

An Efficient k -Means Clustering Algorithm Using Simple Partitioning*

MING-CHUAN HUNG, JUNGPIN WU⁺, JIN-HUA CHANG
AND DON-LIN YANG

Department of Information Engineering and Computer Science

⁺*Department of Statistics*

Feng Chia University

Taichung, 407 Taiwan

The k -means algorithm is one of the most widely used methods to partition a dataset into groups of patterns. However, most k -means methods require expensive distance calculations of centroids to achieve convergence. In this paper, we present an efficient algorithm to implement a k -means clustering that produces clusters comparable to slower methods. In our algorithm, we partition the original dataset into blocks; each block unit, called a unit block (UB), contains at least one pattern. We can locate the centroid of a unit block (CUB) by using a simple calculation. All the computed CUBs form a reduced dataset that represents the original dataset. The reduced dataset is then used to compute the final centroid of the original dataset. We only need to examine each UB on the boundary of candidate clusters to find the closest final centroid for every pattern in the UB. In this way, we can dramatically reduce the time for calculating final converged centroids. In our experiments, this algorithm produces comparable clustering results as other k -means algorithms, but with much better performance.

Keywords: clustering, k -means algorithm, centroid, k - d tree, data mining

1. INTRODUCTION

Clustering is a process in which a group of unlabeled patterns are partitioned into a number of sets so that similar patterns are assigned to the same cluster, and dissimilar patterns are assigned to different clusters. There are two goals for a clustering algorithm: determining good clusters and doing so efficiently. Clustering has become a widely studied problem in a variety of application domains, such as in data mining and knowledge discovery [4, 19], statistical data analysis [5, 17], data classification and compression [6], medical image processing [22, 23, 29] and bioinformatics [25, 26, 30].

Several algorithms have been proposed in the literature for clustering [1, 7-16]. The k -means clustering algorithm is the most commonly used [7] because it can be easily implemented. However, employing the k -means method requires an execution time proportional to the product of the number of clusters and the number of patterns per iteration. This total execution time is computationally very expensive, especially for large datasets. Therefore, the k -means clustering algorithm cannot satisfy the need for

Received April 30, 2004; revised November 23, 2004; accepted February 3, 2005.

Communicated by Ding-Zhu Du.

* This work was supported by the National Science Council, Taiwan, under contract No. NSC 92-2213-E-035-039.

fast response time for some applications. How to reduce the computational time required to cluster a large dataset becomes an important operational objective. To solve this and other related performance problems, Alsabti *et al.* [1] proposed an algorithm based on the data structure of the k - d tree and used a pruning function on the candidate centroid of a cluster. While this method can reduce the number of distance calculations and the execution time, the time required to build the k - d tree structure is proportional to the size of the dataset. The total processing time is still too long when a large dataset is involved.

We propose an efficient algorithm for implementing the k -means method. It can produce comparable clustering results with much better performance by simplifying distance calculations and reducing total execution time. Our algorithm first partitions the dataset into several blocks of equal size, called Unit Blocks (UBs). Instead of using the k - d tree approach [17] to generate unit blocks, we use a simple middle point method to subdivide a dimension one at a time. Therefore, every unit block's range can be calculated much more quickly. This involves scanning the dataset twice to make sure which block a pattern is in and complete the process of assigning all patterns to related unit blocks. How many blocks to use is an important factor that directly relates to execution time? We will discuss the optimum arrangement of unit blocks for our algorithm.

The method we propose is more efficient than that of Alsabti. Alsabti's algorithm must scan the dataset each time the tree's node is partitioned into the subset of the next layer in order to calculate the scope of the subspace set of this node until the recursive execution of the maximal leaf size.

Our algorithm calculates the centroid of the dataset using the simplified data in each unit block, which is an object that contains at least one pattern, separately, after the completion of unit block partitioning. The centroid of a unit block approximates the information about its patterns, so we only need to use a few unit block centroids instead of calculating the centroid of the patterns of the original dataset. If we use the derived approximation of the centroid to calculate the division of clusters, we can increase the speed of convergence by reducing the number of calculations. The reduction in computational complexity makes this algorithm much more efficient and robust than the direct k -means and Alsabti's algorithms in clustering datasets, especially large ones.

The rest of this paper is organized as follows: section 2 surveys related work and describes the contributions and limitations of k -means clustering. We present our algorithm in section 3 and discuss the optimization for dividing unit blocks section 4. The performance analysis of our algorithm and a comparison of results with the direct k -means and Alsabti's algorithms on synthetic datasets are presented in section 5. Finally, our conclusions and recommendations for future research are presented in section 6.

2. RELATED WORK

There are many algorithms for clustering datasets. The k -means clustering is a popular method used to divide n patterns $\{x_1, \dots, x_n\}$ in d dimensional space into k clusters [7]. The result is a set of k centers, each of which is located at the centroid of the partitioned dataset. This algorithm can be summarized in the following steps:

- (1) Choose the number of clusters k and input a dataset of n patterns $X = \{x_1, \dots, x_n\}$. Randomly select the initial candidates for k cluster centers matrix $V^{(0)}$ from the dataset.
- (2) Assign each pattern to the nearest cluster using a distance measure. For each pattern x_i , compute its membership $m(C_j | x_i)$ in each cluster C_j . The membership function $m(C_j | x_i)$ defines the proportion of pattern x_i that belongs to the j th cluster C_j . The k -means algorithm uses a hard membership function, that is the membership $m(C_j | x_i) \in \{0, 1\}$. If the pattern x_i is closest to cluster C_j (i.e., the distance between x_i and cluster center v_j is minimal), then $m(C_j | x_i) = 1$; otherwise $m(C_j | x_i) = 0$.
- (3) Recompute the centroids (centers) of these k clusters to find new cluster centers v_j , and compute the sum of square error E .

$$v_j = \frac{\sum_{i=1}^n m(C_j | x_i) x_i}{\sum_{i=1}^n m(C_j | x_i)} \quad \text{for } j = 1, \dots, k. \quad (1)$$

$$E = \sum_{j=1}^k \sum_{x_i \in C_j} \|x_i - v_j\|^2 \quad \text{for } i = 1, \dots, n; j = 1, \dots, k. \quad (2)$$

- (4) Repeat steps 2 and 3 until convergence. Typical convergence criteria are: no more reassignment of patterns to new clusters, the change in error function E falls below a threshold, or a predetermined number of iterations have been reached.

To choose a proper number of clusters k is a domain dependent problem. To resolve this, some have proposed methods to perform k -clustering for various numbers of clusters and employ certain criteria for selecting the most suitable value of k [3, 18].

Several variants of the k -means algorithm have been proposed. Their purpose is to improve efficiency or find better clusters, mostly the former. Improved efficiency is usually accomplished by either reducing the number of iterations to reach final convergence or reducing the total number of distance calculations.

In the first step, the algorithm randomly selects k initial cluster centers from the original dataset. Then, in the later steps, the algorithm will converge to the actual cluster centers after several iterations that can vary in a wide range from a few to several thousand. Therefore, choosing a good set of initial cluster centers is very important for the algorithm. However, it is difficult to select a good set of initial cluster centers randomly. Bradley and Fayyad have proposed an algorithm for refining the initial cluster centers. Not only are the true clusters found more often, but the clustering algorithm also iterates fewer times [31]. Our proposed method uses a simple process to obtain initial cluster centers from a simplified dataset. These centers are very close to the actual cluster centers of the original dataset, and so only a few iterations are needed for convergence.

Some clustering methods improve performance by reducing the distance calculations. For example, Judd *et al.* proposed a parallel clustering algorithm P-CLUSTER [11] which uses three pruning techniques. These include the assignment of cluster centroids, the maximum movement of patterns at each iteration, and the maintenance of partial

sums for centroids. Alsabti *et al.* [1] proposed an algorithm based on the data structure of the k - d tree and used a pruning function on the candidate centroid of a cluster. Although the distance calculation is done only at the internal nodes, the time to build and traverse the k - d tree can be time consuming. Kanungo *et al.* [28] proposed a filtering algorithm which begins by storing the data points in a k - d tree. For each node of the tree maintaining a set of candidate centers, they will be pruned, or filtered as they are propagated to the node's children. Kanungo *et al.*'s implementation of the filtering algorithm is more robust than Alsabti's, because Alsabti's method relies on a less effective pruning mechanism based on computing the minimum and maximum distances to each cell.

3. OUR PROPOSED ALGORITHM

We propose an enhanced version of the k -means algorithm with simple partitioning to speed up the time in finding the final converged centroids. Figs. 5, 6, 7 and 8 depict the pseudo-code of our algorithm. Table 1 shows the notation used in describing the algorithm. Our algorithm is composed of three parts, as explained in the following subsections.

Table 1. Notation used in our algorithm.

u	The number of unit blocks that contain at least one pattern
α	The number of unit blocks on the boundary
k	The number of clusters
k'	The average number of clusters on the boundary
n	The number of patterns in a dataset
x_i	The i th data element (pattern)
d	The number of dimensions
v_j	The centroid of the j th cluster
C_j	The j th cluster
UB_a	The a th unit block
BUNB	Unit block not on the boundary
BUB	Unit block on the boundary
CUB	Centroid of unit block
LSUB	Linear sum of unit blocks
WUB	Weight of unit block

3.1 Partitioning the Dataset into Unit Blocks

At the beginning of the algorithm, the dataset is partitioned into several blocks of equal size. Instead of using the k - d tree approach [17], we employ a simple method that does not require more than two scans of the dataset.

Similar to Alsabti's partition method for finding splitting points, for two dimensional data we determine the minimum and maximum values of the data along each di-

mension. This is the first scan of the data set. These values determine a rectangle that bounds all the dataset patterns.

Next, we partition this data space into equally sized blocks. Unlike in k - d tree partition, the midpoint approach is used to recursively divide the splitting dimensions into equal parts. We simply choose a fixed number of splits to produce a specified total number of blocks. This can save some computation time. In our empirical study, we found that 11 splits resulted in near optimal performance for the datasets with a random distribution; these results are discussed in section 4.

After the partitioning, we locate all the blocks that contain at least one pattern and call them Unit Blocks (UB s) as shown in Fig. 1. To find out which UB a pattern belongs to, a second scan of the dataset is performed.

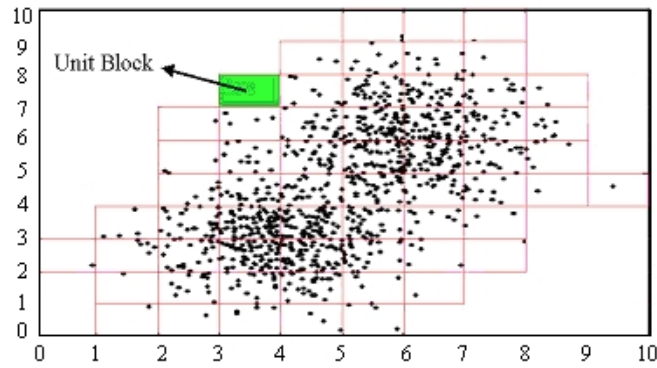


Fig. 1. Partitioning of the original dataset into unit blocks.

3.2 Calculating Centroids of UB s to form a Simplified View of the Dataset

To form a simplified view of the dataset, we use UB s to represent the original patterns in order to improve performance. Each UB can be represented by its centroid; a simplified view is then constructed by using these centroids of unit blocks (CUB s). To compute a CUB , the following steps are used in the second scanning of the dataset mentioned above:

- $LSUB$: For each unit block UB_a we calculate the linear sums of each dimension d individually.

$$LSUB_a = \sum_{x_i \in UB_a} x_i; a \in \{1, \dots, u\}. \quad (3)$$

- WUB : The weight of unit block is the total number of patterns in the unit block. We simply count the elements during the database scan. $|UB_a|$ denotes the number of patterns in the a^{th} unit block.

$$WUB_a = |UB_a|; a \in \{1, \dots, u\}. \tag{4}$$

- *CUB*: The centroid of a unit block can be easily calculated since we have *LSUB* and *WUB*. Since there is at least one pattern in each UB_a , the CUB_a is

$$CUB_a = \frac{LSUB_a}{WUB_a}; a \in \{1, \dots, u\}. \tag{5}$$

Fig. 2 illustrates the results of finding *CUBs* for four *UBs* and how they form a reduced dataset.

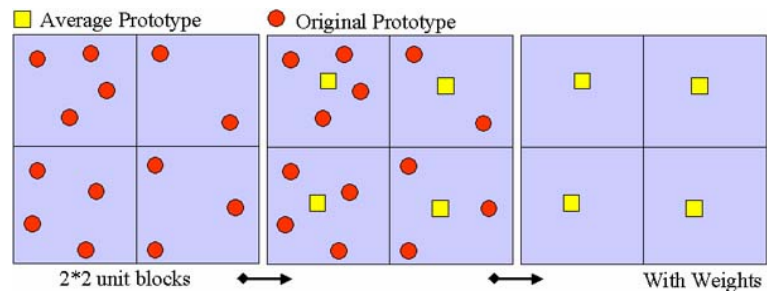


Fig. 2. Calculate *CUBs* for four *UBs* to form a reduced dataset.

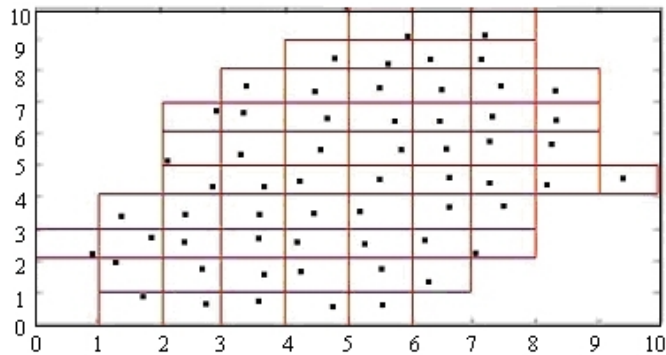


Fig. 3. The reduced dataset.

3.3 Calculating Final Cluster Centers

Fig. 3 is an illustration of the reduced dataset for Fig. 1. The *CUBs* are used as patterns to compute the final centroids; the steps are as follows:

- (1) Randomly select k *CUBs* from the reduced dataset as the initial cluster centers
To simplify the process, randomly select the first k centroids from the reduced data-

set just like the direct k -means clustering method and Alsabti's method. This provides the initial set of k candidate centroids.

- (2) Calculate new k centroids from the reduced dataset

Calculate the Euclidean distance l between each CUB_a and initial centroid with each dimension m to find the closest cluster.

$$l = \sqrt{\sum_{m=1}^d (CUB_a - v_j)^2}. \quad (6)$$

This results in the partitioning of the simplified view into k clusters. New centroids are re-calculated for these k clusters, and the process of finding the nearest centroid continues until the error function E does not change significantly.

$$E = \sum_{j=1}^k \left(\sum_{CUB_a \in C_j} WUB_a \|CUB_a - v_j\|^2 \right). \quad (7)$$

Since the reduced dataset is used in this step, the time needed to calculate centroids is greatly reduced. The larger the UB size is, the less time is needed. It is even possible that if the reduced dataset can fit into available memory, then disk IOs are not even needed.

However, using the simplified view of the dataset alone to compute the new centroids cannot guarantee that they are actual cluster centers for the original dataset. The reason is that any CUB on a cluster boundary cannot represent all patterns in the UB when calculating their final centroids. Fig. 4 shows such a situation where every pattern in the boundary blocks must be considered in the calculation of final centroids.

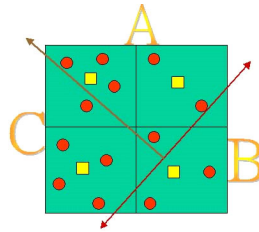


Fig. 4. For a unit block on the boundary, each pattern of the original dataset must be used to calculate the nearest centroid.

To solve this problem, we use both the simplified view and the original dataset to calculate final centroids. The following are modified steps for calculating centroids:

- (a) For each UB , determine if it is on a cluster boundary. In other words, we need to determine if its CUB is close to two or more clusters. If the difference between the dis-

tance measures of the *CUB* to the nearest centroid and the *CUB* to the second nearest centroid is less than the diagonal length of the *UB*, then such a *UB* is called a Unit Block on the Boundary or *BUB*. Otherwise, it is called *BUNB*.

- (i) If a *UB* is on the cluster boundary, calculate the nearest cluster centroid for each pattern in the *UB* from the original dataset. For each pattern x_i in the *BUB*, we only calculate the distance between pattern x_i with the neighboring candidate centroids that share the boundary, then assign x_i to the nearest cluster C_j and update the centroid's statistics for C_j .
 - (ii) If it is not on the cluster boundary, use the previous procedure to calculate the nearest centroid. Since *LSUB* and *WUB* are available for each *UB*, the computation is short and simple.
- (b) If the change in the following error function E , modified from Eq. (7), falls below a threshold or the predetermined number of iterations has been reached, then terminate the procedure; else go to step (a).

$$E = \sum_{j=1}^k \left(\sum_{(CUB_a \succ BUNB) \in C_j} WUB_a \|CUB_a - v_j\|^2 + \sum_{(x_i \succ BUB) \in C_j} \|x_i - v_j\|^2 \right). \quad (8)$$

Figs. 5-8 show pseudo-code for our simple partitioning (SP) k -means algorithm.

4. ANALYSIS OF UNIT BLOCK PARTITION

First, the proposed algorithm divides the dataset into several identical unit blocks. Then, it calculates the centroids and related statistics of patterns in each unit block to represent an approximation of the information in the unit blocks. We use this reduced data to reduce the overall time for distance calculations. We find that the clustering efficiency is closely related to determining how many blocks should be partitioned. This problem was investigated in the following.

There are n patterns in the dataset and k clusters were specified, and we use our algorithm to implement the k -means method. The performance from our experiments reveals some variation when the numbers of unit blocks increase or decrease, resulting in the following phenomenon:

- (1) When number of unit blocks increases, we find the number of unit blocks on cluster boundaries also increases, while the average number of patterns in unit blocks on the cluster boundary decreases. Therefore, the number of distance calculations that determines which patterns in the boundary blocks belong to which clusters will decrease with an increase in the number of unit blocks.
- (2) Conversely, when the number of unit blocks decreases, the number of unit blocks on the cluster boundary decreases as well, but the average number of patterns in unit blocks on the cluster boundary increases. Since the number of patterns belonging to these clusters increases, the number of distance calculations also increases, as shown in Fig. 9.

```

Proc SP  $k$ -means main ()
/* Partition the dataset into several unit blocks and obtain the simplified dataset */
Input a proper splitting number to generate  $u$  unit blocks in the dataset
partition_dataset (original dataset)
/* Randomly select the initial candidate for  $k$  cluster centers */
Initialize  $k$  cluster centers ( $v_1, \dots, v_k$ ) from the simplified dataset such that
 $v_j = CUB_a, a \in \{1, \dots, u\}, j \in \{1, \dots, k\}$ 
Each cluster  $C_j$  is associated with a cluster center (centroid)  $v_j$ 
/* Calculate the next  $k$  centroids from the simplified dataset */
Calcu_Cen_for_Simp_Dataset()
/* Calculate final  $k$  centroids using the modified process */
repeat
for each  $CUB_a$ , where  $a \in \{1, \dots, u\}$ , do
if  $UB_a$  is on the cluster boundary which is called  $BUB$ 
(i.e.  $(\|CUB_a - v_{j^*}\| - \|CUB_a - v_{j^{**}}\|) < L_a$ ; where  $L_a$  is the diagonal length of  $UB_a$ )
then
for each  $x_i \in BUB$ , do
find the cluster  $C_j$  with the nearest cluster center  $v_{j^*}$  in the clusters
( $C_{j^*}, C_{j^{**}}$ ) with the common cluster boundary
(i.e.  $\|x_i - v_{j^*}\| < \|x_i - v_{j^{**}}\|; j^*, j^{**} \in \{1, \dots, k\}$  and  $j^* \neq j^{**}$ )
assign  $x_i$  to  $C_{j^*}$ 
update the centroid's statistics for  $C_{j^*}$ 
else
 $UB_a$  is not on the cluster boundary which is called  $BUNB$ 
find the cluster  $C_{j^*}$  with the nearest cluster center  $v_{j^*}$ 
(i.e.  $\|CUB_a - v_{j^*}\| \leq \|CUB_a - v_j\|; j, j^* \in \{1, \dots, k\}$ )
assign  $LSUB_a$  and  $WUB_a$  to  $C_{j^*}$ 
update the centroid's statistics for  $C_{j^*}$ 
end if
for each cluster  $C_j$ , where  $j \in \{1, \dots, k\}$ , do
compute the cluster center  $v_j$  to be the centroid of all  $BUNBs$  and each  $x_i \in BUB$ 
in the cluster  $C_j$ 

$$v_j = \frac{\sum_{(CUB_a \succ BUNB) \in C_j} LSUB_a + \sum_{(x_i \succ BUB) \in C_j} x_i}{\sum_{(CUB_a \succ BUNB) \in C_j} WUB_a + |(x_i \succ BUB) \in C_j|}$$

/* where  $|(x_i \succ BUB) \in C_j|$  denotes the number of patterns in the BUB with cluster  $C_j$  */
Compute the error function:

$$E = \sum_{j=1}^k \left( \sum_{(CUB_a \succ BUNB) \in C_j} WUB_a \|CUB_a - v_j\|^2 + \sum_{(x_i \succ BUB) \in C_j} \|x_i - v_j\|^2 \right)$$

until  $E$  does not change significantly or the predetermined number of iterations has been reached
End main

```

Fig. 5. The proposed k -means algorithm with simple partitioning.

```

Proc partition_dataset (original dataset)
  for each dimension  $D_m$  of the dataset, where  $m \in \{1, \dots, d\}$ , do
    /* find the range for dimension  $D_m$  */
    Range_Max[m] = the maximum value of dimension  $D_m$ 
    Range_Min[m] = the minimum value of dimension  $D_m$ 
    /* calculate the number of segments it would partition for dimension  $D_m$  */
    /* Num_of_Split[m] is the number of splits for dimension  $D_m$  */
    Num_of_Seg[m] = (Range_Max[m] - Range_Min[m])/Num_of_Split[m]
  for each data pattern  $x_i$ 
    /* calculate the UB that data element  $x_i$  belongs to */
    for each dimension  $D_m$  of  $x_i$ , named  $x_i[m]$ , do
      Point_in_Dim[m] =  $(x_i[m] - Range_Min[m])/Num_of_Seg[m]$ 
      /* Use the value of Point_in_Dim[m] to calculate the  $UB_a$  that  $x_i$  belongs to, named
      UB_Location of  $x_i$ , where  $a \in \{1, \dots, u\}$  */
      /* calculate LSUB and WUB for each  $UB_a$  */
      UB_process ( $x_i$ , UB_Location)
    for each  $UB_a$ , do
      /* compute CUB: Centroid of Unit Block */
      
$$CUB_a = \frac{LSUB_a}{WUB_a}$$

  End partition_dataset

```

Fig. 6. Partition a dataset.

```

Proc Calcu_Cen_for_Simp_Dataset ()
/* Calculate cluster centers (centroids) */
  repeat
    for each  $CUB_a$ , where  $a \in \{1, \dots, u\}$ , do
      find the cluster  $C_{j^*}$  with the nearest cluster center  $v_{j^*}$ 
      (i.e.  $\|CUB_a - v_{j^*}\| \leq \|CUB_a - v_j\|$ ;  $j, j^* \in \{1, \dots, k\}$ )
      assign  $LSUB_a$  and  $WUB_a$  to  $C_{j^*}$ 
      update the cluster center's statistics for  $C_{j^*}$ 
    for each cluster  $C_j$ , do
      compute the cluster center  $v_j$  to be the centroid of all unit blocks currently in the
      cluster  $C_j$ 
      
$$v_j = \frac{\sum_{CUB_a \in C_j} LSUB_a}{\sum_{CUB_a \in C_j} WUB_a}$$

      Compute the error function:
      
$$E = \sum_{j=1}^k \left( \sum_{CUB_a \in C_j} WUB_a \|CUB_a - v_j\|^2 \right)$$

    until  $E$  does not change significantly
  End Calcu_Cen_for_Simp_Dataset

```

Fig. 7. Calculate centroids for the reduced dataset.

```

Proc UB_process ( $x_i, a$ )
  /* compute  $LSUB$ : Linear Sum of Unit Block */
   $LSUB_a = LSUB_a + x_i$ 
  /* assign  $WUB$ : Weight of Unit Block */
   $WUB_a = WUB_a + 1$ 
End UB_process

```

Fig. 8. Process Unit Blocks.

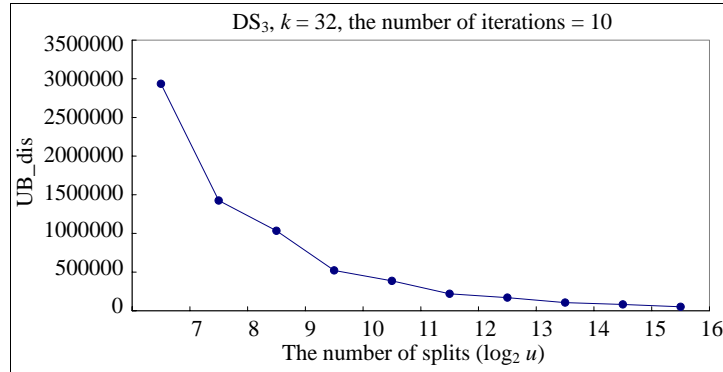


Fig. 9. The relationship between the number of distance calculations (UB_dis) for all patterns on the cluster boundary and the number of unit blocks.

The curve in Fig. 9 represents the model $y = 57912x^2 - 1581766x + 10746945$ which was calculated from the experimental data and gives the total number of distance calculations (UB_dis). The number of splits is $x = \log_2 u$, where u is the number of unit blocks. We used the DS₃ dataset that is depicted in section 5, set $k = 32$, and set the maximum number of iterations at 10 as an example.

The number of unit blocks not on the cluster boundary increases as the total number of unit blocks increases. How do we decide if those blocks not on the cluster boundary? Just like our proposed algorithm described in the previous section, we found the closest centroids, then set all patterns in this unit block to belong to the nearest cluster. The number of distance calculations in one iteration can be represented by the product of the number of unit blocks not on a cluster boundary and the number of clusters k . Consequently, the number of distance calculations will increase with an increase in the number of unit blocks. The number of distance calculations from the model is (DT_dis) $y = 110055x^2 - 2067684x + 9456495$, which was derived from experimental dataset DS₃ with the number of clusters $k = 32$ and the number of iterations = 10. This model is shown in Fig. 10.

Therefore, the process of clustering in our algorithm involves two cases regarding the unit blocks. They are the blocks on the cluster boundary and the blocks not on the boundary. The total number of distance calculations is the sum of both kinds of blocks (Total_dis = UB_dis + DT_dis). Our model was derived from experimental dataset DS₃

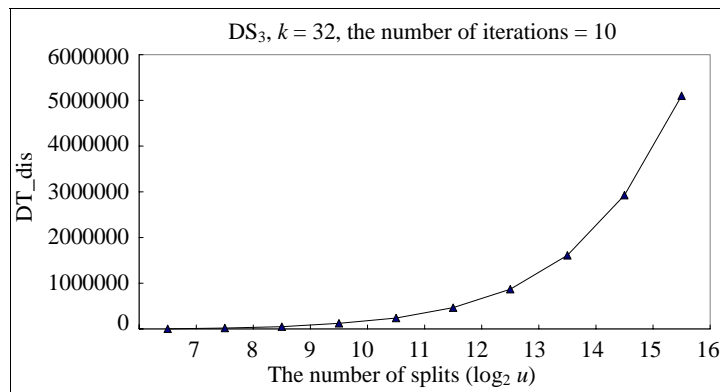


Fig. 10. The relationship between the number of distance calculations (DT_dis) of all CUBs, which were not on the cluster boundary, and the number of unit blocks.

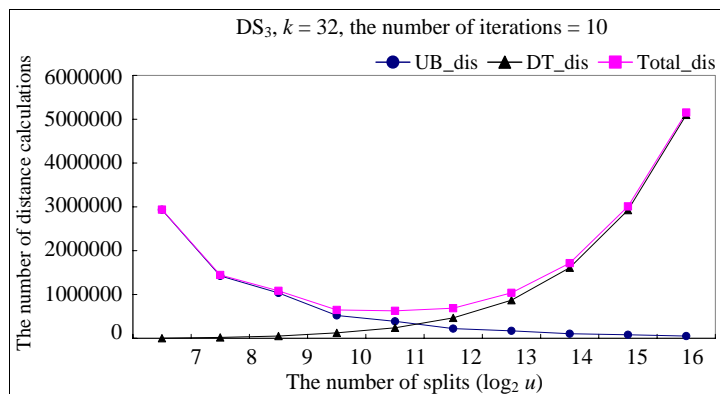


Fig. 11. The relationship between the total number of distance calculations (Total_dis) and the number of unit blocks.

with the number of clusters $k = 32$ and the number of iterations = 10. In this model, the total number of distance calculations (Total_dis) is $y = 167967x^2 - 3649450x + 20203440$; its outcome is shown in Fig. 11. The best number of splits is 11, which corresponds to the lowest point of the Total_dis curve. The number of unit blocks is 2048 in this case.

Now, we consider the selection of the total number of unit blocks to get the best result with respect to the number of clusters k . For simplification, we chose $k = 8$ to 64 for analysis and tried to find a *proper* number of unit blocks for each k . The results of our experiment show the smallest Total_dis appears at the lowest points of the curves as shown in Fig. 12. Assume that there are n patterns, each of dimension d , and the number of clusters is k . When we let the first differential of the model equation Total_dis with respect to the number of unit blocks be equal to zero, we can find the *proper* number of

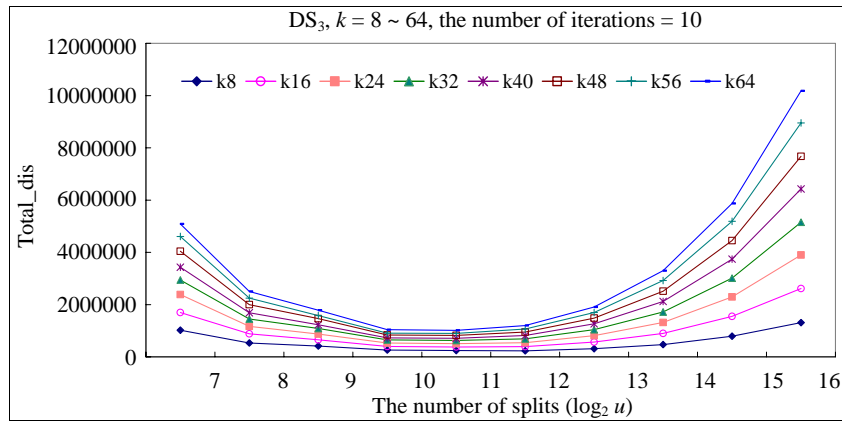


Fig. 12. The relationship between the number of distance calculations and the number of unit blocks with different k .

Table 2. Best number of blocks for different numbers of clusters.

Datasets	Clusters (k) Blocks	Clusters (k)							
		8	16	24	32	40	48	56	64
DS ₁	U'	1350	1330	1310	1290	1270	1250	1220	1200
	U_{proper}	1024	1024	1024	1024	1024	1024	1024	1024
DS ₂	U'	5970	5950	5910	5980	5870	5840	5810	5780
	U_{proper}	4096	4096	4096	4096	4096	4096	4096	4096
DS ₃	U'	2560	2530	2490	2460	2430	2390	2360	2320
	U_{proper}	2048	2048	2048	2048	2048	2048	2048	2048

unit blocks. The results of our model equation match the observed results of our experimental data. It verifies that our proposed model for calculating the *proper* number of unit blocks works well. More experimental results are given below.

We used three different datasets DS₁, DS₂ and DS₃ [6] for our study. The *proper* number of unit blocks for different values of k is given in Table 2. U' is the *proper* number of unit blocks derived using the models formulated from the experimental data. U_{proper} is the actual number of the proper blocks as observed in our study. We found that U' approximates U_{proper} . This provides evidence that our proposed method approaches the best result we can find.

From Table 2, the *proper* number of unit blocks is obviously not related to the specified number of clusters. Using dataset DS₃ for example, no matter what k is, the *proper* number of unit blocks is 2048. When k is 32, the *proper* numbers of unit blocks for the datasets DS₁, DS₂ and DS₃ are 1024, 4096 and 2048, respectively, which are related to the distributions of the datasets.

5. EXPERIMENTAL RESULTS AND DISCUSSION

In the previous sections we gave a systematic analysis, illustrated the methodology, and described the procedures and merits of our proposed algorithm. We showed that it is more efficient than the direct k -means algorithm or Alsabti's algorithm. The following is a discussion of the time complexity of our algorithm:

- (1) By partitioning the dataset into several unit blocks and using the centroid of a unit block to represent the patterns within the block, we know the time complexity of this procedure is $O(2n)$. The following details are noted:
 - (a) The time complexity of scanning a dataset to find the range of values for each dimension is $O(n)$.
 - (b) The time complexity of using a binary partition to divide a dataset into an optimized number of blocks is $O(\log u)$.
 - (c) For determining patterns, calculating the centroid in every unit block which contains at least one pattern, and computing statistical information for every unit block, the time complexity is $O(n)$.
- (2) The time required to assign the patterns to their closest clusters for each iteration is $O((n/u)\alpha k'd) + O((u - \alpha)kd)$.
- (3) The time required to calculate the candidate centroid is $O(k)$.
- (4) The time required to calculate the error function is $O(((n/u)\alpha + (u - \alpha))d)$.

The complexity of a direct k -means algorithm per iteration can be decomposed into three parts [7]:

- (1) The time required to assign the patterns to their closest clusters is $O(nkd)$.
- (2) The time required to calculate the candidate centroid is $O(nd)$.
- (3) The time required to calculate the error function is $O(nd)$.

The time complexity of the other methods usually depends on their proposed filtering mechanisms [1, 28]. Our proposed approach is much better than the direct k -means with a complexity of $O(n(k + 2)d)$ for each iteration.

To test and verify that our algorithm is more efficient than the direct k -means and Alsabti's algorithms, we used several synthetically generated datasets. This was done to study the scaling properties of our algorithm for different values of n and k . We used the k - d tree [1] in terms of the total execution time and the total number of distance calculations.

The experiments were performed on an Ultra enterprise 10000 running the Solaris. The clock speed of the processor was 250 MHz and the memory size was 4G bytes. The following datasets were used:

- Dataset DS_1 has a uniform distribution of the grid and its pattern size is 100,000; the original number of clusters is 100 and the dimension is 2 [6].
- Dataset DS_2 has a special distribution of the sine shape and its size of patterns, the original number of clusters, and dimension sizes are the same as DS_1 .

- Dataset DS_3 has a random distribution and its size of patterns, the original number of clusters, and dimension sizes are the same as DS_1 .
- Datasets R_1 to R_{12} have random distributions but their pattern sizes, original number of clusters, and dimensions are different from each other [1]. These 12 datasets are described in Table 3.

Table 3. Description of the datasets R_1 to R_{12} .

Dataset	Size of Patterns	Dimensionality	No. of Clusters
R_1	128K	2	16
R_2	256K	2	16
R_3	128K	2	128
R_4	256K	2	128
R_5	128K	4	16
R_6	256K	4	16
R_7	128K	4	128
R_8	256K	4	128
R_9	128K	6	16
R_{10}	256K	6	16
R_{11}	128K	6	128
R_{12}	256K	6	128

As mentioned in the previous section, the number of splits determines the number of unit blocks when partitioning a dataset. We have designed some experiments to determine what the total number of splits was for optimal performance. The results are as follows.

In addition to the above fifteen datasets, we increased the size of R_1 by 2 and 3 times to generate more test datasets. For each experiment, the total number of unit blocks ranged from 128 to at most half of the dataset size. The numbers of clusters (or centroids) are 16, 24, 32, 40, 48, 56 and 64. In the case of Fig. 13, ten iterations for convergence (y -axis) were performed for each of the splitting numbers (x -axis) ranging from 7 to 16. Note that 7 splits can produce $2^7 = 128$ blocks, and so on. One hundred was the maximum number of iterations allowed in our studies. We found that, on the average, the performance was optimal when the total number of blocks was 2048 for dataset R_1 , i.e., 11 splits. Dataset R_1 is similar to dataset DS_3 with a random distribution and its *proper* number of unit blocks is 2048 as shown in Table 2.

Therefore, we used a proper block number to partition the dataset in the experiments that have been described in section 4. For the three different k -means algorithms, all used the same initial candidate centroids to perform clustering operations. Three different comparisons were performed: the total execution time, the total number of distance calculations, and the number of iterations required before convergence. The results are described below.

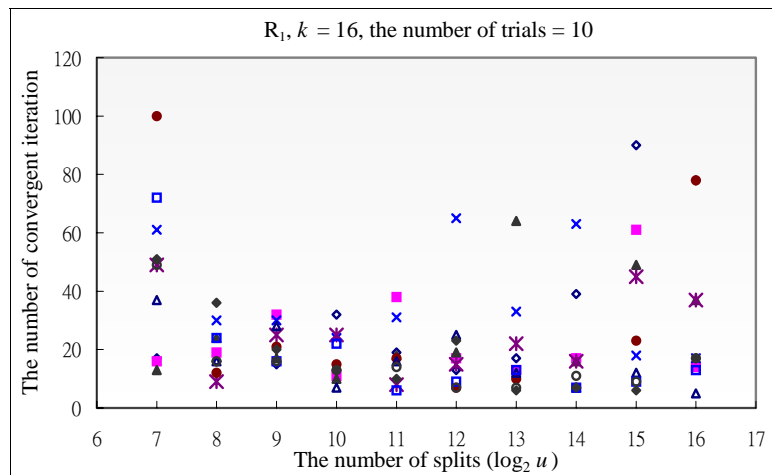


Fig. 13. Ten different numbers of convergent iterations (vertical axis) were performed for each of the splitting numbers (horizontal axis) ranging from 7 to 16 for Dataset R_1 .

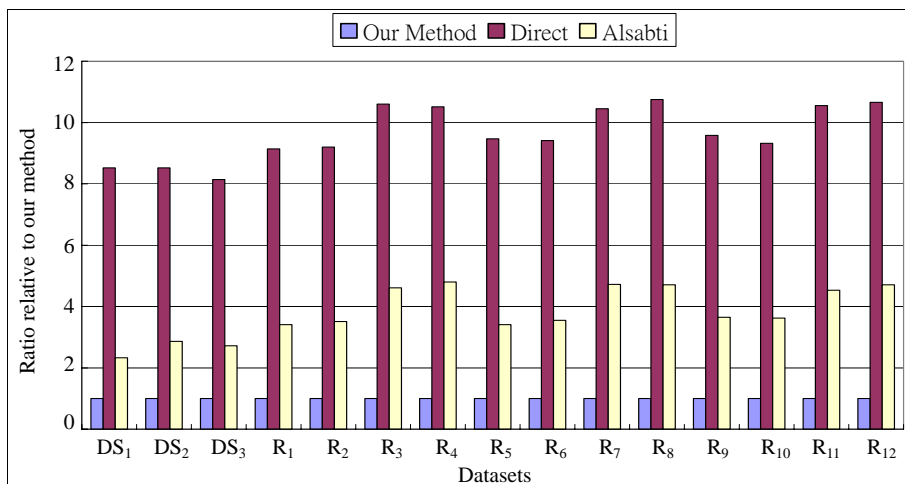


Fig. 14. Ratios of execution time for computing 16 centroids after 10 iterations of calculation.

Fig. 14 shows ratios of execution time for computing 16 clusters after 10 iterations. Compared to the other two methods, the performance of our algorithm was two to more than 10 times faster. Similar results were found for calculating 32, 64, and more clusters. Fig. 15 compares the distance calculations for computing 16 clusters. Other experiments for calculating 32 and 64 clusters have resulted in similar improvements.

Finally, Fig. 16 compares the numbers of convergent iterations for 16 clusters. The performance of our algorithm was five to 10 times better. When the maximum number of iterations was set to 100, we could see that the direct k -means algorithm and Alsabti's

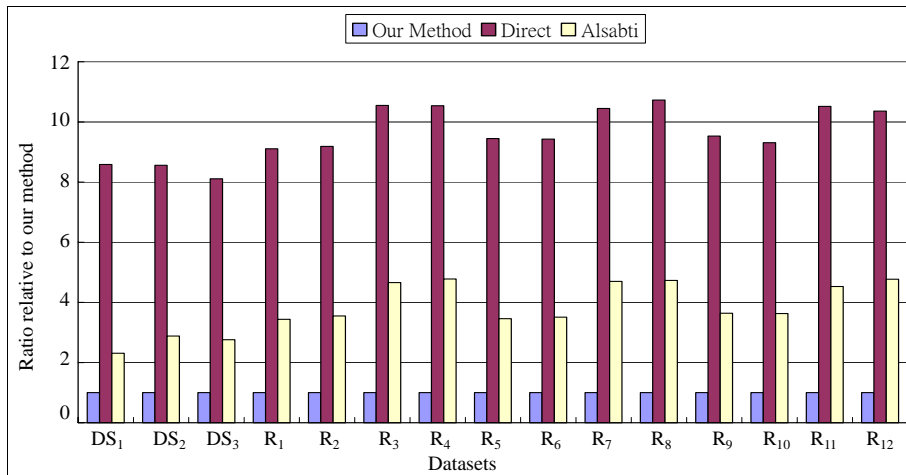


Fig. 15. Ratios of numbers of distance calculations for calculating 16 centroids.

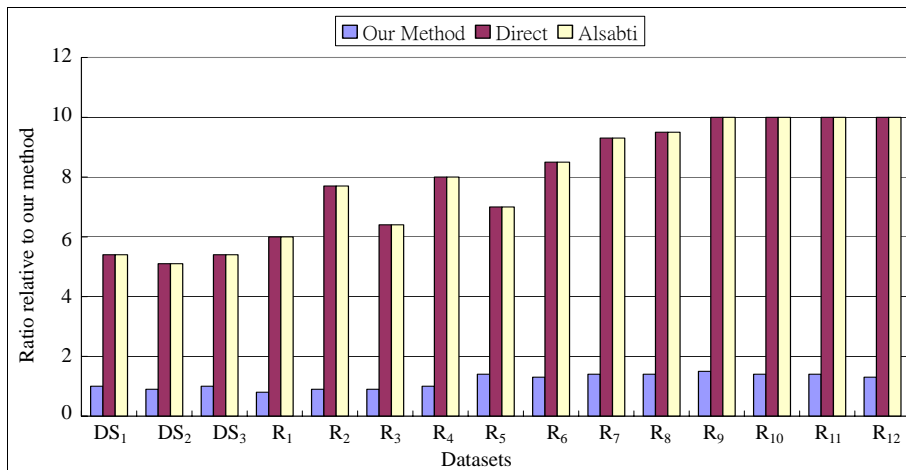


Fig. 16. Number of iterations for convergence for calculating 16 centroids.

algorithm stopped executing before reaching convergence for some datasets, thereby affecting performance. But our algorithm always converged within 20 iterations. So, the execution performance of our algorithm is much better.

6. CONCLUSIONS

In this paper, we have presented an efficient clustering algorithm based on the *k*-means method. We partitioned datasets into several blocks and used reduced versions of the datasets to compute final cluster centroids. The results of our experiments clearly indicate that our algorithm converges quickly. When compared to two other commonly

used clustering algorithms, the direct k -means and Alsabti's algorithms, in terms of total execution time, the number of distance calculations, and the efficiency for clustering, our algorithm was superior.

In the analysis of time complexity, the observed data from our studies proves that our algorithm spends much less time in clustering than the direct k -means method. This is indeed good news to those who develop computationally complex applications with large datasets where response time is critical.

We also discussed the issue of the proper choice for the number of unit blocks. We found that it is independent of the number of clusters but is related to the distribution of the dataset. Presently, we use binary splitting to partition the dataset into unit blocks, but other partition methods will be the subject of further study especially for high dimensional datasets.

We now use the direct k -means method to choose k patterns as the initial candidate centroids in a random process. We all knew that the number of iterations to achieve cluster centroids was influenced by the initial candidate cluster centroids using random choice. In other words, with good luck, if the first chosen centroids were near the actual centroids, we would require fewer iterations to achieve convergence. Conversely, we need more iterations to achieve the k centroids, sometimes even spending the maximum number of iterations will not achieve convergence. We have also found in our algorithm that an initial cluster centroid selected from a unit block with a higher density is closer to the actual cluster centroid. We may select k patterns as the initial candidate centroids from the unit blocks with a higher pattern density, thus obtaining a better performance. This efficiency in the normal distribution dataset or dense dataset is especially obvious. It would appear that our algorithm would inherently achieve more efficient performance. This is also the scope of further research.

REFERENCES

1. K. Alsabti, S. Ranka, and V. Singh, "An efficient k -means clustering algorithm," in *Proceedings of 1st Workshop on High performance Data Mining*, 1998.
2. D. L. Yang, J. H. Chang, M. C. Hung, and J. S. Liu, "An efficient k -means-based clustering algorithm," in *Proceedings of 1st Asia-Pacific Conference on Intelligent Agent Technology*, 1999, pp. 269-273.
3. Y. M. Cheung, " k^* -means: a new generalized k -means clustering algorithm," *Pattern Recognition Letters*, Vol. 24, 2003, pp. 2883-2893.
4. Z. Huang, "Extensions to the k -means algorithm for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, Vol. 2, 1998, pp. 283-304.
5. J. Banfield and A. Raftery, "Model-based gaussian and non-gaussian clustering," *Biometrics*, Vol. 49, 1993, pp. 15-34.
6. T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: a new data clustering algorithm and its applications," *Data Mining and Knowledge Discovery*, Vol. 1, 1997, pp. 141-182.
7. R. C. Dubes and A. K. Jain, *Algorithms for Clustering Data*, Prentice Hall, 1988.
8. J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Density-based clustering in spatial

- databases: the algorithm GDBSCAN and its applications,” *Data Mining and Knowledge Discovery*, Vol. 2, 1998, pp. 169-194.
9. J. Vesanto and E. Alhoniemi, “Clustering of the self-organizing map,” *IEEE Transactions on Neural Networks*, Vol. 11, 2000, pp. 586-600.
 10. J. Gracia, J. Fdez-Valdivia, F. Cortijo, and R. Molina, “Dynamic approach for clustering data,” *Signal Processing*, Vol. 44, 1995, pp. 181-196.
 11. D. Judd, P. McKinley, and A. Jain, “Large-scale parallel data clustering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, 1998, pp. 871-876.
 12. L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley & Sons, 1990.
 13. R. T. Ng and J. Han, “Efficient and effective clustering methods for spatial data mining,” in *Proceedings of 20th International Conference on Very Large Databases*, 1994, pp. 144-155.
 14. E. Schikuta, “Grid clustering: an efficient hierarchical clustering method for very large data sets,” in *Proceedings of 13th International Conference on Pattern Recognition*, 1996, pp. 101-105.
 15. S. Guha, R. Rastogi, and K. Shim, “CURE: an efficient clustering algorithm for large database,” in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998, pp. 73-84.
 16. A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: a review,” *ACM Computing Surveys*, Vol. 31, 1999, pp. 264-323.
 17. J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, Vol. 18, 1975, pp. 509-517.
 18. G. W. Milligan and M. C. Cooper, “An examination of procedures for determining the number of clusters in a data set,” *Psychometrika*, Vol. 50, 1985, pp. 159-179.
 19. J. R. Wen, J. Y. Nie, and H. J. Zhang, “Query clustering using user logs,” *ACM Transactions on Information Systems*, Vol. 20, 2002, pp. 59-81.
 20. T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu, “The analysis of a simple k -means clustering algorithm,” in *Proceedings of 16th Annual Symposium of Computational Geometry*, 2000, pp. 100-109.
 21. J. M. Pena, J. A. Lozano, and P. Larranaga, “An empirical comparison of four initialization methods for the K -means algorithm,” *Pattern Recognition Letters*, Vol. 20, 1999, pp. 1027-1040.
 22. M. Singh, P. Patel, D. Khosla, and T. Kim, “Segmentation of functional MRI by k -means clustering,” *IEEE Transactions on Nuclear Science*, Vol. 43, 1996, pp. 2030-2036.
 23. C. W. Chen, J. Luo, and K. J. Parker, “Image segmentation via adaptive k -mean clustering and knowledge-based morphological operations with biomedical applications,” *IEEE Transactions on Image Processing*, Vol. 7, 1998, pp. 1673-1683.
 24. V. Makarenkov and P. Legendre, “Optimal variable weighting for ultrametric and additive trees and K -means partitioning: Methods and software,” *Journal of Classification*, Vol. 18, 2001, pp. 245-271.
 25. G. Sherlock, “Analysis of large-scale gene expression data,” *Current Opinion in Immunology*, Vol. 12, 2000, pp. 201-205.

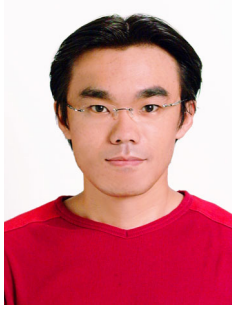
26. T. Burr, J. R. Gattiker, and G. S. Laberge, "Genetic subtyping using cluster analysis," *ACM SIGKDD Explorations*, Vol. 3, 2001, pp. 33-42.
27. V. Ramasubramanian and K. Paliwai, "Fast K -dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding," *IEEE Transactions on Signal Processing*, Vol. 40, 1992, pp. 518-531.
28. T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k -means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, 2002, pp. 881-892.
29. D. A. Clausi, " K -means iterative fisher unsupervised clustering algorithm applied to image texture segmentation," *Pattern Recognition*, Vol. 35, 2002, pp. 1959-1972.
30. F. X. Wu, W. J. Zhang, and A. L. Kusalik, "Determination of the minimum samples size in microarray experiments to cluster genes using K -means clustering," in *Proceedings of 3rd IEEE Symposium on Bioinformatics and Bioengineering*, 2003, pp. 401-406.
31. P. S. Bradley and U. M. Fayyad, "Refining initial points for k -means clustering," in *Proceedings of 15th International Conference on Machine Learning*, 1998, pp. 91-99.



Ming-Chuan Hung (洪明傳) received the B.E. degree in Industrial Engineering and the M.S. degree in Automatic Control Engineering from Feng Chia University, Taiwan, in 1979 and 1985, respectively. He is now a Ph.D. candidate in the Department of Information Engineering and Computer Science at Feng Chia University. From 1985 to 1987, he was an instructor in the Mechanics Engineering Department at National Chin-Yi Institute of Technology. Since 1987, he has been an instructor in the Industrial Engineering Department at Feng Chia University and served as a secretary in the College of Engineering from 1991 to 1996. His research interests include data mining, CIM, and e-commerce applications. He is a member of the CIIE.



Jungpin Wu (吳榮彬) received the B.S. degree in Applied Mathematics from Tatung University, Taiwan in 1988, the M.S. degree in Statistics from the Graduate Institute of Statistics of National Central University, Taiwan in 1993, and the Ph.D. degree in Statistics from the North Carolina State University in 1998. He was a postdoctoral staff at Academia Sinica from 1998 to 1999. Since then, he joined the faculty of Feng Chia University, where he was an Assistant Professor in the Department of Statistics from 1999 to 2004. Dr. Wu is currently an Associate Professor. His research interests include spatial statistics, generalized estimating equations, empirical process approach, and data mining.



Chin-Hua Chang (張金華) received the B.E. degree in Computer Science from Feng Chia University, Taiwan in 1998 and the M.S. degree in Computer Science from Feng Chia University in 2000. His research interest is data mining.



Don-Lin Yang (楊東麟) received the B.E. degree in Computer Science from Feng Chia University, Taiwan in 1973, the M.S. degree in Applied Science from the College of William and Mary in 1979, and the Ph.D. degree in Computer Science from the University of Virginia in 1985. He was a staff programmer at IBM Santa Teresa Laboratory from 1985 to 1987 and a member of the technical staff at AT&T Bell Laboratories from 1987 to 1991. Since then, he joined the faculty of Feng Chia University, where he was in charge of the University Computer Center from 1993 to 1997 and served as the Chairperson of the Department of Information Engineering and Computer Science from 2001 to 2003. Dr. Yang is currently a professor and the head of the Information Technology Office at Feng Chia University. His research interests include distributed and parallel computing, image processing, and data mining. He is a member of the IEEE computer society and the ACM.