequality (4). That is, in the finding process of *kNN* for a query point, inequality (4) is first applied to reject unlikely nodes. For a leaf node which cannot be rejected by inequality (4), inequality (7) is used to check every data point in the leaf node. If inequality (7) is satisfied for a data point, the data point can be rejected. Otherwise the distance between the query point and the data point must be computed.

Since data points in a leaf node are arranged in ascending order according to their projection values, once a data point with a projection value close to $b$ is rejected, data points having projection values of further distance to $b$ than that of the rejected data point can also be rejected. To reject multiple data points at a time, the most commonly used method

```
searchLeaf(N_l, Q, d_LB, b, type)
{
    switch (type) {
      case 'L':
          for (X_i = from the last point to the first point in N_l) {
          evaluate d = d_LB + (b − p_i^l)^2
          if (d_k ⩽ d) terminate the search process of this node.
          else compute D(Q, X_i) and update d_k and the kNN set.
          }
          break;
      case 'R':
          for (X_i = from the first point to the last point in N_l) {
          evaluate d = d_LB + (p_i^l − b)^2
          if (d_k ⩽ d) terminate the search process of this node.
          else compute D(Q, X_i) and update d_k and the kNN set.
          }
          break;
      case 'M':
          for (each data point X_i in N_l) {
          evaluate d = d_LB + (b − p_i^l)^2
          if (d_k ⩽ d) delete data point X_i.
          else compute D(Q, X_i) and update d_k and the kNN set.
          }
      }
}
```

**Fig. 1.** Pseudo-codes of the *searchLeaf* function.

is to check data points in ascending order of differences between projection values of data points and $b$. In using such a method, the data point with the most similar projection value to $b$ must be determined first. This action induces additional computation load and decreases the speed of finding *kNN* from a leaf node. To avoid the determination process of the data point with the most similar projection value to $b$, leaf nodes are classified into three different types. Data points in leaf nodes of different types will be checked in different orders.

By observing the relationship between projection values of data points in a leaf node and $b$, a leaf node can be categorized into one of the following types:

$L$ type: Projection values of data points in the leaf node all smaller than $b$.

$R$ type: Projection values of data points in the leaf node all greater than $b$.

$M$ type: Some data points in the leaf node with projection values smaller than $b$ and others not.

Since the minimum and maximum projection values of data points in a child node are already stored in the PAT, the type of a leaf node can be determined using such information. Let $p^l_{\min}$ and $p^l_{\max}$ represent the minimum and maximum projection values of data points in leaf node $\mathbf{N}_l$, respectively. The type of leaf node $\mathbf{N}_l$ can be determined using the following equation:

$$\begin{cases} L \text{ type, for } p^l_{\max} \leqslant b, \\ R \text{ type, for } b \leqslant p^l_{\min}, \\ M \text{ type, for } p^l_{\min} \leqslant b \leqslant p^l_{\max}. \end{cases} \tag{8}$$

For a leaf node of $L$ type, data points in the leaf node are checked from the last point to the first point. While for a leaf node of other types, data points are checked in the reverse order. If the type of a leaf node is $M$ type, every data point in the leaf node is checked using inequality (7) to see if they can be rejected. In case a leaf node is of $L$ or $R$ type, during the checking process of data points in the leaf node, once a data point is determined could be rejected, all other unchecked data points in the same leaf node can also be rejected.

Now, we would like to present our proposed algorithm. In our proposed method, the principal axis search tree building process is similar to that of the original PAT algorithm. The only difference is that the projection values for every data point onto its parent node are stored in our method. That is, our method requires additional space of $O(n)$ than the PAT algorithm, where $n$ is the number of total data points. For a query point $\mathbf{Q}$, we use the same node rejection process as that used in the PAT algorithm [13] to reject impossible nodes. Once a leaf node cannot be rejected in the node rejection process, a new function called *searchLeaf* is applied to speed up the *kNN* search process for the leaf node. The prototype of the new function is

$$\mathit{searchLeaf}(\mathbf{N}_l, \mathbf{Q}, d_{\mathrm{LB}}, b, \mathit{type}),$$

where five parameters in function *searchLeaf* represent the leaf node to be processed, the query point, the lower bound of $\mathbf{N}_l$'s parent, the projection value of $\mathbf{B}$ onto the principal axis of $\mathbf{N}_l$'s parent, and the type of $\mathbf{N}_l$, respectively. The pseudo-codes of the *searchLeaf* function are listed in Fig. 1.

**Table 1**
Average computing time (in milliseconds) and number of distance calculations to find nearest 3 neighbors from data sets with size of 16 807 and $n_l = 1$.

| Dimension | Computing time | | Number of distance calculations | |
|---|---|---|---|---|
| | PAT | MPAT | PAT | MPAT |
| 8 | 87.5 | 87.5 | 295 | 275 |
| 12 | 179.7 | 178.2 | 565 | 541 |
| 16 | 259.4 | 259.4 | 746 | 718 |
| 20 | 360.9 | 359.4 | 946 | 914 |
| 24 | 443.8 | 442.2 | 1064 | 1028 |
| 28 | 557.8 | 554.7 | 1243 | 1202 |

**Table 2**
Average computing time (in milliseconds) and number of distance calculations to find nearest 3 neighbors from data sets with size of 33 614 and $n_l = 2$.

| Dimension | Computing time | | Number of distance calculations | |
|---|---|---|---|---|
| | PAT | MPAT | PAT | MPAT |
| 8 | 123.5 | 117.2 | 561 | 478 |
| 12 | 271.9 | 259.4 | 1131 | 993 |
| 16 | 412.5 | 392.2 | 1534 | 1359 |
| 20 | 564.1 | 535.9 | 1926 | 1712 |
| 24 | 748.4 | 709.3 | 2337 | 2077 |
| 28 | 910.9 | 864.0 | 2664 | 2377 |

## 4. Experimental results

To evaluate the performance of the proposed algorithm, the uniform Markov sequence [15,19], one set of images, and the Statlog data set from Ref. [20] were used. In Example 1, the uniform Markov source was used to generate data sets of various dimensions. In Example 2, codebooks with various sizes were generated using the Generalized Lloyd Algorithm (GLA) [21] from one set of images. In the third example, the real data set Statlog with size of 6435 and dimension of 36 was used.

The proposed algorithm MPAT was compared to the PAT algorithm [13] in terms of the average number of distance calculations and computing time per data point. For these two algorithms, search trees were both constructed with $n_c = 7$, where $n_c$ is the number of child nodes. All computing was performed on a Pentium Dual-Core E5200 with a clock rate of 2.5 GHz and memory of 2 GB. All programs were implemented as console applications of Microsoft Visual Studio 2008 and executed under Windows XP Professional SP3. The preprocessing time for constructing the search tree and evaluating the principal axes of nodes and the projection values of data points was not included in the computing time.

### 4.1. Example 1: Uniform Markov sequence

In this example, several data sets with dimensions ranging from 8 to 28 were generated from the uniform Markov sequence, which is in the range of $-10$ to 10, with $a = 0.9$, where $a$ is the correlation coefficient. The uniform Markov sequence $\{y_i\}$ was generated using the following equation:

$$y_{i+1} = \alpha y_i + u, \tag{9}$$

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

To see the influence for the number of data points in the leaf node ($n_l$) on the performance of the proposed algorithm, several data sets with different $n_l$ were generated from the uniform Markov sequence. It is noted that there are only six possible values of $n_l$ exist for the case of $n_c = 7$. The average query time and number of distance calculations per data point were calculated using 1000 query points. Each query point was obtained using the mean value of four data points selected randomly from the corresponding data set.

Tables 1–6 show the average query time and number of distance calculations per data point for using the PAT and MPAT algorithms to find nearest 3 neighbors of 1000 query points from data sets with $n_l = 1$–6, respectively. From these tables, we may find that our method outperforms the PAT method in all cases. Compared with the PAT algorithm, our approach can reduce the computing time by up to 10%. In terms of number of distance calculations, our approach can reduce the distance calculation by 3–22%. It is noted that the improvement of our method to the PAT method is more remarkable for the case of a larger $n_l$ present or a data set with higher dimension.

Table 7 gives the percentages of leaf node types. From this table, we can find that the sum of percentages of $L$ and $R$ types is raised when the number of $n_l$ increase. This is the reason that the improvement of our method over the PAT method is more remarkable when a larger $n_l$ present. From Table 7, we can also see that the percentage of $L$ type is greater than others. That is caused by the traverse order of child nodes is started from the one with projection values less than that of $\mathbf{Q}$ in PAT algorithm.

**Table 3**
Average computing time (in milliseconds) and number of distance calculations to find nearest 3 neighbors from data sets with size of 50 421 and $n_l = 3$.

| Dimension | Computing time | | Number of distance calculations | |
|---|---|---|---|---|
| | PAT | MPAT | PAT | MPAT |
| 8 | 151.6 | 139.1 | 759 | 619 |
| 12 | 364 | 339.1 | 1625 | 1382 |
| 16 | 567.2 | 529.7 | 2316 | 1993 |
| 20 | 770.3 | 717.2 | 2859 | 2474 |
| 24 | 992.2 | 923.5 | 3420 | 2961 |
| 28 | 1209.4 | 1120.3 | 3883 | 3367 |

**Table 4**
Average computing time (in milliseconds) and number of distance calculations to find nearest 3 neighbors from data sets with size of 67 228 and $n_l = 4$.

| Dimension | Computing time | | Number of distance calculations | |
|---|---|---|---|---|
| | PAT | MPAT | PAT | MPAT |
| 8 | 178.1 | 164 | 978 | 782 |
| 12 | 431.2 | 398.5 | 2085 | 1745 |
| 16 | 693.8 | 639.1 | 2974 | 2514 |
| 20 | 968.8 | 892.2 | 3813 | 3249 |
| 24 | 1250.0 | 1145.3 | 4548 | 3879 |
| 28 | 1545.4 | 1412.5 | 5227 | 4466 |

**Table 5**
Average computing time (in milliseconds) and number of distance calculations to find nearest 3 neighbors from data sets with size of 84 035 and $n_l = 5$.

| Dimension | Computing time | | Number of distance calculations | |
|---|---|---|---|---|
| | PAT | MPAT | PAT | MPAT |
| 8 | 200 | 182.8 | 1134 | 892 |
| 12 | 518.8 | 476.6 | 2572 | 2129 |
| 16 | 825 | 756.2 | 3662 | 3077 |
| 20 | 1137.5 | 1037.5 | 4622 | 3904 |
| 24 | 1462.5 | 1331.2 | 5495 | 4651 |
| 28 | 1809.4 | 1640.7 | 6334 | 5372 |

**Table 6**
Average computing time (in milliseconds) and number of distance calculations to find nearest 3 neighbors from data sets with size of 100 842 and $n_l = 6$.

| Dimension | Computing time | | Number of distance calculations | |
|---|---|---|---|---|
| | PAT | MPAT | PAT | MPAT |
| 8 | 214.1 | 193.8 | 1267 | 990 |
| 12 | 568.8 | 518.7 | 2918 | 2398 |
| 16 | 932.9 | 848.4 | 4273 | 3565 |
| 20 | 1292.2 | 1168.8 | 5394 | 4525 |
| 24 | 1671.8 | 1507.8 | 6471 | 5437 |
| 28 | 2050.0 | 1843.7 | 7391 | 6232 |

**Table 7**
Percentages of different leaf node types in finding nearest 3 neighbors from data sets with various $n_l$ and dimension = 16.

| Size ($n_l$) | Type of a leaf node | | |
|---|---|---|---|
| | *L* type | *M* type | *R* type |
| 16 807 ($n_l = 1$) | 47.8% | 26.0% | 26.2% |
| 33 614 ($n_l = 2$) | 44.0% | 22.6% | 33.3% |
| 50 421 ($n_l = 3$) | 42.8% | 21.7% | 35.5% |
| 67 228 ($n_l = 4$) | 41.7% | 21.3% | 37.0% |
| 84 035 ($n_l = 5$) | 41.4% | 21.1% | 37.5% |
| 100 842 ($n_l = 6$) | 41.4% | 20.9% | 37.7% |

**Table 8**
Average computing time (in milliseconds) and number of distance calculations to find the nearest neighbor from data sets with various $n_l$.

| Size ($n_l$) | Computing time | | Number of distance calculations | |
|---|---|---|---|---|
| | PAT | MPAT | PAT | MPAT |
| 2401 ($n_l = 1$) | 42.1 | 42.0 | 88 | 84 |
| 4802 ($n_l = 2$) | 57.8 | 54.7 | 176 | 155 |
| 7203 ($n_l = 3$) | 67.2 | 65.6 | 244 | 208 |
| 9604 ($n_l = 4$) | 87.5 | 84.4 | 328 | 276 |
| 12 005 ($n_l = 5$) | 110.9 | 106.3 | 406 | 339 |
| 14 406 ($n_l = 6$) | 125.0 | 117.2 | 446 | 368 |

**Table 9**
Average computing time (in milliseconds) and number of distance calculations to find nearest 3 neighbors from the Statlog data set using two algorithms.

| Algorithms | Computing time | Number of distance calculations |
|---|---|---|
| PAT | 231.3 | 489 |
| MPAT | 220.3 | 433 |

### *4.2. Example 2: Codebooks*

In this example, data sets, which consist of 2401 to 14 406 data points with dimension 16, were generated from six images ("Lena," "Peppers," "Baboon," "Airplane," "Island," and "Parrot") using GLA [21]. Each data point is a non-overlapping $4 \times 4$ pixel block obtained from these images.

The average query time and number of distance calculations were determined using 2000 query points selected randomly from these six training images. Table 8 shows the average query time and number of distance calculations per data point to find the nearest neighbors for 2000 query points and reveals that our method outperforms the PAT method in all cases for a data set from real images. Comparing with the PAT method, MPAT can reduce the computing time and number of distance calculations per data point by up to about 6.2% and 17.5%, respectively.

### *4.3. Example 3: The Statlog data set*

In this example, the Statlog data set [20], which consists of spectral values from a satellite image, will be used to estimate the performance of PAT and MPAT 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 to 255. The average query time and number of distance calculations per data point were calculated using 2000 query points. Each query point was obtained using the mean value of four data points selected randomly from the Statlog data set. Table 9 gives the computing time and number of distance calculations per data point to find three nearest neighbors for two algorithms. From Table 9, we can see that the MPAT method is still better than the PAT method both in terms of the computing time and number of distance calculations. Compared to the PAT method, MPAT can reduce the computing time and number of distance calculations per data point in average by about 5% and 11%, respectively.

## 5. Conclusions

In this paper, we have presented a new *kNN* search algorithm to reduce the computation time for the PAT algorithm. The proposed algorithm stores projection values for all data points in leaf nodes. If a leaf node in the PAT cannot be rejected by the node elimination criterion of the PAT, data points in the leaf node are further checked using their projection values to reject more impossible data points. Experimental results show that our proposed method can effectively reduce the number of distance calculations and computation time for the PAT algorithm, especially for the data set with a large dimension or for a search tree with large number of data points in a leaf node. Comparing with the PAT algorithm, the proposed algorithm can reduce the computing time and number of distance calculations by up to 10% and 22%, respectively, for a data set is from the uniform Markov sequence. When using real images as the data set, the computing time and number of distance calculations can be reduced by up to 6.5% and 17.5%, respectively. While using the Statlog data set, the proposed method can reduce the computation time and number of distance calculations by 5% and 11%, respectively.

## Acknowledgment

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, Academic Press, USA, 2003.
[2] H. Murase, S.K. Nayar, Visual learning and recognition of 3D objects from appearance, Int. J. Comput. Vision 1 (1995) 5–24.
[3] V. Roth, J. Laub, M. Kawanabe, J.M. Buhmann, Optimal cluster preserving embedding of nonmetric proximity data, IEEE Trans. Pattern Anal. Mach. Intell. 12 (2003) 1540–1551.
[4] Y.C. Liaw, Improvement of the fast exact pairwise-nearest-neighbor algorithm, Pattern Recognition 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 Lett. 4 (1995) 339–345.
[6] Y.C. Liaw, J.Z.C. Lai, Winston Lo, Image restoration of compressed image using classified vector quantization, Pattern Recognition 2 (2002) 329–340.
[7] A. Gersho, R.M. Gray, Vector Quantization and Signal Compression, Kluwer Academic Publishers, Boston, 1991.
[8] D.-Y. Cheng, A. Gersho, B. Ramamurthi, Y. Shoham, Fast search algorithms for vector quantization and pattern matching, in: IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, 1984, pp. 9.11.1–9.11.4.
[9] C.D. Bei, R.M. Gray, An improvement of the minimum distortion encoding algorithm for vector quantization, IEEE Trans. Commun. 10 (1985) 1132–1133.
[10] B.S. Kim, S.B. Park, A fast $k$ nearest neighboring finding algorithm based on the ordered partition, IEEE Trans. Pattern Anal. Mach. Intell. 6 (1986) 761–766.
[11] L. Mico, J. Oncina, R.C. Carrasco, A fast branch and bound nearest neighbor classifier in metric spaces, Pattern Recognition Lett. 7 (1996) 731–739.
[12] S.A. Nene, S.K. Nayar, A simple algorithm for nearest neighbor search in high dimensions, IEEE Trans. Pattern Anal. Mach. Intell. 9 (1997) 989–1003.
[13] J. McNames, A fast nearest-neighbor algorithm based on a principal axis search tree, IEEE Trans. Pattern Anal. Mach. Intell. 9 (2001) 964–976.
[14] Ben Wang, John Q. Gan, Integration of projected clusters and principal axis trees for high-dimensional data indexing and query, in: Lecture Notes in Computer Science, vol. 3177, Springer, Berlin, 2004, pp. 191–196.
[15] 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.
[16] J.S. Pan, Z.M. Lu, S.H. Sun, An efficient encoding algorithm for vector quantization based on sub-vector technique, IEEE Trans. Image Process. 12 (3) (2003) 265–270.
[17] Shu-Chuan Chu, Zhe-Ming Lu, Jeng-Shyang Pan, Hadamard transform based fast codeword search algorithm for high-dimensional VQ encoding, Inf. Sci. 177 (3) (2007) 734–746.
[18] I.T. Jolliffe, Principal Component Analysis, Springer, New York, 2002.
[19] W.H. Cooley, P.R. Lohnes, Multivariate Data Analysis, Wiley, New York, 1971.
[20] Statlog (Landsat Satellite) data set, http://archive.ics.uci.edu/ml.
[21] Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design, IEEE Trans. Commun. 28 (1) (1980) 84–95.



**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.

**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.

**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 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 was an instructor in the Department of Electronic Engineering, China Institute of Technology, Taiwan. From 1999 to 2005, he was 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.