Pattern Recognition 43 (2010) 3958–3968

Contents lists available at ScienceDirect

Pattern Recognition

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

Fast agglomerative clustering using information of *k*-nearest neighbors

Chih-Tang Chang [b], Jim Z.C. Lai [a,*], M.D. Jeng [b]

[a] Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung, Taiwan 202, ROC
[b] Department of Electrical Engineering, National Taiwan Ocean University, Keelung, Taiwan 202, ROC

ARTICLE INFO

Article history:
Received 2 November 2009
Received in revised form
13 June 2010
Accepted 27 June 2010

Keywords:
Nearest neighbor
Agglomerative clustering
Vector quantization

ABSTRACT

In this paper, we develop a method to lower the computational complexity of pairwise nearest neighbor (PNN) algorithm. Our approach determines a set of candidate clusters being updated after each cluster merge. If the updating process is required for some of these clusters, *k*-nearest neighbors are found for them. The number of distance calculations for our method is $O(N^2)$, where N is the number of data points. To further reduce the computational complexity of the proposed algorithm, some available fast search approaches are used. Compared to available approaches, our proposed algorithm can reduce the computing time and number of distance calculations significantly. Compared to FPNN, our method can reduce the computing time by a factor of about 26.8 for the data set from a real image. Compared with PMLFPNN, our approach can reduce the computing time by a factor of about 3.8 for the same data set.

1. Introduction

Data clustering is used frequently for a number of applications, such as vector quantization (VQ) [1–4], pattern recognition [5], knowledge discovery [6], speaker recognition [7], fault detection [8], and web/data mining [9]. Among clustering formulations that minimize an objective function, *k*-means clustering is perhaps the most widely used and studied one [10]. The *k*-means clustering algorithm, which is also called the generalized Lloyd algorithm (GLA), is a special case of the generalized hard clustering scheme, when point representatives are adopted and the squared Euclidean distance is used to measure the dissimilarity between a vector $\mathbf{x}$ and its cluster representative (cluster center) $\mathbf{c}$. Bisecting *k*-means (BKM) clustering and partitioning around medoids (PAM) clustering are two variants of *k*-means clustering [11]. Using BKM, the center of the whole data set is used in the beginning. At each step, a cluster is selected and divided into two clusters until the desired number of clusters is reached. PAM chooses a medoid rather the mean for each cluster at the iterative process.

The main drawback of GLA is that it gets stuck to the local optimal solution. To solve this problem, simulated annealing was proposed [12]. However, simulated annealing requires a large amount of computing time and gains a little improvement only [13]. Another approach is the hierarchical clustering [11,14]. The hierarchical clustering can be further divided into divisive approach [15] and agglomerative approach [14]. The agglomerative hierarchical clustering is also called the pairwise nearest neighbor (PNN) algorithm [16]. In Ref. [16], the *k*-nearest-neighbor graph is used for agglomerative clustering. Compared to GLA with randomly selected initial codebooks, adopting the cluster centers obtained using hierarchical agglomerative clustering as the initial codebook for GLA can usually obtain a better clustering result [16,17]. For a data set of N training vectors, the computational complexity, in terms of the number of distance calculations, of GLA is $O(NMt)$, where M is the number of clusters, and t is the number of iterations. It is noted that $M \ll N$ and $t \ll N$, in general. The computational complexity of hierarchical agglomerative clustering is $O(N^3)$ [14,18].

To reduce the running time of PNN algorithm, Kurita proposed a method of storing all pairwise cluster distances into a heap structure for reducing distance calculations [19]. However, this method requires $O(N^2)$ memory to store distances and it is impractical for large data sets. The computational complexity of Kurita's method is in the order of $N^2 \log_2 N$. Fränti et al. [17] also proposed a fast PNN (FPNN) algorithm. The computational complexity of FPNN algorithm is $O(\tau N^2)$, where τ is the average number of clusters to be updated for each stage of cluster merge. Kaukoranta et al. [20] presented the lazy pairwise nearest neighbor method (Lazy) to postpone some distance calculations in the process of cluster merge. Virmajoki et al. [21] proposed a fast PNN algorithm, which combines PDS (partial distortion search) [22], MPS (mean-distance-ordered partial search) [25], and Lazy [20] to speed up FPNN. This algorithm is referred to as PMLFPNN in this paper.

Recently, several interesting clustering algorithms are also developed for different purposes. Kashef and Kamel [11] developed a cooperative clustering model, which involves cooperation among multiple clustering techniques, to handle data sets with different properties. In Ref. [23], an incremental nested partition method is developed to reveal different levels of

* Corresponding author.
E-mail address: zclai@mail.ntou.edu.tw (J.Z.C. Lai).

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.06.021

information hidden in data. Ayad and Kamel [24] developed an algorithm to improve the clustering accuracy, stability, and estimation of the true number of clusters.

In this paper, we will present a method to reduce the computational complexity of PNN algorithm. Our method requires the information of nearest neighbors for every cluster at each cluster merge. Therefore, after each merge, our approach first determines a set of candidate clusters being updated. If the updating process is required for some of these clusters, nearest neighbors are found for them. To further reduce the computational complexity of our method, fast search presented in Ref. [26] is used to determine the k-nearest neighbors for a cluster. Using our approaches, k-nearest neighbors for each data point should be determined in the initialization process. The fast k-nearest-neighbor search algorithm (FKNNSA) [27] will be used to determine k-nearest neighbors of a data point in the initialization phase of our proposed method. MPS [25] uses one projection value on an axis to reject unlikely candidates during the process of nearest neighbor search; while FKNNSA adopts three projection values on three axes to delete impossible data points.

This paper is organized as follows. Section 2 describes the PNN and FPNN algorithms. Section 3 presents the algorithm developed in this paper. Some experimental results are given in Section 4 and concluding remarks are presented in Section 5.

2. Pairwise nearest neighbor algorithm

The agglomerative clustering method, which is also called the PNN (pairwise nearest neighbor) algorithm, partitions a set of N training vectors into M clusters through a sequence of merge operations. The dissimilarity between two vectors $\mathbf{x}=[x_1, x_2,\ldots, x_d]^t$ and $\mathbf{y}=[y_1, y_2,\ldots, y_d]^t$ is defined as the squared Euclidean distance between these two vectors. That is, $d(\mathbf{x},\mathbf{y})$ is defined by the following equation:

$$d(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} |x_i\text{-}y_i|^2 \tag{1}$$

The increase of dissimilarity of merging two clusters R_a and R_b into one cluster R_{ab} can be calculated by the following equation [28]:

$$D_{a,b} = \frac{n_a n_b}{n_a+n_b} d(\mathbf{c}_a,\mathbf{c}_b) \tag{2}$$

where $n_a=|R_a|$ is the number of training vectors in R_a; $n_b=|R_b|$is the number of training vectors in R_b; $\mathbf{c}_a$ is the center of R_a; and $\mathbf{c}_b$ is the center of R_b. $D_{a,b}$ is called the cluster distance between R_a and R_b. The cluster center $\mathbf{c}_{ab}$ and cardinality n_{ab} (the number of training vectors) of R_{ab} are updated as follows:

$$\mathbf{c}_{ab} = (n_a\mathbf{c}_a+n_b\mathbf{c}_b)/(n_a+n_b) \tag{3}$$

$$n_{ab} = n_a+n_b \tag{4}$$

At each stage of merge, two clusters which have the least cluster distance are determined and merged. The merged cluster should update its center and cardinality using Eqs. (3) and (4). For the reason of simplicity, clusters R_a and R_b will be abbreviated to clusters a and b, respectively. The PNN algorithm is now presented as follows:

(1) Set N clusters, each with one data point and let $N_t=N$.
(2) Find two clusters a and b, which have the minimum cluster distance. Merge clusters a and b into cluster ab and let $N_t=N_t-1$.
(3) Update $\mathbf{c}_{ab}$ and n_{ab} using Eqs. (3) and (4). Set $n_a=n_{ab}$, $\mathbf{c}_a=\mathbf{c}_{ab}$ and delete cluster b.
(4) If $N_t > M$, go to step (2), where M is the number of clusters.

Since the PNN algorithm has extensive computations, Fränti et al. [19] used a nearest neighbor table NT, which records the nearest neighbor for each cluster to reduce the corresponding computational complexity, where $NT[j]$ denotes the nearest neighbor of cluster j. If two clusters a and b are merged, then a set S_u of clusters being updated is determined, where $S_u=\{l\colon NT[l]=a$ or $NT[l]=b\}\backslash\{a, b\}$. That is, S_u is the set of clusters, which have either cluster b or a as their nearest neighbor. The FPNN algorithm [17] is presented below for the reason of completeness.

FPNN algorithm

(1) For each cluster $l(l=1, 2,\ldots, N)$, find the corresponding nearest neighbor and generate the nearest neighbor table NT. Set $N_t=N$.
(2) Find two clusters a and b, which have the minimum cluster distance. Merge clusters a and b into cluster ab and let $N_t=N_t-1$.
(3) Determine the set $S_u=\{l\colon NT[l]=a$ or $NT[l]=b\}\backslash\{a, b\}$.
(4) Update $\mathbf{c}_{ab}$ and n_{ab}. Set $n_a=n_{ab}$, $\mathbf{c}_a=\mathbf{c}_{ab}$ and delete cluster b. For each cluster in S_u, find the corresponding nearest neighbor and update the table NT.
(5) If $N_t > M$, go to step (2), where M is the number of clusters.

3. Proposed algorithm

In this section, we will present a fast agglomerative clustering algorithm. Our method uses the k-nearest neighbors of a cluster to reduce the number of nearest neighbor search for a cluster after each cluster merge. Double linked algorithm (DLA) adopts k neighbors (not real k-nearest neighbors) of some clusters to find the approximate nearest neighbors for a cluster after each merge process [16]. That is, our proposed method is an exact version of PNN and DLA is not. Therefore, our approach can obtain a clustering result as good as that of PNN or FPNN. It is noted here that the parameter k is a knob that adjusts between approximate solutions and running time for DLA. Now, we will briefly discuss DLA below.

Denote by KNT the k-nearest-neighbor table with $KNT[l]$ consisting of k-nearest clusters for cluster l. For each cluster l, DLA also maintains a set of clusters, which have cluster l as one of their k-nearest neighbors. This set of clusters for cluster l is denoted as $INNS_l$, where $INNS_l=\{j\colon l\in KNT[j]\}$. In this paper, $INNS_l$ is called the inverse nearest-neighbor set for cluster l. After merging clusters a and b, DLA updates $INNS_j$ by setting $INNS_j=INNS_j\backslash\{a, b\}$ for each cluster $j\in S_u$. The k-nearest neighbors of cluster ab are determined from $KNT[a]\cup KNT[b]\backslash\{a, b\}$. For each cluster $c\in S_u$, DLA calculates $D_{a,c}$ and updates $KNT[c]=KNT[c]\backslash\{b\}$. Finally, for each cluster $j\in KNT[a]$, $INNS_j$ is updated by setting $INNS_j=INNS_j\cup\{a\}$. Now, we will present the double linked algorithm below.

Double linked algorithm

(1) Initialization phase:
  (1.1) Allocate N data points to N clusters with each cluster having one data point. Set $N_t=N$.
  (1.2) For each cluster $l(l=1\text{–}N)$, determine the corresponding k-nearest neighbors $KNT[l]$.
  (1.3) For each cluster $l(l=1\text{–}N)$, generate $INNS_l$.
(2) Merge phase:
  (2.1) From the KNT, find clusters a and b, which have the minimum cluster distance, and merge clusters a and b into cluster ab.

(2.2) Determine $KNT[ab]$ from $KNT[a] \cup KNT[b] \backslash \{a, b\}$.
(2.3) Update n_{ab} and $\mathbf{c}_{ab}$.
(2.4) Determine S_u and delete cluster b, where $S_u = INNSa \cup INNS_b \backslash \{a, b\}$.
(3) Update phase:
(3.1) For each cluster $j \in S_u$, update $KNT[j]$ by $KNT[j] = KNT[j] \backslash \{b\} \cup \{a\}$.
(3.2) Replace $KNT[a]$, n_a and $\mathbf{c}_a$ by $KNT[ab]$, n_{ab}, and $\mathbf{c}_{ab}$, respectively, and update $INNS_a = \emptyset$, where Ø is an empty set.
(3.3) For each cluster $j \in S_u$, calculate $D_{a,j}$.
(3.4) For each cluster $j \in S_u$, update $INNS_a = INNS_a \cup \{j\}$.
(3.5) For each cluster $j \in KNT[a]$, update $INNS_j = INNS_j \cup \{a\}$.
(4) Set $N_t = N_t - 1$. Repeat steps (2) and (3) until $N_t \le M$.

It is noted again that FKNNSA [27] will be used only in the initialization phase of our proposed algorithm. FKNNSA uses a data point's projections on some axes and triangular inequality to reject impossible candidates during the process of determining k-nearest neighbors for the data point. Using this approach, FKNNSA avoids many distance calculations for kNN search.

For a data set, denote $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ as three eigenvectors corresponding to the three largest eigenvalues. Let q_l be the projection value of a query point $\mathbf{q}$ on the lth axis. That is, q_l is the inner product of $\mathbf{q}$ and $\mathbf{v}_l$, and can be calculated as follows:

$$q_l = \langle \mathbf{q}, \mathbf{v}_l \rangle \tag{5}$$

Similarly, denote x_{jl} as the projection value of a data point $\mathbf{x}_j$ on the lth axis. To speed up the searching process, FKNNSA sorts all data points in the ascending order of their projections on the first axis.

In the beginning of determining k-nearest neighbors for a query point $\mathbf{q}$, FKNNSA first selects k data points and calculate their distances to $\mathbf{q}$. Let $\mathbf{x}_{lk}$ be the kth nearest neighbor of $\mathbf{q}$ and denote r_k as the distance between $\mathbf{x}_{lk}$ and $\mathbf{q}$. If another data point $\mathbf{x}_j$ cannot satisfy the following condition, it will be rejected directly by FKNNSA in the process of finding the kth closest neighbor for $\mathbf{q}$.

$$|x_{jl} - q_l| < r_k, \quad l = 1,2,3 \tag{6a}$$

It is shown in [27] that if a candidate data point $\mathbf{x}_j$ cannot satisfy the following inequality, it will be rejected to be a candidate for the kth neighbor of $\mathbf{q}$:

$$[(x_{j1}-q_1)^2 + (x_{j2}-q_2)^2 + (x_{j3}-q_3)^2] + (||\mathbf{s}_{xj}|| - ||\mathbf{s}_q||)^2 < (r_k)^2 \tag{6b}$$

where $||\mathbf{S}_{Xj}|| = [(||\mathbf{x}||)^2 - (x_{j1})^2 - (x_{j2})^2 - (x_{j3})^2]^{0.5}$ and $||\mathbf{S}_q|| = [(||\mathbf{q}||)^2 - (q_1)^2 - (q_2)^2 - (q_3)^2]^{0.5}$. FKNNA uses expressions (6a) and (6b) to reject impossible candidates for a query point.

3.1. Agglomerative clustering using k-nearest neighbors

For each cluster, our method will determine its k-nearest neighbors and arrange the corresponding k-nearest neighbors in the ascending order of cluster distances in the initialization process. Let cluster a be the nearest neighbor of cluster l, then we set $D_{min}(l) = D_{l,a}$, where $D_{l,a}$ defined by Eq. (2) is the cluster distance between clusters a and b. From KNT, we can easily find the two clusters being merged. Denote these two clusters being merged as a and b. From $INNS_a$ and $INNS_b$, we can find the corresponding set of clusters whose nearest neighbors should be updated. Denote a set of clusters being updated as S_u, where $S_u = INNS_a \cup INNS_b \backslash \{a, b\}$. For a cluster $c \notin S_u$, we will have $c \notin INNS_a$ and $c \notin INNS_b$. That is, we have $D_{c,a} > D_{c,KNT[c][k]}$ and $D_{c,b} > D_{c,KNT[c][k]}$, where $KNT[c][k]$ is the kth nearest neighbor of cluster c. Since $D_{c,ab} > D_{c,b} > D_{c,KNT[c][k]}$ [19], the merged cluster ab is impossible to be in $KNT[c]$ for $c \notin S_u$. Therefore for any cluster not in S_u, it will not be updated after each cluster merge. Let the number of times for the set S_u being empty is E. The probability of S_u being empty is denoted as P_e, which is defined as follows:

$$P_e \frac{E}{N-M} \approx \frac{E}{N}$$

where N is the number of data points and M is the number of clusters being generated. P_e is less than 0.01% for our proposed method with $k = 3$–5. The value of P_e is small due to $KNT[j]$ which is empty after ζ steps of cluster merge, where $\zeta > k$.

At this stage, we will calculate the k-nearest neighbors of cluster ab ($KNT[ab]$) and replace all the information of cluster a by the one of ab, where ab is the merged cluster. The only information required to determine which two clusters being merged is the cluster distance between a cluster and its nearest neighbor. Therefore, if $|KNT[j]| > 0$ no search is performed to obtain k-nearest neighbors of cluster j even in the case of $|KNT[j]| < k$. For each cluster $j \in S_u$, calculate $D_{j,ab}$ and update $KNT[j] = KNT[j] \backslash \{a, b\}$. If $|KNT[j]| > 0$ and $D_{j,ab} \le D_{j,KNT[j][jm]}$, cluster ab is one of the k-nearest neighbors for cluster j, where $jm = |KNT[j]|$ is the cardinality of $KNT[j]$. In this case, we will add cluster ab to $KNT[j]$ and arrange $KNT[j]$ in the ascending order of cluster distances. If $|KNT[j]| > 0$ and $D_{j,ab} > D_{j,KNT[j][jm]}$, $KNT[j]$ will be unchanged. In the case that $|KNT[j]| = 0$ ($KNT[j]$ is empty), we will determine its k-nearest neighbors from all clusters. Finally, we should update the k-nearest-neighbor table KNT for cluster a.

The proposed fast agglomerative clustering algorithm using k-nearest neighbors is referred to as ACUN in this paper. ACUN consists of three phases: the initial phase, merge phase, and update phase. In the initial phase, $KNT[l]$ and $INNS_l$ are generated for each cluster l. In the merge phase, two clusters with the minimum cluster distance are found from KNT and these two clusters are merged into one cluster. Finally, the corresponding inverse nearest-neighbor sets are updated temporarily due to this cluster merge. The update phase can be divided into two steps: the initial step and final step. In the initial step, the cluster center, cardinality, and k-nearest neighbors of merged cluster ($KNT[ab]$) are determined. Also, for each cluster j in S_u, $KNT[j]$ is updated. In the final step of update phase, the nearest neighbors of clusters in S_u are determined. The merge and update phases of ACUN are repeated until the desired number of clusters is obtained. Now, we can present ACUN as follows:

ACUN

(1) Initialization phase:
(1.1) Allocate N data points to N clusters with each cluster having one data point. Set $N_t = N$.
(2.1) For each cluster l($l = 1$–N), use FKNNSA to determine the corresponding k-nearest neighbors $KNT[l]$ and arrange $KNT[l]$ in the ascending order of their cluster distances to l.
(3.1) For each cluster l($l = 1$–N), generate $INNS_l$.
(2) Merge phase:
(2.1) From the k-nearest-neighbor table KNT, find clusters a and b which have the minimum cluster distance, and merge clusters a and b into cluster ab.
(2.2) For $j = 1$ to $|KNT[b]|$, set $INNS_i = INNS_i \backslash \{b\}$, where $i = KNT[b][j]$.
(2.3) For $j = 1$ to $|KNT[a]|$, set $INNS_i = INNS_i \backslash \{a\}$, where $i = KNT[a][j]$.
(3) Update phase:
(3.1) Initial step:
(3.1.1) Update $\mathbf{c}_{ab}$ and n_{ab} using Eqs. (3) and (4) and find $KNT[ab]$.

(3.2.1) Determine $S_u = INNS_a \cup INNS_b \setminus \{a,b\}$ and set $\mathbf{c}_a = \mathbf{c}_{ab}$, $n_a = n_{ab}$, $KNT[a] = KNT[ab]$, as well as $D_{min}(a) = D_{min}(ab)$.

(3.3.1) For $j=1$ to $|KNT[a]|$, set $INNS_i = INNS_i \cup \{a\}$, where $i = KNT[a][j]$.

(3.2) Final step:

(3.2.1) For each cluster $j \in S_u$, update $KNT[j] = KNT[j] \setminus \{a,b\}$ and calculate $D_{j,a}$, which equals $D_{j,ab}$. Delete cluster b.

(3.2.2) If $D_{j,a} \le D_{j,KNT[j][jm]}$, add cluster a to $KNT[j]$ and update $INNS_a = INNS_a \cup \{j\}$, where $jm = |KNT[j]|$.

(3.2.3) If $KNT[j] = \emptyset$, determine $KNT[j]$ from all clusters. For $l=1$ to k, set $INNS_i = INNS_i \cup \{j\}$, where $i = KNT[j][l]$.

(4) Set $N_t = N_t - 1$. Repeat steps (2) and (3) until $N_t \le M$, where M is the number of clusters.

Now, we would like to summarize ACUN below: (a) Allocate N data points to N clusters. For each cluster l, generate $KNT[l]$ and $INNS_l$. (b) Find two clusters being merged and update $KNT[a]$ for the merged cluster a. Update $INNS_i$, where $i \in KNT[a]$. (c) Generate the set S_u, which consists of the clusters being updated. (d) For each cluster j in S_u, update $KNT[j]$ and $INNS_i$, where $i \in KNT[j]$.

The procedure of cluster merge can be depicted by a k-nearest-neighbor graph (k-NN graph). A k-NN graph is defined as a weighted directed graph, where a node represents a cluster and corresponding directed edges represent pointers to nearest neighbors. The graph shows that the clusters being updated consist of nearest nodes pointing to the merged clusters. An illustration of the k-NN graph with $k=2$ is given in Fig. 1, where clusters a and b having the minimum cluster distance are to be merged. In this illustration, $INNS_a = \{2, 3, b\}$ and $INNS_b = \{4, 8, a\}$. Therefore, we can obtain $S_u = INNS_a \cup INNS_b \setminus \{a, b\} = \{2, 3, 4, 8\}$. That is, the nearest neighbors of clusters 2, 3, 4, and 8 should be updated. Note here that $KNT[2] = \{a\}$, $KNT[3] = \{5,a\}$, $KNT[4] = \{1,b\}$, and $KNT[8] = \{3,b\}$. If $D_{2,ab} \le D_{2,KNT}[2][2]$, we will update $KNT[2] = KNT[2] \setminus \{a,b\} \cup \{a\} = \{2\}$, where the information of cluster a is replaced by the one of cluster ab. In the case of $D_{2,ab} > D_{2,KNT[2][2]}$, we will first obtain $KNT[2] = \emptyset$. Therefore the two nearest neighbors of cluster 2 should be determined from all clusters. To update cluster 3, we first check whether $D_{3,ab} \le D_{3,KNT[3][2]}$. If $D_{3,ab} \le D_{3,KNT[3][2]}$, then we will update $KNT[3] = KNT[3] \setminus (a,b) \cup \{a\} = \{5,a\}$. If $D_{3,ab} > D_{3,KNT[3][2]}$, we first update $KNT[3] = KNT[3] \setminus (a,b) = \{5\}$. Since $KNT[3]$ is not an empty set, we will not find nearest neighbors of cluster 3 from all clusters. Similar procedures can be applied to update clusters 4 and 8.

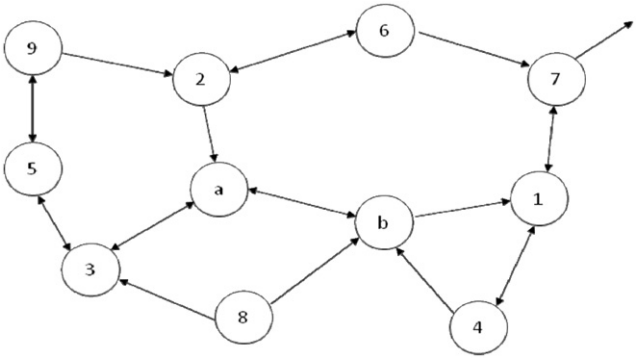

Fig. 1. An illustration of the k-NN graph ($k=2$), where a and b are the merged clusters.

Let β_i be the number of clusters in S_u at the ith step of cluster merge. Define $\beta = \left(\sum_{i=1}^{N-M} \beta_i\right)/(N-M)$ is the average value of β_i, where $\beta \ll N$. The probability of a cluster being updated at each step of cluster merge is β/N, where $\beta/N \ll 1$. After ζ steps of merge, we can expect that $KNT[j]$ is empty for a cluster j, where $\zeta \ll N$. In this case, the k-nearest neighbors of $KNT[j]$ are determined from all clusters for cluster j. That is, the nearest neighbor search is performed after ζ steps of cluster merge for ACUN. For FPNN, the nearest neighbor search is performed for each cluster in S_u after each step of cluster merge. This explains that ACUN can reduce the computations of FPNN. From our experiments, we find that the value of ζ is almost remained constant when $k \ge 6$. However, the cost of maintaining KNT increases as k increases. That is, we can conclude that the optimal value of k corresponding to the minimum computing time of ACUN may be less than 6. To further reduce the computational complexity of ACUN, PDS, MPS, Lazy [21] and FS (fast search algorithm) [26] can be used to determine the k-nearest neighbors for a cluster at steps (3.1.1) and (3.2.3) of ACUN. Adopting PDS, MPS, and Lazy for ACUN is denoted as PMLACUN. Similarly, using FS for ACUN is denoted as FACUN in this paper.

3.2. Complexity analysis

The computational complexity of ACUN is given in Table 1, where "steps" indicates the number of loops used to update data structures. Table 2 presents the numbers of distance calculations, comparisons, and steps of ACUN with $k=5$ to divide a data set of 16,384 data points with dimension 16 into 256 clusters. This data set is generated from a real image "Lena" [31]. Our proposed method uses FKNNSA to construct KNT in the initialization phase. The computational complexity, in terms of the number of distance calculations, of FKNNSA is $O(\tau_1 N)$, where τ_1 is the number of data points accessed to determine a query point's nearest neighbors and N is the number of data points. Step (1.1) of ACUN needs $O(N)$ steps; step (1.2) requires $O(\tau_1 N)$ distance calculations and $O(\beta_1 N)$ steps to construct KNT; and step (1.3) uses $O(kN)$ steps to construct $INNS_l$ for each cluster l, where $\tau_1 \ll N$ is the average number of data points accessed to determine a data point's k-nearest neighbors; β_1 is the average number of steps per data point; and k is the number of nearest neighbors for a cluster. That is, the initial phase of ACUN requires $O(\tau_1 N)$ distance calculations and $O((1+\beta_1+k)N)$ steps. In the merge phase of ACUN, step (2.1) requires $O(N)$ comparisons to find the clusters being merged; and steps (2.2) as well as (2.3) need $O(2k)$ steps. Therefore the merged phase of ACUN needs $O(N)$ comparisons and $O(2k)$ steps for each iteration. In the initial step of update phase, step (3.1.1) needs $O(N)$ distance calculations to find k-nearest neighbors of merged cluster and $O(1)$ steps to update the corresponding cluster center

Table 1
Estimated numbers of comparisons, steps, and number of distance calculations for ACUN.

	Initial phase	Merged phase per iteration	Initial step of update phase per iteration	Final step of update phase per iteration	Total
Comparisons	–	N	–	–	N^2
Steps	$(1+\beta_1+k)N$	$2k$	β_2+k	$\tau+k$	$(1+\beta_1+k)N + (3k+\beta_2+\tau)N$
Distance calculations	$\tau_1 N$	–	N	$\tau N/\zeta$	$(1+\tau/\zeta)N^2$

Table 2
Observed numbers of comparisons, steps, and number of distance calculations for ACUN using a data set obtained from "Lena."

	Initial phase	Merged phase per iteration	Initial step of update phase per iteration	Final step of update phase per iteration	Total
Comparisons		8189.5			134,176,896
Steps	1,401,971	4.6	14.1	50.4	2,532,727
Distance calculations	3,310,136	–	8189.5	7312.3	257,292,042

and cardinality. Steps (3.1.2) and (3.1.3) require $O(\beta_2)$ as well as $O(k)$ steps, respectively, where β_2 is about the same as the average value of $|S_u|$. Here, we can conclude that the initial step of update phase for ACUN has $O(N)$ distance calculations and $O(\beta_2+k)$ steps per iteration. In the final step of update phase, step (3.2.1) requires $O(1)$ distance calculations and $O(\tau)$ steps to update the clusters in S_u, where τ is the average value of $|S_u|$; and step (3.2.3) needs $O(\tau N/\zeta)$ distance calculations to update k-nearest neighbors of clusters in S_u after ζ steps of cluster merge, where $k<\zeta$ and τ is the average number of clusters in S_u. Steps (3.2.2) requires $O(k)$ steps to update data structure. That is, the final step of update phase for ACUN requires $O(\tau N/\zeta)$ distance calculations and $O(\tau+k)$ steps per iteration. Therefore, the computational complexity, in terms of the number of distance calculations, for ACUN is $O((1+\tau/\zeta)N^2)\approx O(N^2)$.

FKNNSA requires $O(\tau_1 N)$ distance calculations, where $\tau_1 \ll N$ for real data sets. The value of τ_1 depends on the distribution of data set. For a data set with evenly distributed objects, we may have $\tau_1 > N$ due to the calculations of projection values [29]. Fortunately, real data sets are rarely in the form of evenly distributed variables and the much better performance is usually obtained in practice for fast search algorithms using projections [30].

Here, we will present the novelties of our proposed method as follows: (1) use a cluster's k-nearest neighbors to reduce distance calculations; (2) only a small number of clusters in S_u instead of whole clusters are updated after each cluster merge; and (3) reduce distance calculations through determining whether the merged cluster, not all clusters, is in the set of nearest neighbors for a cluster in S_u.

4. Experimental results

To evaluate the performance of proposed algorithm, the Markov source [1], a residual image [19], two sets of original images from [31], and many synthetic data sets are used. In example 1, the data sets are generated from the Gauss Markov source. In the second example, the data set consists of one real image "Lena" [31]. Three real images: "Lena," "Baboon," and "Peppers" from [31] are used in example 3. In the fourth example, an original image "Lena" and a residual image obtained from the video sequence "Miss America" [16] are used. All computing is performed on an AMD Dual Opteron 2.0G with 2 GB of memory. The proposed algorithm is compared to FPNN [19] and PMLFPNN [21] in terms of the number of distance calculations and computing time. It should be noted that our proposed method will not generate exactly the same results as those of PNN or FPNN, although their differences are not significant any more. This is because there are many cluster pairs with the same cluster distance in the early stage of cluster merge and merging different pairs with the same cluster distance in the early stage will result in a little difference for the clustering results. Our method can obtain the clustering result as good as that of PNN or FPNN.

4.1. Example 1: Data sets are generated from the Gauss Markov source

In this example, several data sets are obtained from the Gauss Markov source with $\sigma=10$, $\mu=0$, and $a=0.9$, where σ is the standard deviation, μ is the mean value, and a is the correlation coefficient. Each data set with dimension ranging from 8 to 32 contains 5000 data points.

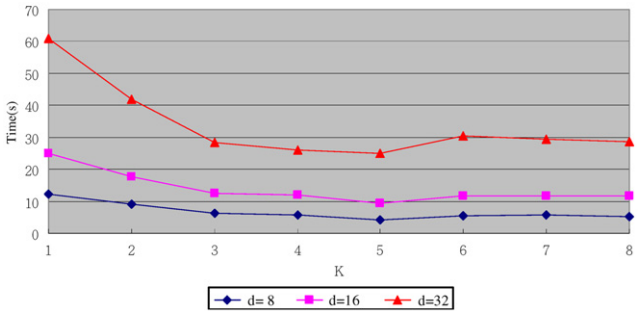

Fig. 2. The computing time (in seconds) of clustering data into 128 clusters for data sets generated from the Gauss Markov source.

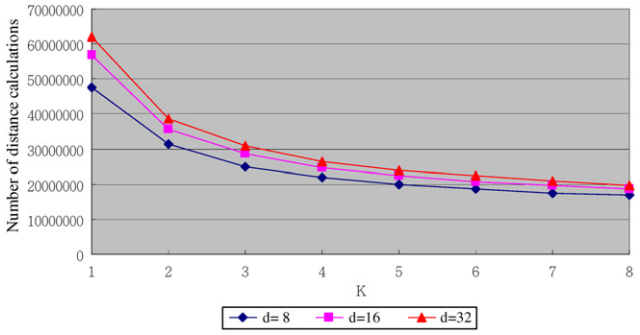

Fig. 3. Number of distance calculations of clustering data into 128 clusters for data sets generated from the Gauss Markov source.

Figs. 2 and 3 present the computing time and number of distance calculations, respectively, of dividing data sets into 128 clusters for ACUN with $k=2$–8. ACUN with $k=5$ gives the least computing time in average. Compared to ACUN with $k=3$ or 4, the decrease of computing time for ACUN with $k=5$ is not significant. Therefore, $k=3$ and 4 are also used for ACUN and FACUN in the following examples to save memory. From Fig. 3, we can find that when $k<6$, the number of distance calculations for ACUN deceases as k increases and is almost remained constant for $k\geq 6$. The computing time of ACUN increases as k increases when $k\geq 6$. This is due to the complexity of updating the k-nearest neighbors for a cluster increases as k increases.

4.2. Example 2: Using one image to generate the data set

In this example, the data set consisting of 16,384 data points with dimension 16 is generated from an original image "Lena." The block size for each data point is 4×4 in this example. Table 2 presents the numbers of distance calculations, comparisons, and steps of ACUN with $k=5$ to divide this data set into 256 clusters. Table 3 shows the mean square errors of FPNN and FACUN with

Table 3
Mean square errors of FPNN and FACUN using the data set from an image "Lena."

Method	M	
	256	512
FPNN	41.14	30.66
FACUN ($k=3$)	41.11	30.66
FACUN ($k=4$)	41.11	30.64
FACUN ($k=5$)	41.08	30.64

Table 4
The computing time (in seconds) of FPNN, DLA, PMLFPNN, ACUN, PMLACUN, and FACUN using the data set from an image "Lena."

Method	M		
	128	256	512
FPNN	322.73	321.63	322.73
DLA	3.05	3.00	3.00
PMLFPNN	40.47	40.45	40.27
ACUN ($k=3$)	118.72	118.41	118.02
ACUN ($k=4$)	105.70	105.50	105.06
ACUN ($k=5$)	96.44	96.63	97.19
PMLACUN ($k=3$)	28.58	28.52	28.50
PMLACUN ($k=4$)	33.14	33.13	33.06
PMLACUN ($k=5$)	32.05	31.97	31.84
FACUN ($k=3$)	12.30	12.38	12.33
FACUN ($k=4$)	11.95	11.98	11.98
FACUN ($k=5$)	12.06	12.11	12.02

$k=3$, 4, and 5. From Table 3, we can find that there is a little difference between the mean square errors of FPNN and FACUN.

4.2.1. Computing time

Table 4 presents the computing time of FPNN, DLA [16], PMLFPNN, ACUN, PMLACUN, and FACUN. It is noted here that DLA is not an exact version of FPNN. From Table 4, we can conclude that DLA has the least computing time and FACUN with $k=4$ is the best among the exact variants of PNN. It should be noted that DLA can provide a suboptimal solution only and our method can obtain a clustering result as good as that of FPNN. Compared to FPNN, FACUN with $k=5$ can reduce the computing time by a factor of about 26.8. Compared with PMLFPNN, FACUN with $k=5$ can reduce the computing time by a factor of about 3.8. Compared with FPNN, FACUN with $k=4$ can reduce the computing time by 96%.

4.2.2. Distance calculations and mean square error

Table 5 gives the number of distance calculations for FPNN, DLA, PMLFPNN, ACUN, PMLACUN, and FACUN. From Table 5, we can find that DLA gives the least number of distance calculations and FACUN with $k=5$ is the second best. Compared to FPNN, FACUN with $k=5$ can reduce the number of number of distance calculations by a factor of about 88.1. Compared with PMLFPNN, FACUN with $k=5$ can reduce the number of number of distance calculations by 77.7–79.3%. Table 6 presents the mean square errors for six methods. These methods, except DLA, have almost the same mean square errors and their differences are not significant anymore. Since DLA is not an exact version of PNN, it will obtain higher MSE (mean square error) in general.

4.3. Example 3: The data set is generated from three images.

In this example, the data set having of 59,152 data points with dimension 16 is generated using three original images "Lena," "Baboon," and "Peppers." The block size for each data point is also 4×4 in the third example. Table 7 presents the numbers of distance calculations, comparisons, and steps of ACUN with $k=5$ to divide this data set into 256 clusters. Table 8 presents the computing time. From Table 8, we can find that our method FACUN with $k=4$ has the best performance, in terms of the computing time, among the exact variants of FPNN. Compared to FPNN, FACUN with $k=5$ can reduce the computing time by a factor of about 18.6. Compared to PMLFPNN, FACUN with $k=5$ can reduce the computing time by a factor of about 4.9.

Table 5
Number of distance calculations for FPNN, DLA, PMLFPNN, ACUN, PMLACUN, and FACUN using the data set from an image "Lena."

Method	M		
	128	256	512
FPNN	622,447,541	622,357,490	621,979,736
DLA	6,679,125	6,678,131	6,675,993
PMLFPNN	42,624,071	42,595,119	42,490,070
ACUN ($k=3$)	323,258,888	323,206,360	322,982,363
ACUN ($k=4$)	280,381,105	280,335,476	280,132,325
ACUN ($k=5$)	254,144,700	254,100,840	253,913,429
PMLACUN ($k=3$)	18,876,619	18,858,952	18,786,564
PMLACUN ($k=4$)	17,413,207	17,396,016	17,329,231
PMLACUN ($k=5$)	16,488,128	16,471,538	16,406,956
FACUN ($k=3$)	8,069,326	8,060,711	8,031,526
FACUN ($k=4$)	7,437,600	7,428,871	7,399,256
FACUN ($k=5$)	7,068,937	7,059,362	7,029,075

Table 6
Mean square errors for FPNN, DLA, PMLFPNN, ACUN, PMLACUN, and FACUN using the data set from an image "Lena."

Method	M		
	128	256	512
FPNN	54.121899	41.138968	30.639848
DLA	54.120307	41.396088	30.771194
PMLFPNN	54.122876	41.085641	30.644361
ACUN ($k=3$)	54.134736	41.131362	30.661992
ACUN ($k=4$)	54.146235	41.098996	30.648818
ACUN ($k=5$)	54.056895	41.066463	30.634128
PMLACUN ($k=3$)	54.324590	41.158439	30.654418
PMLACUN ($k=4$)	54.274170	41.119261	30.650324
PMLACUN ($k=5$)	54.190855	41.146525	30.663374
FACUN ($k=3$)	54.150483	41.111726	30.663863
FACUN ($k=4$)	54.119565	41.108899	30.641013
FACUN ($k=5$)	54.138193	41.084576	30.639943

Table 9 shows the number of distance calculations for several methods. Compared to FPNN, FACUN with $k=5$ can reduce the number of distance calculations by a factor of 107.1. Compared to PMLFPNN, FACUN with $k=5$ can reduce the number of distance calculations by 82.7%. Table 10 presents the mean square errors for various methods. Again, DLA has the highest MSE in general because it is not an exact version of PNN.

Here, we will show that using the set of cluster centers generated from our method as the initial codebook for GLA (k-means clustering) can obtain the better clustering results than GLA with randomly selected initial codebook. Table 11 shows the least mean square error of GLA with 100 runs and mean square error of FACUN using the data set from three images "Lena" "Baboon," and "Peppers," where FACUN uses the set of cluster centers generated by ACUN as the initial codebook for GLA. Table 11 does show that FACUN can obtain the better cluster results than GLA.

Table 7
Observed numbers of comparisons, steps, and number of distance calculations for ACUN using a data set obtained from three images "Lena," "Baboon," and "Peppers."

	Initial phase	Merged phase per iteration	Initial step of update phase per iteration	Final step of update phase per iteration	Total
Comparisons		24,703.5			1,207,902,336
Steps	4,859,720	4.4	7.4	5.3	5,385,233
Distance calculations	31,090,138	–	24,703.5	320,042.1	1,691,396,723

Table 8
Computing time (in seconds) of FPNN, DLA, PMLFPNN, ACUN, PMLACUN, and FACUN using the data set from three images "Lena" "Baboon," and "Peppers."

Method	M		
	128	256	512
FPNN	3192.36	3181.99	3177.69
DLA	61.59	61.47	61.74
PMLFPNN	844.88	847.30	844.48
ACUN ($k=3$)	637.81	637.72	636.20
ACUN ($k=4$)	584.78	579.38	580.30
ACUN ($k=5$)	868.56	867.38	867.53
PMLACUN ($k=3$)	477.53	480.94	477.73
PMLACUN ($k=4$)	467.81	467.20	469.67
PMLACUN ($k=5$)	477.09	477.01	477.86
FACUN ($k=3$)	179.03	179.33	179.34
FACUN ($k=4$)	171.06	171.23	170.91
FACUN ($k=5$)	172.06	171.53	171.59

Table 9
Number of distance calculations for FPNN, DLA, PMLFPNN, ACUN, PMLACUN, and FACUN using the data set from three images "Lena" "Baboon," and "Peppers."

Method	M		
	128	256	512
FPNN	8,438,439,497	8,438,341,654	8,437,940,378
DLA	90,228,949	90,231,206	90,232,345
PMLFPNN	705,179,411	705,138,294	704,976,312
ACUN ($k=3$)	2,530,004,165	2,529,964,210	2,529,792,736
ACUN ($k=4$)	2,019,714,874	2,019,680,538	2,019,531,790
ACUN ($k=5$)	1,691,550,851	1,691,522,539	1,691,396,723
PMLACUN ($k=3$)	302,501,436	302,478,133	302,379,816
PMLACUN PMLACUN ($k=4$)	273,023,792	273,001,577	272,909,047
PMLACUN ($k=5$)	254,187,482	254,165,517	254,078,745
FACUN ($k=3$)	90,894,076	90,879,424	90,832,663
FACUN ($k=4$)	83,402,468	83,389,570	83,340,897
FACUN ($k=5$)	78,796,877	78,783,133	78,734,317

Table 10
The mean square errors for FPNN, DLA, PMLFPNN, ACUN, PMLACUN, and FACUN using the data set from three images "Lena," "Baboon," and "Peppers."

Method	M		
	128	256	512
FPNN	147.710213	125.642842	105.700978
DLA	148.451855	125.848335	106.093845
PMLFPNN	147.916142	125.617498	105.802250
ACUN ($k=3$)	148.196827	125.869707	105.836351
ACUN ($k=4$)	147.742458	125.661177	105.780367
ACUN ($k=5$)	147.750681	125.779576	105.815815
PMLACUN ($k=3$)	147.645711	125.844655	105.980509
PMLACUN ($k=4$)	148.124799	125.868563	105.781754
PMLACUN ($k=5$)	148.205086	125.693862	105.881913
FACUN ($k=3$)	147.706021	125.546569	105.917910
FACUN ($k=4$)	147.656681	125.608787	105.985700
FACUN ($k=5$)	147.707543	125.758550	105.923524

Table 11
The least mean square error of GLA with 100 randomly selected initial codebooks and mean square error of MACUN using the data set from three images "Lena," "Baboon," and "Peppers."

Method	M		
	128	256	512
GLA with 100 runs	144.015528	122.699041	105.406689
MACUN	143.374579	121.545285	102.120814

Table 12
Computing time (in seconds) of dividing two data sets from a residual image and an original one into 256 clusters.

Method	Residual image	Original image
FPNN	46.09	25,165.78
DLA	7.84	176.94
PMLFPNN	15.25	12,298.94
ACUN ($k=3$)	34.03	15,693.80
ACUN ($k=4$)	31.31	15,691.67
ACUN ($k=5$)	30.14	14,706.11
PMLACUN ($k=3$)	17.17	6846.25
PMLACUN ($k=4$)	16.56	6412.45
PMLACUN ($k=5$)	17.39	7195.86
FACUN ($k=3$)	27.88	1501.91
FACUN ($k=4$)	24.69	1477.53
FACUN ($k=5$)	24.44	1502.88

Table 13
The mean square errors of dividing two data sets from a residual image and an original one into 256 clusters.

Method	Residual image	Original image
FPNN	5.313831	1.935127
DLA	5.404240	10.353935
PMLFPNN	5.277697	1.946243
ACUN ($k=3$)	5.269476	1.952223
ACUN ($k=4$)	5.274652	1.949495
ACUN ($k=5$)	5.266987	1.944594
PMLACUN ($k=3$)	5.274643	1.943778
PMLACUN ($k=4$)	5.275617	1.955851
PMLACUN ($k=5$)	5.260774	1.941510
FACUN ($k=3$)	5.278013	1.953878
FACUN ($k=4$)	5.276483	1.945353
FACUN ($k=5$)	5.268533	1.942222

4.4. Example 4: Data sets are generated from a residual image and an original one

Two data sets from [16,31] are used in this example. The first data set in this example consisting of 6480 data points with dimension 16 is generated from a residual image. This residual image of 360×288 pixels is obtained by subtracting a subsequent image frame from the previous one using the original video

sequence "Miss America." The second data set in this example having 262,144 data points with dimension 3 is obtained from the real image "Lena." Each data point in this data set consists of the R, G, and B components of a color.

Table 12 presents the computing time of FPNN, DLA, PMLFPNN, ACUN, PMLACUN, and FACUN. From Table 12, we can find that DLA has the least computing time for the residual and original images, although it is not the exact variant of PNN. Among exact variants of PNN, PMLFPNN with $k=4$ has the best performance, in terms of the computing time, for the residual image and FACUN with $k=4$ gives the least computing for the original image. Compared with FPNN, PMLACUN with $k=5$ can reduce the computing time by a factor of about 17.4 for the original image. Table 13 shows the mean square errors for six approaches. These methods except DLA give almost the same MSEs.

4.5. Example 5: Synthetic data sets with evenly distributed cluster centers

These data sets are generated by a method described in [10]. In this example, several data sets with $N=10{,}000$ and $d=400$ are generated, where N is the data size and d is the data dimension. These data sets are generated using seventy two cluster centers, which are evenly distributed over the hypercube $[-1, 1]^d$ with $d=400$. Several Gaussian distribution with standard deviations $\sigma=0.01$–0.30 along each coordinate are used to generate data points around each center, where each coordinate is generated independently. The degree of cluster separation in this example is related to the standard deviation σ of Gaussian distribution. The higher standard deviation indicates a worse cluster separation. Figs. 4 and 5 list the computing time and mean square errors, respectively, for six algorithms to merge 10,000 data points into 72 clusters. From Fig. 4, we can find that FACUN with $k=5$ has the less computing time than FPNN. In this example, the mean square errors of FPNN, PMLFPNN, ACUN, PMLACUN, and FACUN are the same. Therefore, just the mean square errors of FACUN and DLA with $k=10$ are presented in Fig. 5. Note here that the desired number of clusters cannot be obtained using DLA with $k<10$. From Fig. 5, we can find that the mean square error of DLA is significantly higher than that of FPNN, FACUN, or PMLACUN for the data set with low cluster separation.

To visualize the clustering result, another data set with $N=1000$ and $d=2$ is generated using ten clusters, which are evenly distributed over the hypercube $[-1, 1]^2$. A Gaussian distribution with standard deviation $\sigma=0.15$ along each coordinate is used to generate data points around each center for this data set. This data set is divided into 10 clusters using DLA and FACUN. Fig. 6 presents the clustering results. From Fig. 6, we can find that FACUN can obtain the better clustering result.

To perform the statistical test for our methods and several available approaches, two hundred data sets with $N=10{,}000$ and $d=100$ are generated using 72 evenly distributed cluster centers over the hypercube $[-1, 1]^{100}$. Several Gaussian distributions with standard deviations ranging from 0.01 to 0.21 along each coordinate are used to generate data points around each center for these data sets, where each coordinate is generated independently. It is noted that 40 data sets are generated for a given

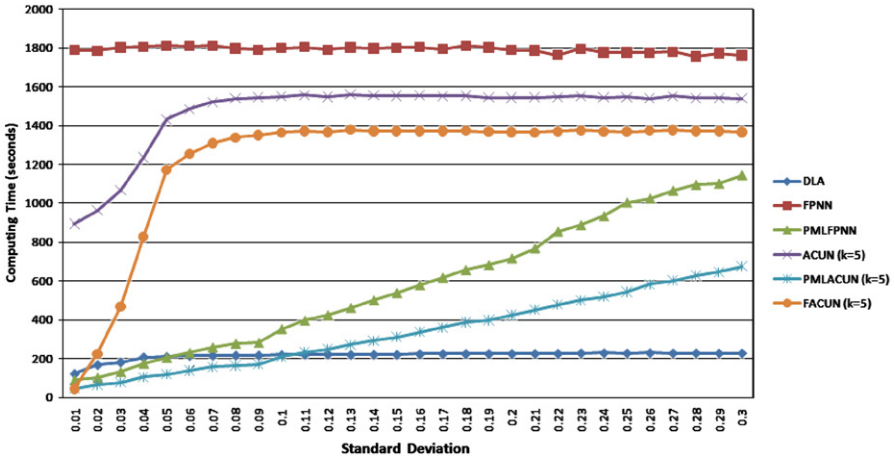

Fig. 4. Computing time for synthetic data sets with various standard deviations.

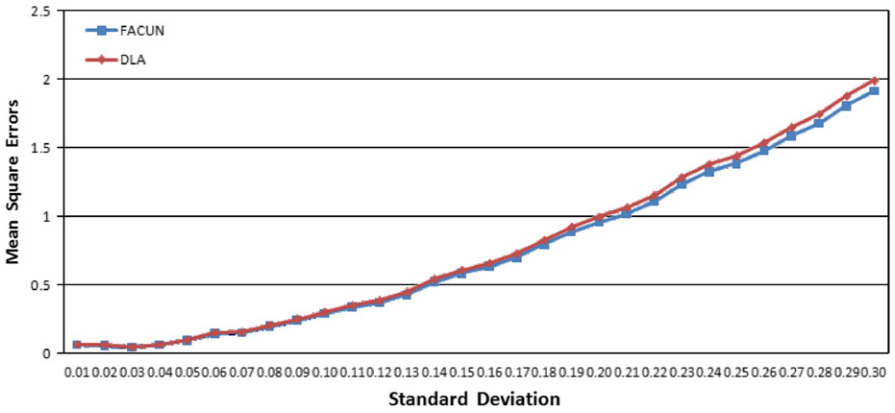

Fig. 5. Mean square errors for synthetic data sets with $N=10{,}000$, $d=400$, and various standard deviations.

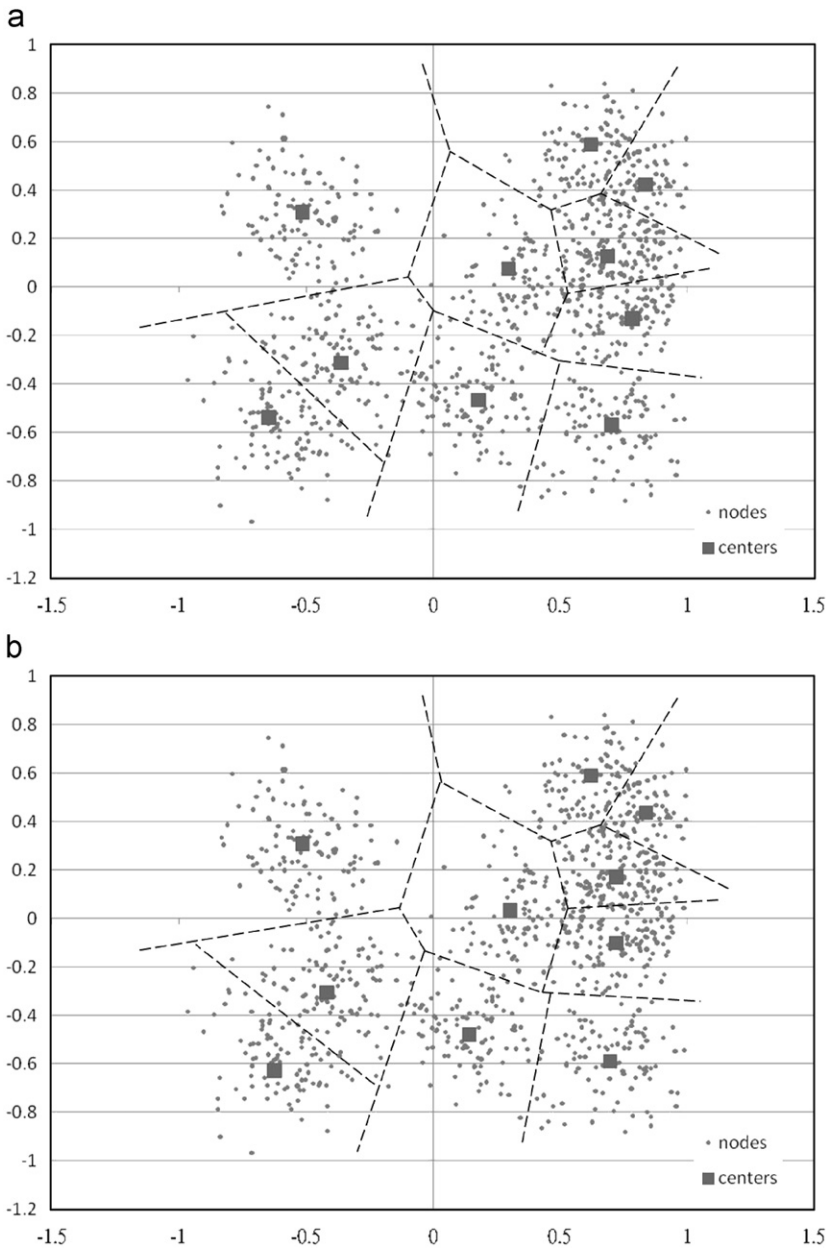

Fig. 6. The clustering results for (a) DLA and (b) FACUN using a synthetic data set with $N=1000$, $d=2$, and $\sigma=0.15$.

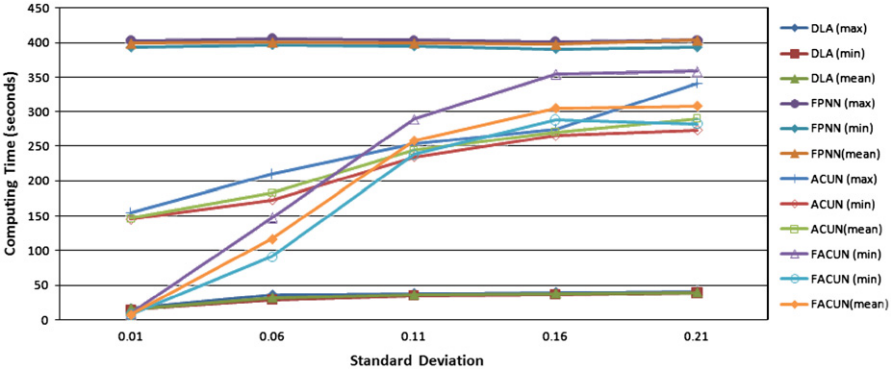

Fig. 7. The statistical information of computing time (in seconds) for FPNN, DLA, ACUN, and FACUN using 200 Synthetic data sets.

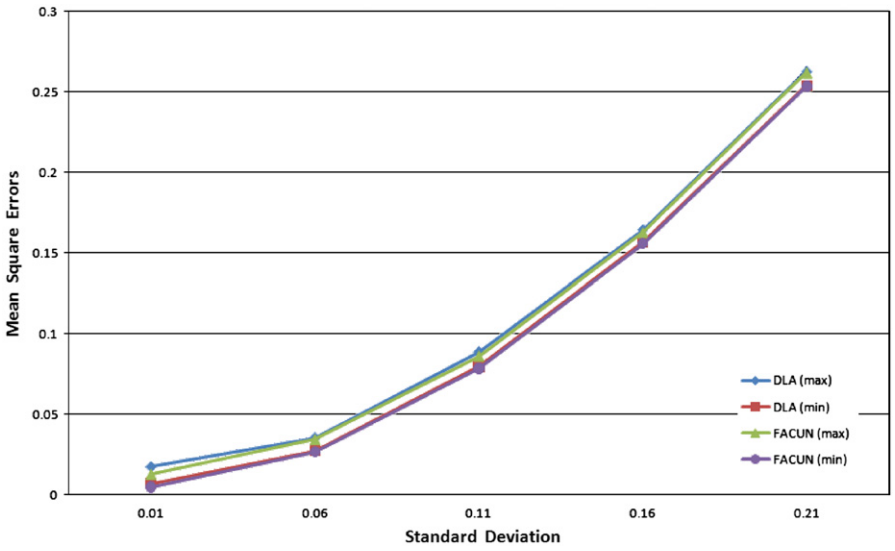

Fig. 8. The statistical information of mean square errors for DLA and FACUN using 200 synthetic data sets.

standard deviation. Fig. 7 shows the minimum and maximum computing time for FPNN, DLA ($k=10$), ACUN ($k=5$), and FACUN ($k=5$) to divide these two hundred data sets into 72 clusters. Since the mean square errors of FPNN, ACUN, and FACUN are almost the same, just the minimum and maximum mean square errors for DLA and FACUN are presented in Fig. 8. In this test, the computing time of our methods ACUN and FACUN is always less than FPNN and the difference of mean square errors for these three methods are always not significant. Compared to DLA, the probabilities of obtaining less computing time and mean square error for FACUN are 20% and 99.2%, respectively. From Fig. 7, we can find that FPNN has the longest computing time.

5. Conclusions

In this paper, we develop an algorithm to reduce the computational complexity of PNN. Our method can obtain the clustering results as good as those of PNN or FPNN, since our approach is an exact version of PNN. The computational complexity, in terms of the number of distance calculations, of our proposed method ACUN is $O(N^2)$, where N is the number of data points. The computational complexities of PNN and FPNN are $O(N^3)$ and $O(\tau N^2)$, respectively, with $\tau \ll N$. Compared to FPNN, our proposed method ACUN with $k=4$ can reduce the computing time by a factor of about 3.1 for the data set consisting of 16,384 data points from an original image "Lena." Compared with FPNN, our method FACUN with $k=5$ can reduce the computing time by a factor of about 26.8 for the same data set. For the data set obtained from a residual image, PMLFPNN has the less computing time than FACUN.

References

[1] A. Gersho, R.M. Gray, in: Vector Quantization and Signal Compression, Kluwer Academic Publishers, Boston, MA, 1991.
[2] Y.C. Liaw, J.Z.C. Lai, Winston Lo, Image restoration of compressed image using classified vector quantization, Pattern Recognition 35 (2002) 181–192.
[3] J. Foster, R.M. Gray, M.O. Dunham, Finite state vector quantization for waveform coding, IEEE Transactions on Information Theory 31 (1985) 348–359.
[4] J.Z.C. Lai, Y.C. Liaw, Winston Lo, Artifact reduction of JPEG coded images using mean-removed classified vector quantization, Signal Processing 82 (2002) 1375–1388.
[5] S. Theodoridis, K. Koutroumbas, in: Pattern Recognition, 2nd edition, Academic Press, New York, 2003.
[6] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, in: Advances in Knowledge Discovery and Data Mining, MIT Press, Boston, MA, 1996.
[7] D. Liu, F. Kubala, Online speaker clustering, in: Proceedings of IEEE Conference on Acoustic, Speech, and Signal Processing, 2004, pp. 333–336.
[8] P. Hojen-Sorensen, N. de Freitas, T. Fog, On-line probabilistic classification with particle filters, in: Proceedings of IEEE Signal Processing Society Workshop, 2000, pp. 386–395.
[9] M. Eirinaki, M. Vazirgiannis, Web mining for web personalization, ACM Transactions on Internet Technology 3 (2003) 1–27.
[10] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, A. Wu, An efficient k-means clustering algorithm: analysis and implementation, IEEE Transactions on PAMI 24 (2002) 881–892.
[11] R. Kashef, M.S. Kamel, Cooperative clustering, Pattern Recognition 43 (2010) 2315–2339.
[12] A. Gersho, R.M. Gray, in: Vector Quantization and Signal Compression, Kluwer Academic Publishers, Boston, MA, 1991.
[13] J.Z.C. Lai, C.C. Lue, Fast search algorithms for VQ codebook generation, Journal of Visual Communication and Image Representation 7 (1996) 163–168.
[14] J. Shanbehzadeh, P.O. Ogunbona, On the computational complexity of the LBG and PNN algorithm, IEEE Transactions on Image Processing 6 (1997) 614–616.
[15] P. Guénoche, Hansen, B. Jaumard, Efficient algorithms for divisive hierarchical clustering with the diameter criterion, Journal of Classification 1 (1991) 5–30.
[16] P. Fränti, O. Virmajoki, Ville Hautamäki, Fast agglomerative clustering using a k-nearest neighbor graph, IEEE Transactions on PAMI 26 (2006) 1875–1881.
[17] P. Fränti, T. Kaukoranta, D.F. Shen, K.S. Chang, Fast and memory efficient implementation of the exact PNN, IEEE Transactions on Image Processing 9 (2000) 773–777.
[18] J.H. Ward, Hierarchical grouping to optimize an objective function, Journal of the American Statistical Association 58 (1963) 238–244.
[19] T. Kurita, An efficient agglomerative clustering algorithm using a heap, Pattern Recognition 24 (1991) 205–209.
[20] T. Kaukoranta, P. Fränti, O. Nevalainen, Vector quantization by lazy pairwise nearest neighbor method, Optical Engineering 38 (1999) 1862–1868.
[21] O. Virmajoki, P. Fränti, T. Kaukoranta, Practical methods for speeding-up the pairwise nearest neighbor method, Optical Engineering 40 (2001) 2495–2504.
[22] C.-D. Bei, R.M. Gray, An improvement of the minimum distortion encoding algorithms for vector quantization and pattern matching, IEEE Transactions on Communications (1985) 1132–1133 COMM-33.
[23] J. Correa-Morris, D.L. Espinosa-Isidrón, D.R. Álvarez-Nadiozhin, An incremental nested partition method for data clustering, Pattern Recognition 43 (2010) 2439–2455.
[24] H.G. Ayad, M.S. Kamel, On voting-based consensus of cluster ensembles, Pattern Recognition 43 (2010) 1943–1953.
[25] S.-W. Ra, J.-K. Kim, A fast mean-distance-ordered partial codebook search algorithm for image vector quantization, IEEE Transactions on Circuits and Systems II 40 (1993) 576–579.

[26] Y.C. Liaw, Improvement of the fast exact pairwise-nearest-neighbor algorithm, Pattern Recognition 42 (2009) 867–870.
[27] J.Z.C. Lai, Y.C. Liaw, Julie Liu, Fast *k*-nearest-neighbor search based on projection and triangular inequality, Pattern Recognition 40 (2007) 351–359.
[28] W.H. Equitz, A new vector quantization clustering algorithm, IEEE Transactions on Acoustics, Speech, and Signal Processing 37 (1989) 1568–1575.
[29] J.Z.C. Lai, Y.C. Liaw, Fast-searching algorithm for vector quantization using projection and triangular inequality, IEEE Transactions on Image Processing 13 (2004) 1554–1558.
[30] J. McNames, A fast nearest-neighbor algorithm based on a principal axis search tree, IEEE Transactions on PAMI 23 (2001) 964–976.
[31] J.Z.C. Lai, J.Y. Yen, Inverse error-diffusion using classified vector quantization, IEEE Transactions on Image Processing 7 (1998) 1753–1758.

Chih-Tang Chang received his B.S. and M.S. degrees in Computer Science and Engineering from National Taiwan Ocean University, Taiwan, in 2001 and 2004, respectively. He is currently working toward the Ph.D. degree in the area of multimedia communication. His current interests are in data clustering, image processing, and multimedia.

Jim Z.C. Lai received his Ph.D. in Engineering from the University of California, Los Angeles in 1986. He then worked as a research engineer at GM and a lead project engineer at Rockwell International. He joined the Department of Information Engineering and Computer Science in 1988, Feng-Chia University as an associate professor. From 1994 to 2003, he was the full professor at the same department. Since 2004, he has been the full professor at the Department of Computer Science and Engineering, National Taiwan Ocean University. His current interests are in image processing, multimedia, and network security.

M.D. Jeng received his Ph.D. in Electrical, Computer & Systems Engineering Department, Rensselaer Polytechnic Institute in 1992. He is currently the full professor at the Department of Electrical Engineering, National Taiwan Ocean University. His current interests are in embedded system, automation, and wireless network.