Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

An efficient enhanced *k*-means clustering algorithm

FAHIM A.M.[†1], SALEM A.M.[2], TORKEY F.A.[3], RAMADAN M.A.[4]

([1]*Department of Mathematics, Faculty of Education, Suez Canal University, Suez city, Egypt*)
([2]*Department of Computer Science, Faculty of Computers & Information, Ain Shams University, Cairo city, Egypt*)
([3]*Department of Computer Science, Faculty of Computers & Information, Minufiya University, Shbeen El Koom City, Egypt*)
([4]*Department of Mathematics, Faculty of Science, Minufiya University, Shbeen El Koom City, Egypt*)
[†]E-mail: ahmmedfahim@yahoo.com

Received Mar. 15, 2006; revision accepted May 11, 2006

Abstract: In *k*-means clustering, we are given a set of *n* data points in *d*-dimensional space $\mathbb{R}^d$ and an integer *k* and the problem is to determine a set of *k* points in $\mathbb{R}^d$, called centers, so as to minimize the mean squared distance from each data point to its nearest center. In this paper, we present a simple and efficient clustering algorithm based on the *k*-means algorithm, which we call enhanced *k*-means algorithm. This algorithm is easy to implement, requiring a simple data structure to keep some information in each iteration to be used in the next iteration. Our experimental results demonstrated that our scheme can improve the computational speed of the *k*-means algorithm by the magnitude in the total number of distance calculations and the overall time of computation.

Key words: Clustering algorithms, Cluster analysis, *k*-means algorithm, Data analysis
doi:10.1631/jzus.2006.A1626 **Document code:** A **CLC number:** TP301.6

INTRODUCTION

The huge amount of data collected and stored in databases increases the need for effective analysis methods to use the information contained implicitly there. One of the primary data analysis tasks is cluster analysis, intended to help a user understand the natural grouping or structure in a dataset. Therefore, the development of improved clustering algorithms has received much attention. The goal of a clustering algorithm is to group the objects of a database into a set of meaningful subclasses (Ankerst *et al*., 1999).

Clustering is the process of partitioning or grouping a given set of patterns into disjoint clusters. This is done such that patterns in the same cluster are alike, and patterns belonging to two different clusters are different. Clustering has been a widely studied problem in a variety of application domains including data mining and knowledge discovery (Fayyad *et al*., 1996), data compression and vector quantization (Gersho and Gray, 1992), pattern recognition and pattern classification (Duda and Hart, 1973), neural networks, artificial intelligence, and statistics.

Existing clustering algorithms can be broadly classified into hierarchical and partitioning clustering algorithms (Jain and Dubes, 1988). Hierarchical algorithms decompose a database *D* of *n* objects into several levels of nested partitionings (clusterings), represented by a dendrogram, i.e., a tree that iteratively splits *D* into smaller subsets until each subset consists of only one object. There are two types of hierarchical algorithms; an agglomerative that builds the tree from the leaf nodes up, whereas a divisive builds the tree from the top down. Partitioning algorithms construct a single partition of a database *D* of *n* objects into a set of *k* clusters, such that the objects in a cluster are more similar to each other than to objects in different clusters.

The Single-Link method is a commonly used hierarchical clustering method (Sibson, 1973). Starting by placing every object in a unique cluster, in every step the two closest clusters are merged, until all points are in one cluster. Another approach to hierarchical clustering is the algorithm CURE proposed in (Guha *et al*., 1998). This algorithm stops the creation of a cluster hierarchy, if a level consists of k clusters, where k is one of several input parameters. It utilizes multiple representative points to evaluate the distance between clusters, thereby adjusting well to arbitrary shaped clusters and avoiding the chain effect.

Optimization based partitioning algorithms typically represent clusters by a prototype. Objects are assigned to the cluster represented by the most similar prototype. An iterative control strategy is used to optimize the whole clustering such that, the average or squared distances of objects to its prototypes are minimized. These clustering algorithms are effective in determining a good clustering, if the clusters are of convex shape, similar size and density, and if their number k can be reasonably estimated. Depending on the kind of prototypes, one can distinguish k-means, k-modes and k-medoid algorithms.

In k-means algorithm (MacQueen, 1967), the prototype, called the center, is the mean value of all objects belonging to a cluster. The k-modes algorithm (Huang, 1997) extends the k-means paradigm to categorical domains. For k-medoid algorithms (Kaufman and Rousseeuw, 1990), the prototype, called the "medoid", is the most centrally located object of a cluster. The algorithm CLARANS, introduced in (Ng and Han, 1994), is an improved k-medoid type algorithm restricting the huge search space by using two additional user-supplied parameters. It is significantly more efficient than the well-known k-medoid algorithms PAM and CLARA, presented in (Kaufman and Rousseeuw, 1990).

Density-based approaches, apply a local cluster criterion, are very popular for database mining. Clusters, are regions in the data space where the objects are dense, and separated by regions of low object density (noise). These regions may have an arbitrary shape. A density-based clustering method is presented in (Ester *et al*., 1996). The basic idea of the algorithm DBSCAN is that, for each point of a cluster, the neighborhood of a given radius (e), has to contain at least a minimum number of points (*MinPts*), where e and *MinPts* are input parameters. Another density-based approach is WaveCluster (Sheikholeslami *et al*., 1998), which applies wavelet transform to the feature space. It can detect arbitrary shape clusters at different scales. In (Hinneburg and Keim, 1998) the density-based algorithm DenClue is proposed. This algorithm uses a grid but is very efficient, because it only keeps information about grid cells that actually contain data points, and manages these cells in a tree-based access structure. This algorithm generalizes some other clustering approaches which, however, results in a large number of input parameters.

Also the density and grid-based clustering technique CLIQUE (Agrawal *et al*., 1998) has been proposed for mining in high-dimensional data spaces. Input parameters are the size of the grid and a global density threshold for clusters. The major difference from all other clustering approaches is that, this method also detects subspaces of the highest dimensionality such that high-density clusters exist in those subspaces.

Another approach to clustering is the BIRCH method (Zhang *et al*., 1996) that constructs a CF-tree, which is a hierarchical data structure designed for a multiphase clustering method. First, the database is scanned to build an initial in memory CF-tree. Second, an arbitrary clustering algorithm can be used to cluster the leaf nodes of the CF-tree.

Among clustering formulations that are based on minimizing a formal objective function, perhaps the most widely used and studied is k-means clustering. Given a set of n data points in real d-dimensional space, $\mathbb{R}^d$, and an integer k, the problem is to determine a set of k points in $\mathbb{R}^d$, called centers, so as to minimize the mean squared distance from each data point to its nearest center.

Recently iterated local search (ILS) was proposed in (Merz, 2003). This algorithm tries to find near optimal solution for criteria of the k-means algorithm. It applies k-means algorithm, tries to swap randomly chosen center with randomly chosen point, compares the current solution with the previous and keeps the best solution. This process is repeated a fixed number of times.

The k-means method has been shown to be effective in producing good clustering results for many practical applications. However, the k-means algorithm requires time proportional to the product of number of patterns and number of clusters per iteration. This computationally may be expensive especially for large datasets. We propose an efficient implementation method for implementing the k-means method. Our algorithm produces the same clustering results as that of the *k*-means algorithm, and has significantly superior performance than the *k*-means algorithm.

The rest of this paper is organized as follows. We review the *k*-means algorithms in Section 2, present the proposed algorithm in Section 3, and describe the datasets used to evaluate the algorithm in Section 4, where we describe the experimental results and conclude with Section 5.

k-MEANS CLUSTERING

In this section, we briefly describe the *k*-means algorithm. Suppose that a dataset of n data points x_1, x_2, ..., x_n such that each data point is in $\mathbb{R}^d$, the problem of finding the minimum variance clustering of the dataset into k clusters is that of finding k points $\{m_j\}$ (j=1, 2, …, k) in $\mathbb{R}^d$ such that

$$\frac{1}{n}\sum_{i=1}^{n}\left[\min_j d^2(x_i, m_j)\right] \tag{1}$$

is minimized, where $d(x_i, m_j)$ denotes the Euclidean distance between x_i and m_j. The points $\{m_j\}$ (j=1, 2, …, k) are known as cluster centroids. The problem in Eq.(1) is to find k cluster centroids, such that the average squared Euclidean distance (mean squared error, MSE) between a data point and its nearest cluster centroid is minimized.

The *k*-means algorithm provides an easy method to implement approximate solution to Eq.(1). The reasons for the popularity of *k*-means are ease and simplicity of implementation, scalability, speed of convergence and adaptability to sparse data.

The *k*-means algorithm can be thought of as a gradient descent procedure, which begins at starting cluster centroids, and iteratively updates these centroids to decrease the objective function in Eq.(1). The *k*-means always converge to a local minimum. The particular local minimum found depends on the starting cluster centroids. The problem of finding the global minimum is NP-complete. The *k*-means algorithm updates cluster centroids till local minimum is found. Fig.1 shows the *k*-means algorithm.

Before the *k*-means algorithm converges, distance and centroid calculations are done while loops are executed a number of times, say l, where the positive integer l is known as the number of *k*-means iterations. The precise value of l varies depending on the initial starting cluster centroids even on the same dataset. So the computational time complexity of the algorithm is $O(nkl)$, where n is the total number of objects in the dataset, k is the required number of clusters we identified and l is the number of iterations, $k \le n$, $l \le n$.

```
1 $MSE$=largenumber;
2 Select initial cluster centroids $\{m_j\}_j^k=1$;
3 Do
4     $OldMSE=MSE$;
5     $MSE1=0$;
6     For $j$=1 to $k$
7         $m_j=0$; $n_j=0$;
8     endfor
9     For $i$=1 to $n$
10        For $j$=1 to $k$
11            Compute squared Euclidean
                  distance $d^2(x_i, m_j)$;
12        endfor
13        Find the closest centroid $m_j$ to $x_i$;
14        $m_j=m_j+x_i$; $n_j=n_j+1$;
15        $MSE1=MSE1+d^2(x_i, m_j)$;
16    endfor
17    For $j$=1 to $k$
18        $n_j=\max(n_j, 1)$; $m_j=m_j/n_j$;
19    endfor
20    $MSE=MSE1$;
  while ($MSE<OldMSE$)
```

Fig.1 *k*-means algorithm

AN EFFICIENT ENHANCED *k*-MEANS CLUSTERING

In this section, we introduce the proposed idea that makes *k*-means more efficient, especially for dataset containing large number of clusters. Since, in each iteration, the *k*-means algorithm computes the distances between data point and all centers, this is computationally very expensive especially for huge datasets. Why we do not benefit from previous iteration of *k*-means algorithm? For each data point, we can keep the distance to the nearest cluster. At the next iteration, we compute the distance to the previous nearest cluster. If the new distance is less than or equal to the previous distance, the point stays in its cluster, and there is no need to compute its distances to the other cluster centers. This saves the time required to compute distances to $k-1$ cluster centers.

The proposed idea comes from the fact that the *k*-means algorithm discovers spherical shaped cluster, whose center is the gravity center of points in that cluster, this center moves as new points are added to or removed from it. This motion makes the center closer to some points and far apart from the other points, the points that become closer to the center will stay in that cluster, so there is no need to find its distances to other cluster centers. The points far apart from the center may change the cluster, so only for these points their distances to other cluster centers will be calculated, and assigned to the nearest center. Fig.2 explains the idea.

Fig.2a represents the dataset points and the initial 3 centroids. Fig.2b shows points distribution over the initial 3 centroids, and the new centroids for the next iteration. Fig.2c shows the final clusters and their centroids.

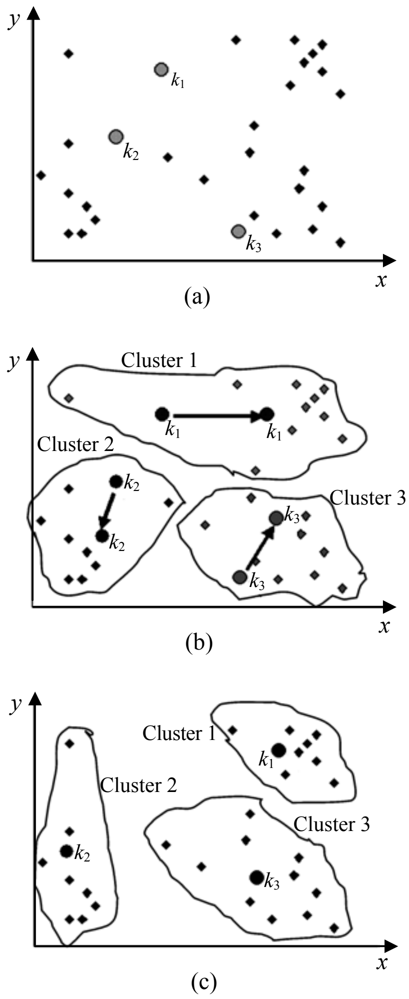

Fig.2 Some points remain in their cluster because the center becomes closer to it. (a) Initial centroids to a dataset; (b) Recalculating the position of the centroids; (c) Final positions of the centroids

When we examine Fig.2b, in Clusters 1, 2 we note that, the most points become closer to their new center, only one point in Cluster 1, and 2 points in Cluster 2 will be redistributed (their distances to all centroids must be computed), and the final clusters are presented in Fig.2c. Based on this idea, the proposed algorithm saves a lot of time.

In the proposed method, we write two functions. The first function is the basic function of the *k*-means algorithm, that finds the nearest center for each data point, by computing the distances to the *k* centers, and for each data point keeps its distance to the nearest center.

The first function is shown in Fig.3, which is similar to that in Fig.1, with adding a simple data structure to keep the distance between each point and its nearest cluster. This function is called *distance*().

```
Function distance()
//assign each point to its nearest cluster
1  For $i$=1 to $n$
2    For $j$=1 to $k$
3      Compute squared Euclidean distance
         $d^2(x_i, m_j)$;
4    endfor
5    Find the closest centroid $m_j$ to $x_i$;
6    $m_j=m_j+x_i$; $n_j=n_j+1$;
7    $MSE=MSE+d^2(x_i, m_j)$;
8    $Clusterid[i]$=number of the closest centroid;
9    $Pointdis[i]$=Euclidean distance to the closest
           centroid;
10 endfor
11 For $j$=1 to $k$
12   $m_j=m_j/n_j$;
13 endfor
```

Fig.3 The first function used in our implementation of *k*-means algorithm

In Line 3 the function finds the distance between point number *i* and all *k* centroids. Line 5 searches for the closest centroid to point number *i*, say the closest centroid is number *j*. Line 6 adds point number *i* to cluster number *j*, and increase the count of points in cluster *j* by one. Lines 8 and 9 are used to enable us to execute the proposed idea; these two lines keep the number of the closest cluster and the distance to the closest cluster. Line 12 does centroids recalculation.

The other function is shown in Fig.4, which is the same as Fig.3 and is called *distance_new*(). Line 1 finds the distance between the current point *i* and the new cluster center assigned to it in the previous iteration, if the computed distance is smaller than or equal to the distance to the old center, the point stays in its cluster that was assigned to in previous iteration, and there is no need to compute the distances to the other $k-1$ centers. Lines 3~5 will be executed if the computed distance is larger than the distance to the old center, this is because the point may change its cluster, so Line 4 computes the distance between the current point and all *k* centers. Line 6 searches for the closest center, Line 7 assigns the current point to the closest cluster and increases the count of points in this cluster by one, Line 8 updates mean squared error. Lines 9 and 10 keep the cluster id, for the current point assigned to it, and its distance to it to be used in next call of that function (i.e. next iteration of that function). This information is kept in Line 9 and Line 10 allows this function to reduce the distance calculation required to assign each point to the closest cluster, and this makes the function faster than the function distance in Fig.3.

```
Function distance_new()
//assign each point to its nearest cluster
1  For $i$=1 to $n$
       Compute squared Euclidean distance
          $d^2(x_i, Clusterid[i])$;
       If ($d^2(x_i, Clusterid[i])$<=$Pointdis[i]$)
          Point stay in its cluster;
2      Else
3         For $j$=1 to $k$
4            Compute squared Euclidean distance
               $d^2(x_i, m_j)$;
5         endfor
6      Find the closest centroid $m_j$ to $x_i$;
7      $m_j=m_j+x_i$; $n_j=n_j+1$;
8      $MSE=MSE+d^2(x_i, m_j)$;
9      $Clustered[i]$=number of the closest centroid;
10     $Pointdis[i]$=Euclidean distance to the closest
                 centroid;
11 endfor
12 For $j$=1 to $k$
13     $m_j=m_j/n_j$;
14 endfor
```

Fig.4 The second function used in our implementation of *k*-means algorithm

Our implementation of *k*-means algorithm is based up on these two functions described in Figs.3 and 4. We have two implementations of the *k*-means algorithm. In the first implementation, these two functions are executed a number of times (first *distance*() then *distance_new*(), i.e., there is overlap between these two functions). This implementation will be referred to as "overlapped *k*-means" algorithm.

In the second implementation, function *distance*() is executed two times, while function *distance_new*() is executed the reminder of iteration. This implementation will be referred to as "enhanced *k*-means" algorithm.

Complexity

As we discussed before, the *k*-means algorithm converges to local minimum. Before the *k*-means converges, the centroids computed number of times, and all points are assigned to their nearest centroids, i.e., complete redistribution of points according to new centroids, this takes $O(nkl)$, where *n* is the number of points, *k* is the number of clusters and *l* is the number of iterations.

In the proposed enhanced *k*-means algorithm, to obtain initial clusters, this process requires $O(nk)$. Here, some points remain in its cluster, the others move to another cluster. If the point stays in its cluster this require $O(1)$, otherwise require $O(k)$. If we suppose that half points move from their clusters, this requires $O(nk/2)$, since the algorithm converges to local minimum, the number of points moved from their clusters decreases in each iteration. So we expect the total cost is $nk\sum_{i=1}^{l} 1/i$. Even for large number of iterations, $nk\sum_{i=1}^{l} 1/i$ is much less than *nkl*. So the cost of using enhanced *k*-means algorithm approximately is $O(nk)$, not $O(nkl)$.

EXPERIMENTAL RESULTS

We have evaluated our algorithm on several different real datasets and synthetic dataset. We have compared our results with that of *k*-means algorithm in terms of the total execution time and quality of clusters. We also did a comparison with the CLARA algorithm. Our experimental results are reported on PC 800 MHz CPU, 128 MB RAM, 256 kB Cache.

Real datasets

In this section, we give a brief description of the datasets used in our algorithm evaluation. Table 1 shows some characteristics of the datasets.

Table 1 Characteristic of the datasets

Dataset	Number of records	Number of attributes
Letters	20000	16
Abalone	4177	7
Wind	6574	15

1. Letter Image dataset

This dataset represents the image of English capital letters. The image consists of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

Fig.5 shows that, enhanced *k*-means algorithm produces the final clusters in shortest period of time. The overlapped curves in Fig.6 show that, the three implementations of *k*-means algorithm produce clusters approximately of the same quality.

2. Abalone dataset

This dataset represents physical measurements of abalone (sea organism). Each abalone is described with 8 attributes.

Fig.7 shows that, the enhanced *k*-means algorithm is the most efficient algorithm. The overlapped curves in Fig.8 shows that the final clusters are approximately of the same quality.

3. Wind dataset

This dataset represents measurements on wind from 1/1/1961 to 31/12/1978 (Figs.9 and 10). These observation of wind are described by 15 attributes.

Synthetic datasets

Our generated datasets are semi-spherical shaped, in two dimensions. This dataset was generated

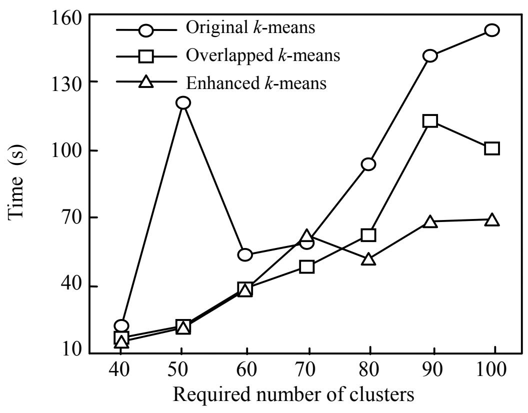

Fig.5 Execution time (Letter Image dataset)

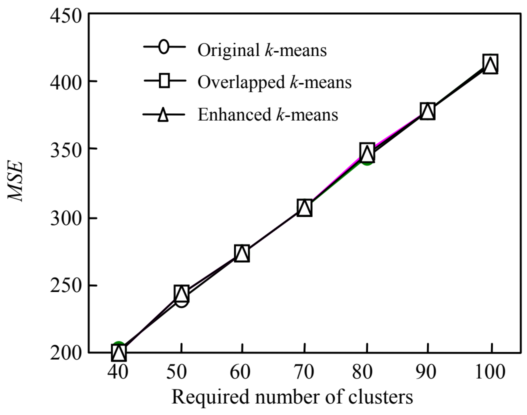

Fig.6 Quality of clusters (Letter Image dataset)

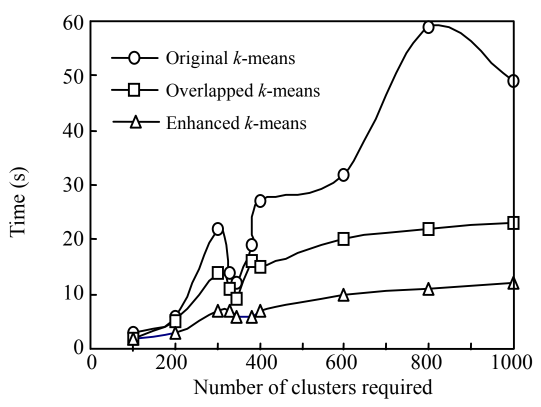

Fig.7 Execution time (Abalone dataset)

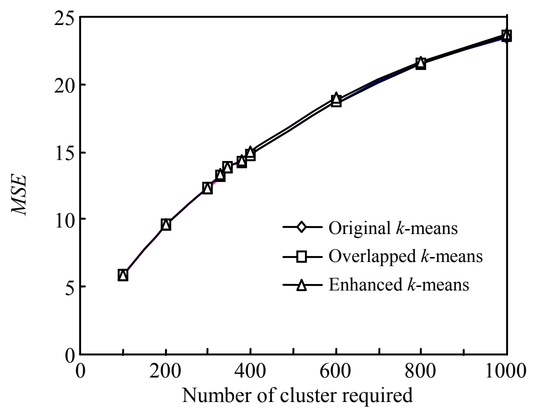

Fig.8 Quality of clusters (Abalone dataset)

randomly using visual basic code. Fig.11 shows a sample of them.

Fig.12 shows that, enhanced *k*-means algorithm is the most efficient algorithm in all cases of our synthetic datasets.

In the following series of experiments, we compared the three implementation of the *k*-means algorithm with CLARA algorithm. Fig.13 presents the performance of these algorithms, that reflect CLARA is not suitable for large dataset especially when it contains a large number of clusters. This is because CLARA is based on sample, with size proportional to the number of clusters.

When we compare the execution time of the three implementations of *k*-means algorithm with CLARA algorithm, we may be unable to distinguish between different implementations of *k*-means algorithm, this because CLARA takes very large time to produce the results, and *k*-means algorithm takes very short time.

CONCLUSION

In this paper we presented a simple idea to enhance the efficiency of *k*-means clustering. Our experimental results demonstrated that both of our schemes can improve the execution time of *k*-means algorithm, with no miss of clustering quality in most cases. From our result we conclude that, the second proposed implementation of the *k*-means algorithm (i.e. enhanced *k*-means algorithm) is the best one.

References

Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P., 1998. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. Proc. ACM SIGMOD Int. Conf. on Management of Data. Seattle, WA, p.94-105.

Ankerst, M., Breunig, M., Kriegel, H.P., Sander, J., 1999. OPTICS: Ordering Points to Identify the Clustering Structure. Proc. ACM SIGMOD Int. Con. Management of Data Mining, p.49-60.

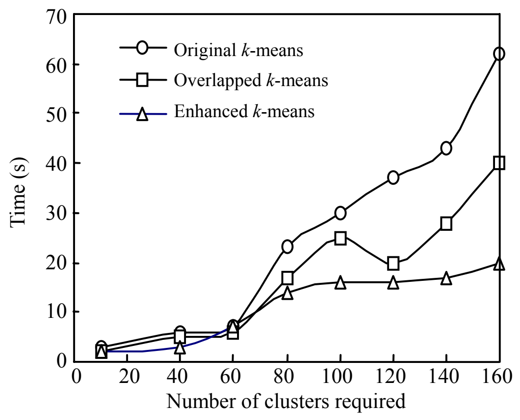

Fig.9 Execution time (Wind dataset)

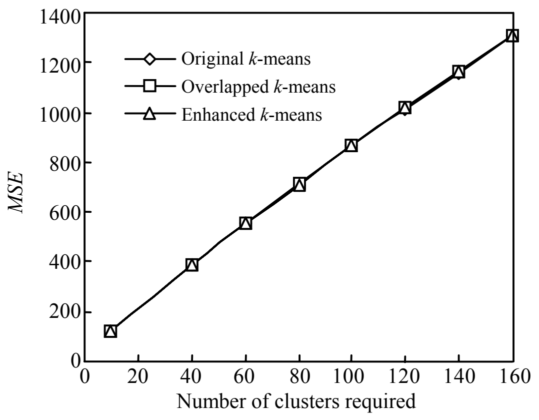

Fig.10 Quality of clusters (Wind dataset)

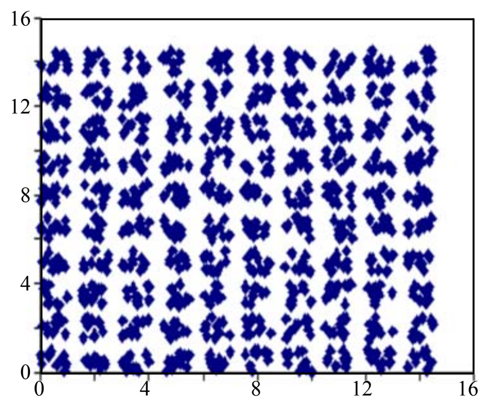

Fig.11 Random sample of the Synthetic dataset

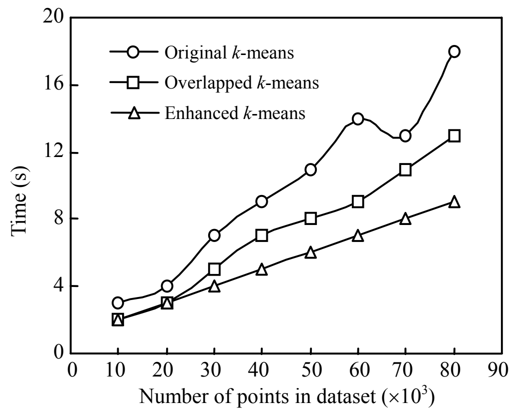

Fig.12 Execution time for different Synthetic datasets

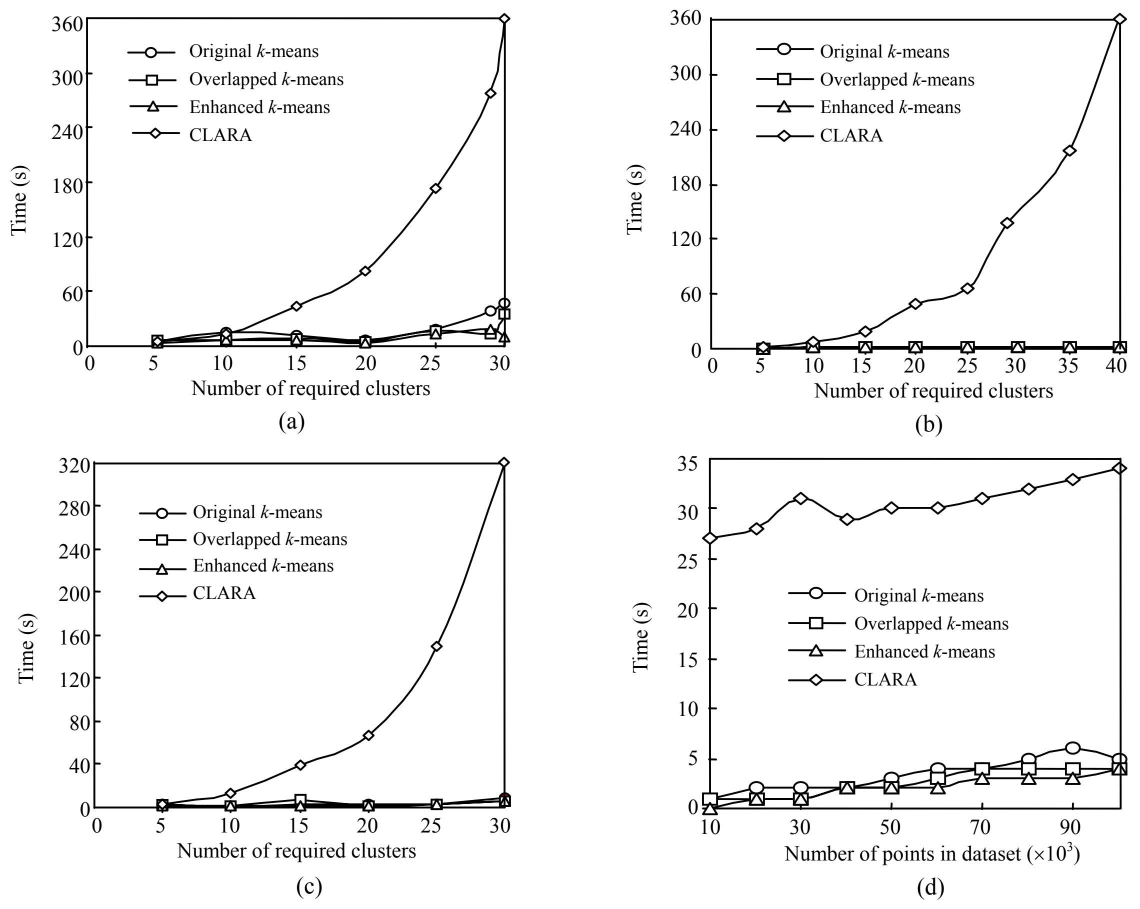

Fig.13 Execution time. (a) Letters dataset; (b) Abalone dataset; (c) Wind dataset; (d) Synthetic dataset

Duda, R.O., Hart, P.E., 1973. Pattern Classification and Scene Analysis. John Wiley & Sons, New York.

Ester, M., Kriegel, H.P., Sander, J., Xu, X., 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining. AAAI Press, Portland, OR, p.226-231.

Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., 1996. Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press.

Gersho, A., Gray, R.M., 1992. Vector Quantization and Signal Compression. Kluwer Academic, Boston.

Guha, S., Rastogi, R., Shim, K., 1998. CURE: An Efficient Clustering Algorithms for Large Databases. Proc. ACM SIGMOD Int. Conf. on Management of Data. Seattle, WA, p.73-84.

Hinneburg, A., Keim, D., 1998. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining. New York City, NY.

Huang, Z., 1997. A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining. Proc. SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery. Tech. Report 97-07, Dept. of CS, UBC.

Jain, A.K., Dubes, R.C., 1988. Algorithms for Clustering Data. Prentice-Hall Inc.

Kaufman, L., Rousseeuw, P.J., 1990. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons.

MacQueen, J., 1967. Some Methods for Classification and Analysis of Multivariate Observations. 5th Berkeley Symp. Math. Statist. Prob., **1**:281-297.

Merz, P., 2003. An Iterated Local Search Approach for Minimum Sum of Squares Clustering. IDA 2003, p.286-296.

Ng, R.T., Han, J., 1994. Efficient and Effective Clustering Methods for Spatial Data Mining. Proc. 20th Int. Conf. on Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, p.144-155.

Sheikholeslami, G., Chatterjee, S., Zhang, A., 1998. Wave-Cluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. Proc. 24th Int. Conf. on Very Large Data Bases. New York, p.428-439.

Sibson, R., 1973. SLINK: an optimally efficient algorithm for the single-link cluster method. *The Comp. Journal*, **16**(1):30-34. [doi:10.1093/comjnl/16.1.30]

Zhang, T., Ramakrishnan, R., Linvy, M., 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. Proc. ACM SIGMOD Int. Conf. on Management of Data. ACM Press, New York, p.103-114.