Information Sciences 181 (2011) 716–731

Contents lists available at ScienceDirect

Information Sciences

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

A time-efficient pattern reduction algorithm for *k*-means clustering

Ming-Chao Chiang [a,*], Chun-Wei Tsai [a,b,c], Chu-Sing Yang [b]

[a] Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan, ROC
[b] Department of Electrical Engineering, National Cheng Kung University, Tainan 70101, Taiwan, ROC
[c] Department of Applied Geoinformatics, Chia Nan University of Pharmacy & Science, Tainan 71710, Taiwan, ROC

ARTICLE INFO

Article history:
Received 25 June 2009
Received in revised form 27 September 2010
Accepted 7 October 2010

Keywords:
Data clustering
k-means
Pattern reduction

ABSTRACT

This paper presents an efficient algorithm, called pattern reduction (PR), for reducing the computation time of *k*-means and *k*-means-based clustering algorithms. The proposed algorithm works by compressing and removing at each iteration patterns that are unlikely to change their membership thereafter. Not only is the proposed algorithm simple and easy to implement, but it can also be applied to many other iterative clustering algorithms such as kernel-based and population-based clustering algorithms. Our experiments—from 2 to 1000 dimensions and 150 to 10,000,000 patterns—indicate that with a small loss of quality, the proposed algorithm can significantly reduce the computation time of all state-of-the-art clustering algorithms evaluated in this paper, especially for large and high-dimensional data sets.

1. Introduction

Data clustering [27,28,62] refers to the process of grouping similar data into the same cluster or splitting dissimilar data into different clusters according to some predefined criteria. Apparently, methods [28] developed to solve the data clustering problem can be easily extended and applied to many different problem domains, such as search engine [23,18,65], document analysis [64,37], bio-information [8,21], financial analysis [35], and human face recognition [60,20]. While considering data clustering as the epitome of metaheuristics, many researchers have adopted different approaches to solve the data clustering problem, including *k*-means [28], genetic algorithm (GA) [31,3], fuzzy system [50], tabu search (TS) [47,42], ant colony optimization (ACO) [32,22], self-organizing map (SOM) [59], particle swarm optimization (PSO) [58,10], support vector machine (SVM) [12,9], and artificial immune system (AIS) [5]. Of them, *k*-means [45] is an extensively adopted clustering algorithm. However, the above approaches largely focus on increasing the accuracy of the clustering result.

Today, most data clustering algorithms are confronted with extremely large data sets that require online processing. That is, in addition to quality, response time is of major concern to most data clustering algorithms nowadays. Despite performing well on small- to medium-sized data sets, traditional clustering algorithms fail to scale up well for large data sets, especially in terms of computation time. Therefore, of major concern is designing a scalable clustering algorithm, capable of solving large data set problems [63,28,30,2,36] that require terabytes or even petabytes of space such as web contents, magnetic resonance imaging (MRI), video streaming, astronomy, and finance.

Xu and Wunsch [63] categorized algorithms for "large-scale" data clustering as random sampling [48], data condensation [67], density-based approaches [16], grid-based approaches [61], divide and conquer [24], and incremental learning [25]. Another problem worthy of note is "high-dimensional" data clustering. As the computation time of high-dimensional data clustering is generally proportional to the number of dimensions of the input data, some researchers [15,63] have focused on reducing the number of dimensions of the input data, by decreasing the number of features in the original data.

* Corresponding author. Tel.: +886 7 5252000x4321; fax: +886 7 5254301.
E-mail address: mcchiang@cse.nsysu.edu.tw (M.-C. Chiang).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2010.10.008

This paper presents an efficient algorithm, called pattern reduction (PR), for reducing the computation time of k-means and k-means-based clustering algorithms. Its performance is evaluated by applying the proposed algorithm to five state-of-the-art clustering algorithms: standard k-means [45], relation k-means [49], kernel k-means [52,53], triangle inequality k-means [14], and genetic k-means algorithm [31]. Moreover, the proposed algorithm is thoroughly analyzed in terms of the adopted strategies and the time complexity.

The remainder of the paper is organized as follows. Section 2 briefly discusses the related work. Section 3 presents the proposed algorithm and the motivation of the work. Section 4 provides a detailed analysis of the proposed algorithm. The experimental results, along with the data sets and the parameter settings, are discussed in Section 5. Conclusions are drawn in Section 6.

2. Related work

This section briefly reviews the clustering problem and discusses the k-means clustering algorithm and the issues it faces.

2.1. The clustering problem

Given n patterns, or data points, in d-dimensional space, the clustering problem refers to the process of partitioning the n patterns into k groups or clusters based on some similarity metrics. An optimal clustering is a partitioning that minimizes the intra-cluster distance and maximizes the inter-cluster distance. Vesanto and Alhoniemi [59] presented several ways to measure the quality of the clustering result. In practice, the most popular metric is the sum of squared errors [28] defined as

$$\mathbf{SSE} = \sum_{i=1}^{k} \sum_{j=1}^{n_i} \|x_{ij} - c_i\|^2, \tag{1}$$

where $c_i = (1/n_i)\sum_{j=1}^{n_i} x_{ij}$ denotes the mean of the ith cluster; k the number of clusters; x_{ij} the jth pattern in the ith cluster; n_i the number of patterns in the ith cluster; and $n = \sum_{i=1}^{k} n_i$. Accuracy, F-measure, and entropy [41] provide alternative ways to measure the quality of the clustering result.

2.2. The k-means clustering algorithm

This work focuses on k-means [45] clustering, which is by far the most widely used partitional algorithm for data clustering for a very simple reason. As outlined in Fig. 1, k-means is simple and easy to implement.

However, as is well known, k-means has several limitations [51,27,28,62]:

- *Scalability:* It scales poorly computationally.
- *Initial means:* The clustering result is extremely sensitive to the initial means.
- *Noise:* Noise, or outliers, deteriorates the quality of the clustering result.
- *Number of clusters:* The number of clusters must be determined before the k-means clustering begins.
- *Local minima:* It always converges to local minima.
- *Inability to cluster non-linearly separable data set:* It fails to split non-linearly separable data sets in the input space.

Scalability of the k-means clustering algorithm, especially for large or high-dimensional data sets, has received extensive attention recently. Scalable k-means [7] uses buffering and a two-stage compression scheme to either compress or discard patterns to enhance the performance of k-means. However, according to Ordonez and Omiecinski [49], scalable k-means is slightly faster than standard k-means, but not always. The most important factors affecting the performance of scalable k-means are parameter settings and compression processes such as buffer size and compression ratio. Simple single pass k-means [17] was also developed to reduce the computation time. Relational k-means [49] uses the block and incremental concept to provide a more stable method than scalable k-means. Moreover, the computation time of k-means can be reduced using parallel [39] and triangle inequality [14] methods.

To resolve the problem of the clustering result being extremely sensitive to the *initial means*, Bradley and Fayyad [6] adopted a random sampling method to obtain $K \times J$ centers for use in building a better initial solution. Laszio and Mukherjee

k1.	Randomly choose k cluster centers from all the patterns in the data set D.
k2.	Assign each pattern in D to the closest cluster center; then update the cluster centers.
k3.	If the new cluster centers and the previous ones are the same, then terminate; otherwise, return to step k2.

Fig. 1. Outline of the k-means algorithm.

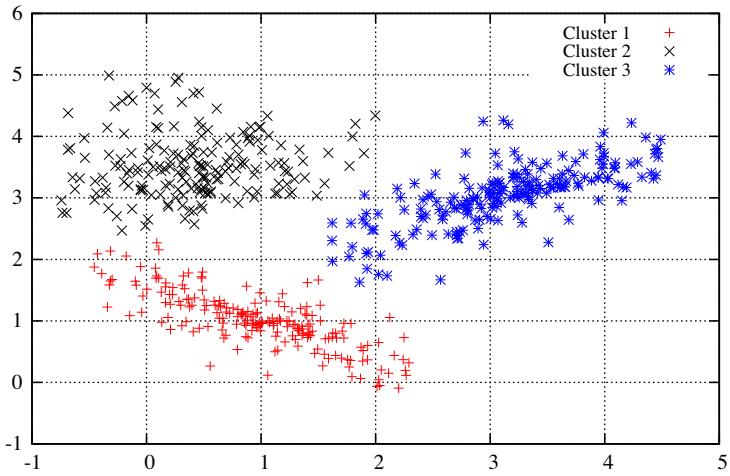

Fig. 2. The end result of partitioning a 579-pattern data set into three clusters [56].

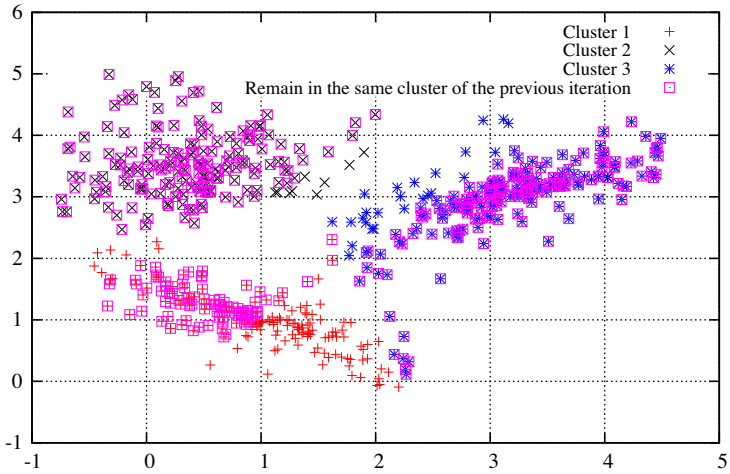

Fig. 3. The result at the end of iteration 2.

[34] devised a genetic algorithm using hyper-quadtrees to identify a good initial center set. However, if the data set is *noisy* or consists of outliers [29,26],[1] the results of classical clustering algorithms degrade. The sampling method [6] can help mitigate this problem. In [26], Hua et al. detected the outliers by using the distances of a pattern to the closest and second closest cluster centers and the distance between the two centers.

Rather than standard k-means determining the *number of clusters* k automatically, k is assumed to be given. Likas et al. [40] developed a modified k-means algorithm that can dynamically add a cluster center to determine suitable initial positions. The X-means [51] can not only accelerate the iterative process but also find the best k for k-means. Bisecting k-means [54,55] starts off with a single cluster containing all of the data. The process of splitting a cluster up into two is then repeated until the desired number of clusters is reached.

Many studies [3,31–33,47] have attempted to combine k-means with other heuristic algorithms to prevent k-means from falling into *local minima*. Genetic k-means algorithm (GKA) [31] and k-means with genetic algorithm (KGA) [3] use k-means to find the local minima and genetic algorithm to search for the global minimum. Kuo et al. [32] used ACS and SOM, while Ng and Wong [47] used tabu search combined with k-means to obtain results better than those of standard k-means.

The inability of k-means to separate data that are *non-linearly separable* in the input space has received considerable attention recently. Kernel k-means [52,53,66,13] was developed to solve this problem by transforming the input data to a

[1] More precisely, as far as this paper is concerned, noise denotes patterns that are close to two or more cluster centers while outlier represents patterns that are distant from the cluster to which they belong.

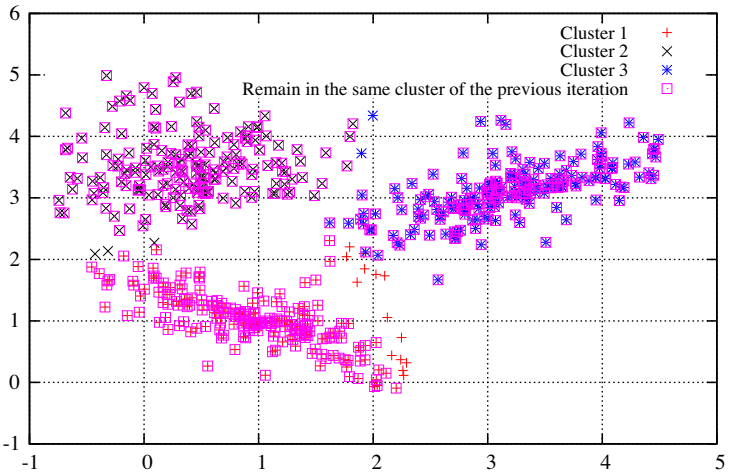

Fig. 4. The result at the end of iteration 3.

Iter	Patterns				Computation of the mean c_a
	x_1	x_2	x_3	x_4	
2	a	a	b	b	$c_a^2 = (x_1 + x_2)/2 = \bar{x}_{1,2}$
3	a	a	a	b	$c_a^3 = ((x_1 + x_2) + x_3)/3 = ((2 \times \bar{x}_{1,2}) + x_3)/3$
4	a	a	a	b	$c_a^4 = ((x_1 + x_2) + x_3)/3 = ((2 \times \bar{x}_{1,2}) + x_3)/3$

Fig. 5. Example illustrating how the mean of group a, c_a, is computed.

high-dimensional space in order to separate them. The difficulty in implementing kernel k-means is that the kernel matrix H takes up a considerable amount of memory. Scheme kernel k-means [66] was proposed to solve this problem by splitting the kernel matrix H into blocks.

3. The proposed algorithm

This section gives the motivation of the work and discusses in detail the PR algorithm.

3.1. Motivation of this work

The work is motivated by the observation that most k-means-based clustering algorithms behave extremely similar at the later stage of convergence in the sense that most of the computations are repeated yet contribute nothing to the final solution. For instance, Fig. 2 shows the final result of a 579-pattern data set partitioned into three clusters by k-means. Figs. 3 and 4 summarize the results at the end of iterations 2 and 3, indicating that a large number of patterns (those enclosed in rectangles) remain in the same cluster of the previous iteration. This makes clear that many computations of k-means such as the computation of means, the computation of distances to means, and the assignment of patterns to the nearest mean are essentially redundant and thus can be eliminated to save its computation time.

For concreteness, Fig. 5 gives a highly simplified example to show that in the computation of a mean, many of the computations are essentially redundant. As depicted in Fig. 5, the data set D consists of four patterns, x_1, x_2, x_3, and x_4. Patterns x_1 and x_2 are clustered into the same group a at iterations 2, 3 and 4. Pattern x_3 is assigned to group b at iteration 2 and to the group a at iterations 3 and 4. Pattern x_4 is assigned to the group b at iterations 2, 3, and 4. By using superscript to indicate the iteration number and subscript to represent the group number, the mean of group a at iteration 2 can be computed as $c_a^2 = (x_1 + x_2)/2$ and at iterations 3 and 4 as $c_a^3 = c_a^4 = ((x_1 + x_2) + x_3)/3$. The mean of group b at iteration 2 can be computed as $c_b^2 = (x_3 + x_4)/2$ and at iterations 3 and 4 as $c_b^3 = c_b^4 = x_4$. Obviously, the computations of $x_1 + x_2$ at iterations 3 and 4 and the computation of $(x_1 + x_2) + x_3$ at iteration 4 are redundant. If patterns x_1 and x_2 are compressed into a single pattern, say, $\bar{x}_{1,2} = (x_1 + x_2)/2$ and removed at iteration 2, then the mean of group a can be computed as $c_a^3 = c_a^4 = ((2 \times \bar{x}_{1,2}) + x_3)/3$ at iterations 3 and 4. It is clear from the above discussion that by compressing and removing at each iteration patterns that do not change their membership thereafter, the computation time can be significantly reduced relatively easily.

One of the reasons why pattern x_3 is not compressed and removed at iterations 3 and 4 is that pattern x_3 is far away from its means c_a^3 and c_a^4 and thus is likely to change its membership later. Compressing and removing pattern x_3 will make it remain in the cluster from which it is compressed and removed permanently. Consequently, the accuracy rate[2] of the clustering result will be decreased. Another reason is that the number of iterations pattern x_3 remains in the cluster a is less than those of the other two patterns x_1 and x_2. From the perspective of the convergence characteristics of the k-means algorithm, it is clear that the longer a pattern remains in a cluster, the less likely it will change its membership.

Despite shedding light on how the PR algorithm works, the above example is probably much too small to accurately reflect the amount of computation time that can be reduced. To further clarify the notion of PR, here is another example. Assume that cluster i consists of 10,000 patterns at the beginning of iteration ℓ. Further assume that half of the 10,000 patterns are compressed and removed at the end of iteration ℓ. Next, consider the cost of computing the mean of cluster i at iteration $\ell + 1$, assuming that no patterns are moved in or out of cluster i at iteration $\ell + 1$. Without pattern compression and removal at iteration ℓ, 9,999 additions and one division are required to compute the mean of cluster i at iteration $\ell + 1$. With pattern compression and removal at iteration ℓ, the cost is reduced to one multiplication,[3] 5,000 additions, plus one division—a considerable savings in terms of the number of computations. This analysis is intended for only one cluster and is based on the assumption that the data points are one-dimensional. If the number of clusters k and the number of dimensions d are considered, the savings is proportional to $k \times d$, which is consistent with the experimental results in Section 5.

Furthermore, as far as the PR algorithm is concerned, patterns that are compressed and removed will remain in the cluster from which they are compressed and removed forever. That is, for each pattern, the compression and removal process is done once and only once, although a compressed pattern may again be compressed and removed at later iterations. More important, by using a pattern to represent a large number of patterns, the number of computations can be significantly reduced.

3.2. Notations

To simplify our discussion of the PR algorithm in the next subsection, the following notations are used throughout the rest of this section.

- n number of input patterns or data points
- k number of clusters
- ℓ superscript denoting the iteration number beginning with 1, i.e., $\ell = 1,2,\ldots$, up until the iteration stops
- d number of dimensions of the input patterns
- $\widetilde{\mathbf{D}}$ array of input patterns or data points, i.e., for each i, $1 \leqslant i \leqslant n$, $\widetilde{\mathbf{D}}[i] = x_i$ where x_i denotes the ith input pattern
- $\mathbf{D}$ copy of $\widetilde{\mathbf{D}}$, i.e., for each i, $1 \leqslant i \leqslant n$, $\mathbf{D}[i] = \widetilde{\mathbf{D}}[i]$ initially (and only initially). In other words, $\widetilde{\mathbf{D}}$ and $\mathbf{D}$ differ in that $\widetilde{\mathbf{D}}$ will remain intact, but $\mathbf{D}$ will be changed by the PR algorithm. From this perspective, the sole purpose of $\widetilde{\mathbf{D}}$ is to make it easy to locate patterns belonging to each cluster at the end of k-means with PR. Furthermore, with the arrays $\widetilde{\mathbf{D}}$ and $\mathbf{D}$ are associated two arrays $\mathbf{C}$ and $\mathbf{M}$ of the same size as the arrays $\widetilde{\mathbf{D}}$ and $\mathbf{D}$.
- $\mathbf{C}$ array whose element denotes the cluster to which each pattern belongs. That is, for each i, $1 \leqslant i \leqslant n$, $\mathbf{C}[i] \in \{1,2,\ldots,k\}$ denotes the cluster to which the pattern $\mathbf{D}[i]$, and the original pattern $\widetilde{\mathbf{D}}[i]$, belongs.
- $\mathbf{M}$ array whose element denotes the state of each pattern. That is, for each i, $1 \leqslant i \leqslant n$, $\mathbf{M}[i]$ denotes if pattern $\mathbf{D}[i]$ is removed or if pattern $\mathbf{D}[i]$ is x_i or if pattern $\mathbf{D}[i]$ is a compression of $\mathbf{M}[i]$ patterns. Initially, $\mathbf{M}[i] = 1$, indicating that no pattern is compressed and removed, i.e., $\mathbf{D}[i] = x_i$. When pattern $\mathbf{D}[i]$ is removed, $\mathbf{M}[i]$ associated with it will be set to 0. $\mathbf{M}[i] > 1$ indicates that the corresponding pattern $\mathbf{D}[i]$ is the average of the $\mathbf{M}[i]$ patterns removed.
- $|\mathbb{S}|$ cardinality of the set $\mathbb{S}$
- $\mathbb{C}_i^\ell$ set of indices to the arrays $\mathbf{D}$, $\mathbf{C}$, and $\mathbf{M}$, indicating patterns assigned to cluster i and their state at iteration ℓ. That is,

$$\mathbb{C}_i^\ell = \{s_{i1}^\ell, s_{i2}^\ell, \ldots, s_{i|\mathbb{C}_i^\ell|}^\ell\}$$

  and for each i and j, $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant |\mathbb{C}_i^\ell|$, $\mathbf{D}[s_{ij}^\ell]$ denotes the jth pattern assigned to cluster i; $\mathbf{C}[s_{ij}^\ell] = i$ the cluster to which pattern $\mathbf{D}[s_{ij}^\ell]$ is assigned; and $\mathbf{M}[s_{ij}^\ell]$ the state of pattern $\mathbf{D}[s_{ij}^\ell]$ at iteration ℓ.
- $\mathbb{R}_i^\ell$ set of indices to the arrays $\mathbf{D}$, $\mathbf{C}$, and $\mathbf{M}$, indicating patterns removed from cluster i and their state at iteration ℓ. That is,

$$\mathbb{R}_i^\ell = \{r_{i1}^\ell, r_{i2}^\ell, \ldots, r_{i|\mathbb{R}_i^\ell|}^\ell\}$$

  and for each i and j, $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant |\mathbb{R}_i^\ell|$, $\mathbf{D}[r_{ij}^\ell]$ denotes the jth pattern removed from cluster i; $\mathbf{C}[r_{ij}^\ell] = i$ the cluster from which pattern $\mathbf{D}[r_{ij}^\ell]$ is removed; and $\mathbf{M}[r_{ij}^\ell]$ the state of pattern $\mathbf{D}[r_{ij}^\ell]$ at iteration ℓ. Moreover, $\mathbb{R}_i^\ell$ is a subset of $\mathbb{C}_i^\ell$, and it can be empty, i.e., $\mathbb{R}_i^\ell = \emptyset$, indicating that no patterns are compressed and removed from cluster i at iteration ℓ.
- $\mathbb{C}^\ell$ union of $\mathbb{C}_i^\ell$, i.e., $\mathbb{C}^\ell = \cup_{i=1}^k \mathbb{C}_i^\ell = \{1,2,\ldots,n\}$
- $\mathbb{R}^\ell$ union of $\mathbb{R}_i^\ell$, i.e., $\mathbb{R}^\ell = \cup_{i=1}^k \mathbb{R}_i^\ell$

[2] The accuracy rate refers to the percentage in which the input patterns are classified into the correct cluster.

[3] For most commercially available computer architectures, one multiplication is obviously less expensive computationally than 4,999 additions.

PRk1.	Generate a better initial solution using any method that can generate such a solution efficiently.
PRk2.	If it is time to start the PR algorithm and the removal bound has not been reached, then apply the PR algorithm; otherwise, assign each pattern in D to the closest cluster center; then update the cluster centers.
PRk3.	If the new cluster centers and the previous ones are the same, then terminate; otherwise, return to step PRk2.

Fig. 6. Outline of the *k*-means algorithm with PR.

3.3. The PR algorithm

Fig. 6 gives an outline of the *k*-means algorithm with PR. Initially, the "combined" algorithm works exactly the same as *k*-means except that it continues to check whether it is the right time to start the PR algorithm. If it determines the right time to do so and the removal bound has not been reached, the PR algorithm is applied. Then, it continues to check whether or not to stop the PR algorithm based on the removal bound.

Basically, the PR algorithm can be divided into two parts:

1. Pattern compression and removal (PCR), and
2. Pattern assignment and mean update (PAMU).

In practice, however, an optional step is generally added to generate a "better" initial solution—better in the sense that it is closer to the optimal solution than a randomly generated one—in order to mitigate the problem of PR being extremely sensitive to the initial solution. A detailed discussion follows.

3.3.1. Generation of a better initial solution

The initial solution of the *k*-means algorithm is usually generated randomly. However, to improve the clustering result of the *k*-means algorithm that is extremely sensitive to the initial solution, many researchers [6,34] have adopted a non-random procedure to create a "better" initial solution, thus yielding a better clustering result. Using the non-random procedure also creates a better initial solution to prevent PR from removing patterns that belong to other clusters and thus should not be removed at early iterations. This is owing to the fact that similar to the *k*-means algorithm, PR is extremely sensitive to the initial solution and probably even more so because our goal is to start PR as early as it can. For instance, as shown in Fig. 7(a), PR compresses and removes patterns that have a small probability of migrating from cluster 1 to clusters 2 and 3. Nevertheless, if the current mean is too far away from the optimal mean, as depicted in Fig. 7(b), it is very likely that PR compresses and removes patterns that belong to clusters 2 and 3 instead of cluster 1.

In this paper, a very simple method is used to create a better initial solution to mitigate the problem of PR being extremely sensitive to the initial solution. It works as follows: First, the proposed algorithm selects a certain percentage of the input patterns by random sampling. Then, these patterns are clustered by using the standard *k*-means algorithm. Finally, the outcome serves as the initial solution of the proposed algorithm. It is worth noting that random sampling is neither the only nor the best method available to generate a better initial solution for PR. In fact, any method that can generate a better initial solution efficiently would suffice. This paper utilizes random sampling to demonstrate the feasibility of using such a simple method to mitigate the problem of PR being extremely sensitive to the initial solution.

3.3.2. Pattern compression and removal (PCR)

Fig. 8 depicts the procedure for pattern compression and removal (PCR). This procedure shows how patterns are compressed and removed by the PR algorithm. First, PR requires that a removal bound be set to denote the percentage of patterns that are allowed to be compressed and removed. Such a bound is set partially owing to the need to reduce the extent to which noise in the input data impacts the clustering result. Ideally, if the input data are free of noise or no patterns are fuzzy about to which clusters they should belong, the removal bound can be set to 100%. This implies that all of the patterns are allowed to be compressed and removed, and at the end, only k patterns representing the k cluster centers are retained. Our experimental results show that setting the removal bound to 80%—with respect to all of the removal bounds from 10% to 100% with an increment of 10% tested in this work—gives a satisfactory result. Notably, setting the removal bound to too large of a value may decrease the accuracy rate whereas setting the removal bound to too small of a value imposes a limit on the amount of computation time that can be reduced.

According to Fig. 8, if the removal bound has not been reached, then for each cluster i, PCR first checks to see which patterns in that cluster are near the mean and thus can be removed. Next, PCR compresses and removes these patterns by selecting one of the patterns to be removed, say, pattern $\mathbf{D}[r_i^\ell]$ where $r_i^\ell \in \mathbb{R}_i^\ell$, as the representative pattern and setting its value to the average of all patterns removed, as follows:

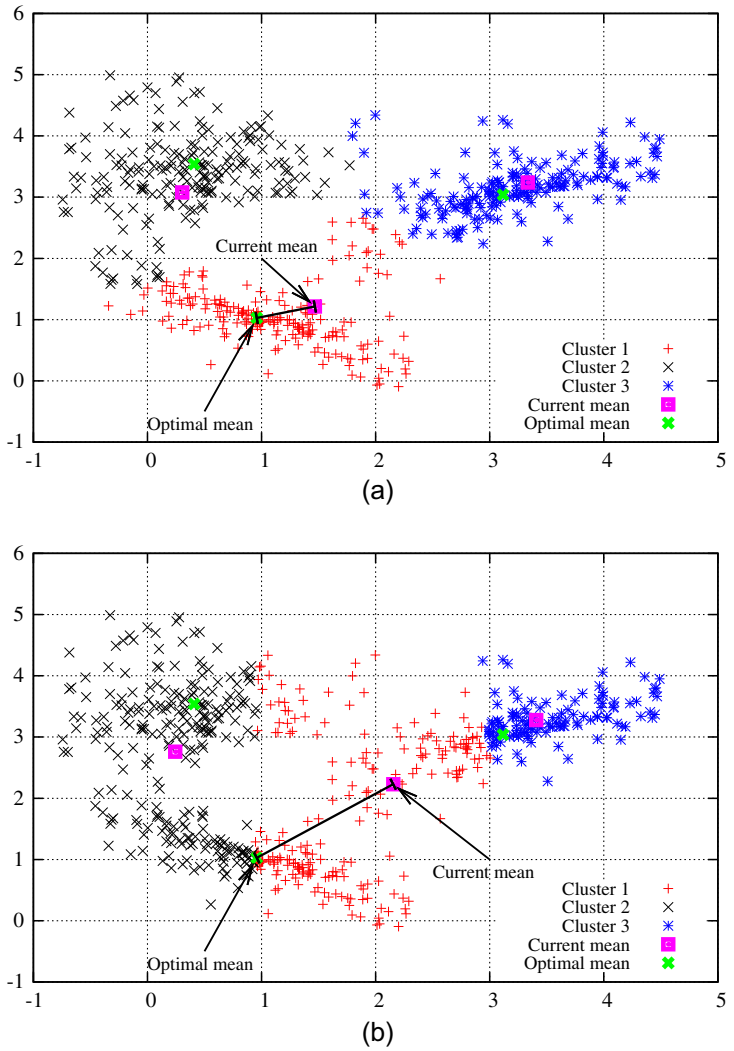

Fig. 7. Example showing that the initial means may affect the compression and removal result. (a) As marked, the current mean is close to the optimal mean. Thus, only patterns that have a low probability of changing their membership are compressed and removed. (b) As marked, the current mean is distant from the optimal mean. Thus, PR has a high probability of compressing and removing patterns that should not be compressed and removed.

procedure PCR
{
  if the percentage of patterns removed < removal bound **do**
    for each cluster i **do**
      1. Compute the mean μ and standard deviation σ of the distances of all patterns in the cluster to their cluster center.
      2. Use the μ and σ computed above to locate and remove patterns near the cluster center.
      3. Compress the patterns removed into a new pattern by Eq. (2) and update **M** accordingly by Eq. (3).
    end
  end
}

Fig. 8. Procedure to compress and remove patterns.

$$\mathbf{D}[r_i^\ell] = \frac{\sum_{j=1}^{|\mathbb{R}_i^\ell|} \mathbf{D}[r_{ij}^\ell]}{|\mathbb{R}_i^\ell|}. \tag{2}$$

```
procedure PAMU
{
    if the percentage of patterns removed < removal bound do
        for each cluster i do
            Reassign each pattern not removed to the closest cluster center.
        end
        for each cluster i do
            Compute the new mean c_i^ℓ by Eq. (4).
        end
        Compute the new SSE at iteration ℓ, i.e., SSE^ℓ, by Eq. (5).
    end
}
```

Fig. 9. Procedure to assign patterns and update means.

Then, for each pattern removed, i.e., for all $z \in \mathbb{R}_i^\ell$, the value of $\mathbf{M}[z]$ is set to zero except, of course, $\mathbf{M}[r_i^\ell]$ whose value is set to the number of patterns removed, as follows:

$$\mathbf{M}[z] = \begin{cases} |\mathbb{R}_i^\ell| & \text{if } z \in \mathbb{R}_i^\ell \text{ and } z = r_i^\ell, \\ 0 & \text{if } z \in \mathbb{R}_i^\ell \text{ and } z \neq r_i^\ell. \end{cases} \tag{3}$$

It is important for the PR algorithm to keep track of the average and the number of patterns removed because once a pattern is removed, all information about it is lost. When computing the means at later iterations, the average and the number of patterns removed play a crucial role in recovering part of the lost information to reduce the degree to which outliers in the input data may drag the new means too far away from the real means, or more precisely, from the means of the standard *k*-means algorithm.

3.3.3. Pattern assignment and mean update (PAMU)

Fig. 9 depicts the procedure for pattern assignment and mean update (PAMU). Like PCR, PAMU is performed if the removal bound has not been reached. PAMU requires that the distances between each pattern and all the means be compared to determine the cluster to which that pattern belongs. Moreover, for the PR algorithm, if a pattern $\mathbf{D}[s_{ij}^\ell]$ is compressed and removed (i.e., $\mathbf{M}[s_{ij}^\ell] = 0$ or $\mathbf{M}[s_{ij}^\ell] > 1$), that pattern will remain in the cluster from which it was compressed and removed permanently. This implies that the representative pattern (i.e., $\mathbf{M}[s_{ij}^\ell] > 1$) will never be reassigned to a new cluster; instead, only patterns with $\mathbf{M}[s_{ij}^\ell] = 1$ can be reassigned to a new cluster. In other words, PAMU described in Fig. 9 reassigns each pattern with $\mathbf{M}[s_{ij}^\ell] = 1$ to the cluster to which it belongs first and then computes the new mean of each cluster i, as follows:

$$c_i^\ell = \frac{\sum_{j=1}^{|\mathbb{C}_i^\ell|} \mathbf{M}[s_{ij}^\ell] \times \mathbf{D}[s_{ij}^\ell]}{\sum_{j=1}^{|\mathbb{C}_i^\ell|} \mathbf{M}[s_{ij}^\ell]}. \tag{4}$$

Finally, the new **SSE** is computed, as follows:

$$\mathbf{SSE}^\ell = \sum_{i=1}^{k} \sum_{j=1}^{|\mathbb{C}_i^\ell|} \|\mathbf{D}[s_{ij}^\ell] - c_i^\ell\|^2 \times \mathbf{M}[s_{ij}^\ell]. \tag{5}$$

3.3.4. Applying PR to k-means-based algorithms

The basic idea of how the PR algorithm is applied to *k*-means-based algorithms is essentially the same as that of *k*-means. Using GKA [31]—a hybrid of GA and *k*-means—as an example, here is how it works. GKA is basically no different from simple GA except that the crossover operator is replaced by the *k*-means algorithm. Besides, GKA encodes solutions in the same way as *k*-means does; that is, each chromosome represents a solution, as the array **C** discussed earlier in Section 3.2 does. GKA uses one iteration *k*-means to replace the crossover operator. For all *k*-means-based algorithms that use a complete *k*-means and encode the solutions in the same way as *k*-means such as GKA, the PR algorithm can be applied directly, i.e., in exactly the same way as it is applied to *k*-means. Otherwise, all the PR algorithm needs to do is to re-encode the solutions first and then proceed as before.

4. Strategy analysis

This section begins with a simple example to illustrate how PR works, followed by the analysis of its strategies.

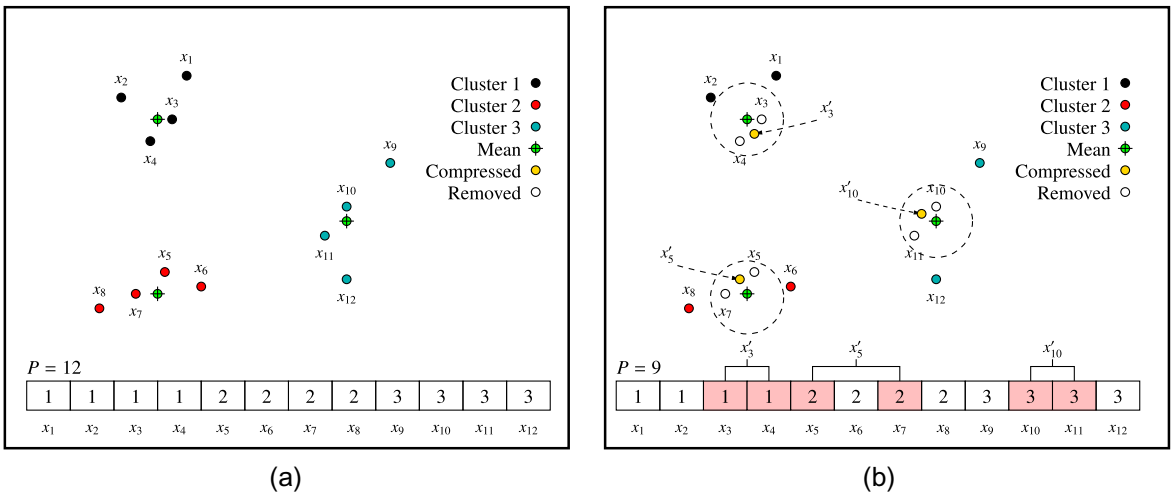

Fig. 10. A simple example illustrating how PR works, where P indicates the number of patterns remaining to be clustered at next iteration. (a) At the end of iteration $t-1$, 12 patterns are assigned to three clusters. (b) At iteration t, PR kicks in; patterns "close" to their means are compressed and removed. As a result, at the end of iteration t, only 9 patterns are left, meaning that PR needs to compute no more than 9 patterns thereafter. In other words, by compressing and removing at each iteration patterns that are unlikely to change their membership afterwards, the computation time can be significantly reduced.

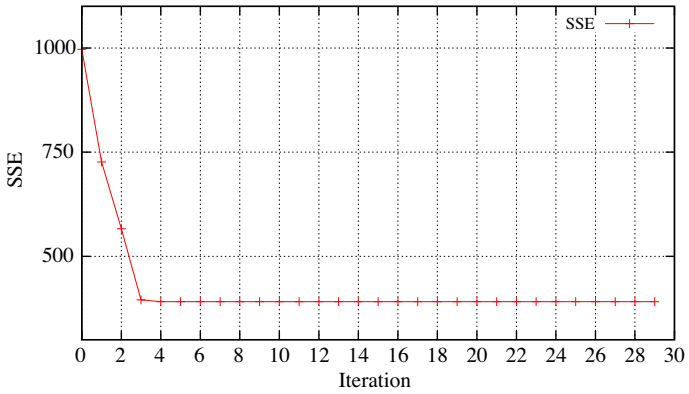

Fig. 11. Convergence process of k-means with 579 patterns.

4.1. Example

Fig. 10(a) shows that at the end of iteration $t-1$, 12 patterns, denoted x_i, for $i = 1,2,\ldots,12$, are assigned to three clusters numbered 1, 2, and 3. Then, at iteration t, PR is applied so that patterns "close" to their means are compressed and removed. More precisely, according to Fig. 10(b), patterns x_3 and x_4 of cluster 1 are compressed into a single pattern x'_3 and then removed, x_5 and x_7 of cluster 2 compressed into x'_5 and removed, and x_{10} and x_{11} of cluster 3 compressed into x'_{10} and removed because they are "close" to their means, i.e., within a predefined distance, and are unlikely to change their membership thereafter. It is important to note that PR needs to compute no more than nine patterns afterwards. In other words, by compressing and removing at each iteration patterns that are unlikely to change their membership later, the computation time of k-means and k-means-based algorithms can be significantly reduced.

4.2. Strategies

Strategies of the PR algorithm are discussed next. It is important to note that such strategies may significantly impact the performance of the PR algorithm.

4.2.1. Time to start

In Section 3.1, we showed that a large number of computations of the k-means algorithm on its convergence process are redundant. The results further indicated that if patterns that do not change their membership at later iterations are com-

Table 1
Comparison of the accuracy rate using different approaches to determining the right time to start the PR algorithm.

Benchmark	KM	KM + TS1	KM + TS2	KM + TS3
Iris (D = 4, P = 150, C = 3)	368 ms (81.28%)	309 ms (78.31%) −16.03% (−3.65%)	344 ms (78.75%) −6.52% (−3.11%)	464 ms (81.28%) +26.09% (+0.00%)
400 (D = 2, P = 400, C = 2)	453 ms (75.21%)	434 ms (75.61%) −4.19% (+0.54%)	354 ms (75.59%) −21.85% (+0.51%)	445 ms (75.21%) −1.77% (+0.00%)
579 (D = 2, P = 579, C = 3)	713 ms (96.55%)	537 ms (90.08%) −24.68% (−6.70%)	565 ms (91.13%) −20.76% (−5.61%)	626 ms (96.55%) −12.20% (+0.00%)
800 (D = 2, P = 800, C = 4)	1477 ms (94.89%)	855 ms (83.50%) −42.11% (−12.00%)	890 ms (88.08%) −39.74% (−7.18%)	1386 ms (94.82%) −6.16% (−0.07%)

D: # of dimensions, *P*: # of patterns, *C*: # of clusters.

pressed and removed at early iterations, a significant amount of computation time can be reduced in such a way that the quality of the clustering result is retained. However, if patterns are compressed and removed too early, say, at iteration 1 in Fig. 11, the *k*-means algorithm may fall into local minima, thus converging to an inaccurate result. If patterns are compressed and removed too late, say, at iteration 15 in Fig. 11, no computation time will be reduced. For this reason, the timing of starting the PR algorithm is extremely important.

In this paper, we assess three approaches to determining the right time to start the PR algorithm:

- The first approach is to start the PR algorithm at iteration 2 because the convergence speed of *k*-means slows down substantially after iteration 2 in the sense that a large number of patterns remain in the same group. Therefore, it is reasonable to assume that iteration 2 is the right time to start the PR algorithm to compress and remove patterns.
- The second approach is to find the best time to start the PR algorithm, which requires knowledge of the convergence characteristics of *k*-means. In other words, this approach requires that factors such as the number of patterns, the number of clusters, and the distribution of patterns be analyzed, and then statistics or other intelligent algorithms knowing the convergence characteristics before the *k*-means algorithm begins be used. For instance, compare the current **SSE** with the previous one. If the decrease rate is smaller than a threshold, the PR algorithm can be started. Unfortunately, it is extremely difficult to end up with a reasonable threshold because the decrease rate of **SSE** depends on factors such as the distribution of patterns and the number of clusters.
- The third approach is to start the PR algorithm when a large number of patterns remain in the same group for a certain number of iterations in a row.

Table 1 compares the accuracy rates of the above three approaches. TS1, TS2, and TS3 represent, respectively, the first, second, and third approaches. Each field is divided into four subfields. The upper-left subfield represents the time that each algorithm takes in milliseconds; the upper-right subfield the accuracy rate; the lower-left subfield the percentage of computation time reduced; the lower-right subfield the percentage of accuracy rate lost. For comparison, we set the threshold for TS2 to 0.3, implying that when the decrease rate of **SSE** is less than 0.3, *k*-means with PR will start the PR algorithm; the threshold and the number of iterations for TS3 to $n/2$ and 3, implying that when more than $n/2$ patterns remain in the same cluster for three iterations in a row, *k*-means with PR will start the PR algorithm. Our experimental results show that the PR algorithm is highly efficient in terms of both the accuracy rate and the computation time. Besides, the percentage of computation time reduced and the percentage of accuracy rate lost provide a viable means of analyzing the effectiveness of trading the accuracy rate for the computation time.

To better understand how time to start impacts the performance of the PR algorithm, we also test thresholds from 10% up to 90% with an increment of 10% for TS2 and TS3. The experimental results indicate that regardless of which threshold is used, the results are approximately the same in terms of both the accuracy rate and the computation time, despite the fact that the optimal thresholds of TS2 and TS3 depend to a certain extent on the data set in question except for TS1. The experimental results further demonstrate that the iterations at which TS2 and TS3 start the PR algorithm are extremely close to each other for all thresholds tested. In addition, TS1 is faster than TS2 and TS3. Therefore, in this work, TS1 is used as the time to start strategy for all the experiments described in Section 5.

4.2.2. Removal strategy

In this paper, we evaluate two methods for pattern compression and removal. The first method utilizes the mean μ and standard deviation σ of the distances of patterns in a cluster to their cluster center to locate and remove patterns that are among the top α% near the cluster center. The second method removes patterns that remain in the same cluster for a certain number of iterations in a row.

Table 2 compares the accuracy rates of the two methods, denoted by RM1 and RM2, respectively. For RM1, α is set to 50 because we want to remove patterns in a cluster the distances of which to their cluster center are smaller than μ. For RM2, the number of iterations is set to 3 because most of the patterns remaining in a cluster for three iterations in a row have a

Table 2
Comparison of the accuracy rate and the running time using different pattern compression and removal methods.

Benchmark	KM	KM + RM1	KM + RM2
Iris (D = 4, P = 150, C = 3)	364 ms (82.56%)	343 ms (80.44%) −5.77% (−2.57%)	281 ms (80.22%) −22.80% (−2.83%)
400 (D = 2, P = 400, C = 2)	419 ms (76.77%)	345 ms (77.02%) −17.66% (+0.33%)	315 ms (77.14%) −24.82% (+0.49%)
579 (D = 2, P = 579, C = 3)	805 ms (96.55%)	547 ms (89.46%) −32.05% (−7.34%)	538 ms (89.05%) −33.17% (−7.76%)
800 (D = 2, P = 800, C = 4)	1508 ms (93.21%)	836 ms (83.16%) −44.56% (−10.78%)	725 ms (82.72%) −51.92% (−11.25%)

D: # of dimensions, *P*: # of patterns, *C*: # of clusters.

Table 3
Data sets.

Data set	Benchmark	*k*	*d*	*n*
DSR1	Reuters-21578	25	302	8,284
DSR2	KDD-98 Data Set	20	56	95,413
DSR3	20 News Groups	10	1000	10,000
DSS1	Iris	3	4	150
DSS2	579	3	2	579
DSS3	800	4	2	800
DSS4	uci-sc	6	60	600
DSH1	c50d2n6000	50	2	6000
DSH2	c50d10n6000	50	10	6000
DSH3	c50d25n6000	50	25	6000
DSH4	c50d50n6000	50	50	6000
DSH5	c50d100n6000	50	100	6000
DSH6	c50d250n6000	50	250	6000
DSH7	c50d500n6000	50	500	6000
DSH8	c50d1000n6000	50	1000	6000
DSL1	c50d2n60000	50	2	60,000
DSL2	c50d2n600000	50	2	600,000
DSL3	c50d2n6000000	50	2	6,000,000
DSL4	c50d2n10000000	50	2	10,000,000

k: # of clusters, *d*: # of dimensions, *n*: # of patterns.

low probability of changing their membership at later iterations. According to Table 2, RM1 outperforms RM2 in terms of the accuracy rate; however, RM2 outperforms RM1 in terms of the computation time. Therefore, for all experiments described in Section 5, RM1 is used because we want to reduce the computation time while at the same time limiting the loss of quality. However, if the loss of quality is not of major concern, RM2 provides an alternative means of reducing even more of the computation time. It is worth pointing out that the time differences as shown in Tables 1 and 2 are generally expected because all experiments are performed independently.

Since RM1 uses μ and σ of each cluster to determine patterns near their cluster center, determining patterns to be compressed and removed takes $O(n)$ time rather than the lower bound of all comparison-based sorting algorithms $O(n\log n)$. Although incapable of locating exactly α% of the patterns, using μ and σ is an efficient means of identifying patterns to be compressed and removed. Obviously, if the distances between patterns and means are not normally distributed, PR can always use other statistical measures, such as median, to locate patterns that are close to means.

4.2.3. Removal bound

Several removal bound settings—from 10% to 100% with an increment of 10%—have also been tested. The experimental results indicate that setting the removal bound to a larger value allows a larger number of patterns to be removed and thus a larger amount of computation time to be reduced. However, doing so may incur a larger loss of quality. Alternatively, setting the removal bound to a smaller value allows a smaller number of patterns to be removed and thus a smaller amount of computation time to be reduced. Fortunately, doing so incurs a smaller loss of quality. Our experimental results show that setting the removal bound to 80% yields the best result. Although setting the removal bound to 90% or higher can reduce 80% or more of the computation time, the accuracy rate decreases. Setting the removal bound to 80% appears to provide a good balance between the computation time and the accuracy rate. Consequently, for all experiments described in Section 5, the removal bound is set to 80%.

Table 4
Experimental results of small and high-dimensional data sets.

Data set	PR-KM			PR-RKM			PR-TKM			PR-KKM			PR-GKA		
	Δ_D	Δ_T	R	Δ_D	Δ_T	R	Δ_D	Δ_T	R	Δ_D	Δ_T	R	Δ_D	Δ_T	R
DSS1	−0.72	−5.56	–	−4.04	−3.45	–	0.18	10.34	57.44	0.18	0.00	0.00	0.45	−39.36	−87.47
DSS2	2.10	−2.56	−1.22	31.21	2.21	0.07	4.20	0.35	0.08	4.20	0.00	0.00	0.00	−36.28	–
DSS3	3.43	−24.79	−7.23	0.16	3.21	20.06	5.18	−24.55	−4.74	5.18	−28.57	−5.52	0.13	−45.73	−351.77
DSS4	−0.17	−49.07	–	15.25	−35.16	−2.31	0.90	−23.16	−25.73	0.90	−40.00	−44.44	0.68	−62.68	−92.18
DSH1	0.99	−76.26	−77.03	2.14	−34.10	−15.93	2.71	−67.36	−24.86	2.34	−81.80	−34.96	9.11	−61.78	−6.78
DSH2	0.74	−79.05	−106.82	1.49	−40.63	−27.27	1.14	−80.08	−70.25	0.94	−81.67	−86.88	3.27	−69.89	−21.37
DSH3	0.62	−79.37	−128.02	1.25	−42.35	−33.88	0.67	−80.66	−120.39	0.55	−81.26	−147.75	2.45	−72.11	−29.43
DSH4	0.63	−76.60	−121.59	0.96	−44.50	−46.35	0.42	−80.58	−191.86	0.33	−81.97	−248.39	2.72	−73.26	−26.93
DSH5	0.47	−76.53	−162.83	0.68	−45.36	−66.71	0.24	−80.19	−334.13	0.19	−82.22	−432.74	0.70	−73.54	−105.06
DSH6	0.23	−72.14	−313.65	0.07	−47.58	−679.71	0.09	−77.68	−863.11	0.07	−80.49	−1149.86	0.09	−73.53	−817.00
DSH7	0.13	−65.48	−503.69	0.32	−49.70	−155.31	0.03	−72.38	−2412.67	0.03	−76.53	−2551.00	0.88	−72.64	−82.55
DSH8	0.04	−56.51	−1412.75	0.26	−50.94	−195.92	0.01	−65.98	−6598.00	0.01	−70.99	−7099.00	0.49	−73.19	−149.37

Note that – means R is undefined.

5. Empirical analysis

As mentioned earlier, this work evaluates the performance of PR by applying it to five state-of-the-art clustering algorithms: standard k-means [45], relational k-means [49], kernel k-means [52,53], triangle inequality k-means [14], and genetic k-means algorithm [31]. The empirical analysis is conducted on an IBM X3400 machine with 2.0 GHz Xeon CPU and 8 GB of memory running CentOS 5.0 with Linux 2.6.18. Also, the programs are written in Java 1.5.0_09.

To simplify the discussion of the experimental results in Tables 4 and 5 and Fig. 12, the following conventions are adopted. Let KM denote standard k-means; RKM relational k-means; TKM triangle inequality k-means; KKM kernel k-means; and GKA genetic k-means algorithm. Let ψ denote either KM, RKM, TKM, KKM, or GKA. Let D and T denote, respectively, the quality of the clustering result in terms of **SSE** and the computation time. Let β denote either D or T. The enhancement of β "the original algorithm with PR" makes, denoted $\beta_{\text{PR}-\psi}$, with respect to β of "the original algorithm," denoted β_{ψ}, in percentage can be expressed as

$$\Delta_\beta = \frac{\beta_{\text{PR}-\psi} - \beta_\psi}{\beta_\psi} \times 100\%. \tag{6}$$

For instance, the enhancement of the computation time PR-KM makes ($T_{\text{PR-KM}}$) with respect to the computation time of KM (T_{KM}) is defined as

$$\Delta_T = \frac{T_{\text{PR-KM}} - T_{\text{KM}}}{T_{\text{KM}}} \times 100\%,$$

where ψ = KM and β = T. Moreover, it is important to note that as defined above in Eq. (6), a more negative value of Δ_β implies a greater enhancement.

Moreover, to measure the rate of reduction of computation time in comparison with the rate of loss of quality, we define the following metric

$$R = \frac{\Delta_T}{\Delta_D}, \tag{7}$$

where $\Delta_T < 0$ denotes the percentage of computation time reduced and $\Delta_D > 0$ represents the percentage of quality lost. In other words, R indicates for every percent loss of quality, what percentage of computation time is reduced. As defined above, a more negative value of R implies a greater reduction of computation time. It is important to note that R is just an indication of how well the proposed algorithm performs instead of stating that the rate of reduction of computation time can exceed 100%. Moreover, R is undefined in the case of a gain in quality ($\Delta_D \leqslant 0$) or an increase in computation time ($\Delta_T \geqslant 0$).

5.1. Data sets and parameter settings

As depicted in Table 3, for comparison, the test data sets are divided into two categories: real and synthetic. The real data sets include Reuters-21578 [38], KDD-98 Data Set [4], 20 News Groups [46], Iris [19], and Synthetic control chart time series (uci-sc) [1]. The synthetic data sets are further divided into the small data sets, which include 579 [56] and 800 [57]; the high-dimensional data sets (from 2 to 1,000 dimensions), which include DSH1 to DSH8; and the large data sets (from 60,000 to 10,000,000 patterns), which include DSL1 to DSL4.

Furthermore, all experiments are carried out for 30 runs and 30 iterations per run, and 10% of the input patterns are randomly sampled to create the initial solution. Also, as noted earlier, for all of the experiments, TS1 is used as the time to start strategy; RM1 the removal strategy; 80% the removal bound.

Table 5
Experimental results of real and large data sets.

Data set	PR-KM			PR-RKM			PR-TKM		
	Δ_D	Δ_T	R	Δ_D	Δ_T	R	Δ_D	Δ_T	R
DSR1	0.26	−77.83	−299.35	2.71	−36.17	−13.35	0.83	−79.85	−96.20
DSR2	6.01	−73.51	−12.23	1.08	−43.32	−40.11	2.76	−71.22	−25.80
DSR3	0.60	−76.85	−128.08	0.42	−33.74	−80.33	0.22	−75.49	−343.14
DSL1	1.74	−69.57	−39.98	1.00	−31.04	−31.04	2.26	−64.76	−28.65
DSL2	2.85	−70.01	−24.56	−2.59	−35.73	–	2.05	−65.07	−31.74
DSL3	4.51	−69.33	−15.37	−4.17	−35.92	–	3.72	−65.36	−17.57
DSL4	2.57	−69.37	−26.99	−5.22	−36.63	–	2.47	−65.16	−26.38

Note that – means R is undefined.

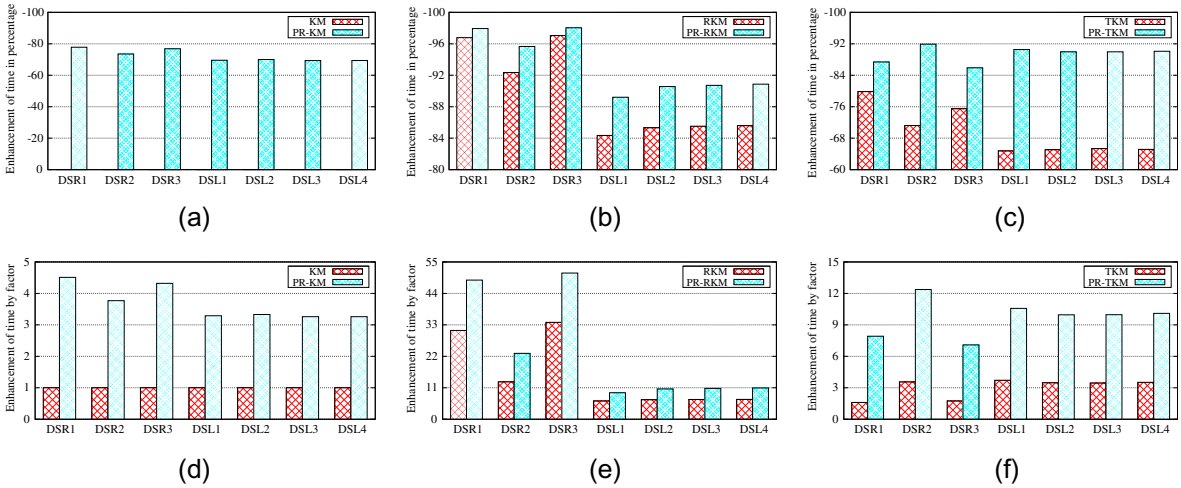

Fig. 12. Experimental results of real and large data sets. (a), (b), and (c) show the results by percentage; (d), (e), and (f) by factor. Note that all the results given in (a) through (f) are with respect to the results of KM given in (a) and (d).

5.2. Experimental results

Table 4 evaluates the performance of PR, by applying it to KM, RKM, TKM, KKM, and GKA and then using them to cluster DSSs and DSHs. As depicted in Table 4, a larger number of dimensions generally implies a smaller loss of quality. Moreover, PR can reduce the computation time of standard k-means by up to 79.37%, relational k-means by up to 50.94%, triangle inequality k-means by up to 80.66%, kernel k-means by up to 82.22%, and genetic k-means algorithm by up to 73.54%, respectively. It is interesting to note that although relational k-means [49] and triangle inequality k-means [14] can reduce the computation time of standard k-means by more than 84% and 64%, respectively, PR can further reduce their computation time by up to 50.94% and 80.66%, respectively, as given in Table 4.

Finally, for completeness, the performance of PR-KM, PR-RKM, and PR-TKM on clustering two large data sets are compared. Table 5 and Fig. 12 show that PR can further reduce the computation time of relational k-means [49] and triangle inequality k-means [14], despite their ability to reduce a significant amount of computation time of standard k-means. In addition, Tables 4 and 5 show the rate of reduction of computation time in comparison with the rate of loss of quality as defined in Eq. (7); that is, for every percent loss of quality, what percentage of computation time is reduced. In case R is undefined, i.e., $\Delta_D \leqslant 0$ or $\Delta_T \geqslant 0$, a dash is given.

According to the experimental results of DSR3 (20 News Groups) in Fig. 12, relational k-means by itself can reduce the computation time of standard k-means by 97.05% or by a factor of 33.88. However, Table 5 indicates that relational k-means with PR can further reduce the computation time of relational k-means by 33.74%. In other words, relational k-means with PR can reduce the computation time of standard k-means by 98.04% or by a factor of 51.13.

Together, Tables 4 and 5 and Fig. 12 indicate that with a small loss of quality, PR can reduce the computation time of all of the clustering algorithms evaluated in this work. More important, they indicate that PR can be applied not only to single-solution-based algorithms such as k-means-based algorithms but also to population-based and kernel-based algorithms such as GKA and kernel k-means to reduce their computation time.

5.3. Time complexity

As described in [28,11], the time complexity of k-means is $O(nk\ell)$, where n denotes the number of patterns; k the number of clusters; and ℓ the number of iterations. This analysis is limited in that it did not take into account the number of dimensions of the input patterns. As is well known, most commercially available computer architectures such as personal computers can not add or subtract two 1000-dimensional vectors in exactly the same number of cycles as adding or subtracting two scalars does. Thus, a more reasonable time complexity analysis of the k-means algorithm would be $O(nkd\ell)$, which considers the number of dimensions of the input patterns. This observation corresponds to the time complexity of **SSE** given by Krishna [31] and Lu [43,44]. According to their results, the time complexity of **SSE** is $O(nd)$ where n denotes the number of patterns and d the number of dimensions of the input patterns. That observation also corresponds to our experimental results, which indicate that the computation time is proportional to the number of dimensions of the input patterns. Therefore, in the following analysis, the time complexity of k-means is assumed to be $O(nkd\ell)$. Theoretically, the PR algorithm can reduce the time complexity of k-means from $O(nkd\ell)$ to $O(nkd)$. This can be easily proved by letting Δ $(0 < \Delta < 1)$ be a constant that denotes the percentage of patterns retained at each iteration. In other words, $1 - \Delta$ denotes the percentage of patterns removed at each iteration. Then,

$$\sum_{i=0}^{\ell-1} \Delta^i nkd = nkd \sum_{i=0}^{\ell-1} \Delta^i \leqslant nkd \sum_{i=0}^{\infty} \Delta^i = nkd \frac{1}{1-\Delta} = O(nkd). \tag{8}$$

However, our experimental results indicate that in practice, a removal bound is required to limit the number of patterns that can be compressed and removed by the PR algorithm. The purpose of this bound is to reduce the extent to which noise in the input data impacts the accuracy rate of the clustering result. For the results described earlier, the bound is set to 80% because we do not know exactly how k-means or k-means-based algorithms converge. If the PR algorithm removes too many patterns at each iteration, the accuracy rate will decrease. Therefore, all we can claim about the time complexity of k-means with PR is that it falls somewhere between $O(nkd)$ and $O(nkd\ell)$. In other words, the time complexity of k-means with PR is bounded from above by $O(nkd\ell)$ and from below by $O(nkd)$. In the best case, when the PR algorithm is started at the first iteration and the removal bound is set to 100%, the time complexity of k-means with PR is $O(nkd)$. In the worst case, when k-means with PR converges before the PR algorithm is started or the removal bound is set to 0%, then k-means with PR falls back to k-means, and the time complexity is $O(nkd\ell)$. Moreover, according to our experimental results of Iris, 400, 579, and 800, k-means with PR converges to the local minima approximately 41% faster than k-means on average. This is consistent with the time complexity of PR, which is somewhere between $O(nkd)$ and $O(nkd\ell)$. Overall, the time complexity and convergence speed of k-means with PR depend on (1) the iteration at which the PR algorithm is started, (2) the number of patterns removed at each iteration, and (3) the removal bound. Our experimental results indicate that for complex data sets, PR can reduce approximately 70% of the computation time of k-means when the removal bound is set to 80%. The above results further demonstrate that if the removal bound is set to a larger value at the right time, the time complexity of k-means can be reduced to approach that of the ideal case, i.e., $O(nkd)$.

6. Conclusions

Inspired by the observation that many computations of k-means and k-means-based clustering algorithms are essentially redundant, this work presents an efficient algorithm, called pattern reduction (PR), to accelerate their performance. The proposed algorithm works by compressing and removing at each iteration patterns that are unlikely to change their membership later. Our experimental results indicate that the proposed algorithm can significantly reduce the computation time of k-means and k-means-based algorithms. This corresponds to the time complexity of PR. In addition, the strategies of PR are analyzed to more thoroughly understand their impact on the performance of PR. Moreover, the convergence speed of k-means with PR is analyzed. Efforts are underway to apply the proposed algorithm to other clustering algorithms in order to enhance their performance.

Acknowledgments

The authors thank the editors and the anonymous reviewers for their constructive comments and suggestions on the paper. This work was supported in part by National Science Council, Taiwan, ROC, under Contract Nos. NSC99-2221-E-110-052 and NSC98-2811-E-006-078.

References

[1] R. Alcock, Synthetic control chart time series, 1999. Available from: <http://kdd.ics.uci.edu/databases/synthetic_control/synthetic_control.html>.
[2] R.M. Aliguliyev, Performance evaluation of density-based clustering methods, Information Sciences 179 (20) (2009) 3583–3602.
[3] S. Bandyopadhyay, U. Maulik, An evolutionary technique based on k-means algorithm for optimal clustering in R^N, Information Sciences 146 (1–4) (2002) 221–237.
[4] S.D. Bay, KDD Cup 1998 data, 2000. Available from: <http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html>.
[5] M. Bereta, T. Burczynski, Immune k-means and negative selection algorithms for data analysis, Information Sciences 179 (10) (2009) 1407–1425.

[6] P.S. Bradley, U.M. Fayyad, Refining initial points for *k*-means clustering, in: Proceedings of the International Conference on Machine Learning, 1998, pp. 91–99.
[7] P.S. Bradley, U.M. Fayyad, C. Reina, Scaling clustering algorithms to large databases, Knowledge Discovery and Data Mining (1998) 9–15.
[8] M.P.S. Brown, W.N. Grundy, D. Lin, N. Cristianini, C.W. Sugnet, T.S. Furey, M. Ares, D. Haussler, Knowledge-based analysis of microarry gene expression data by using support vector machines, in: Proceedings of the National Academy of Sciences of the United States of America, 97,1, 2000, pp. 262–267.
[9] F. Camastra, A. Verri, A novel kernel method for clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (5) (2005) 801–804.
[10] X. Cui, T.E. Potok, P. Palathingal, Document clustering using particle swarm optimization, in: Proceedings of the IEEE Swarm Intelligence Symposium, 2005, pp. 185–191.
[11] W.H.E. Day, Complexity theory: an introduction for practitioners of classification, in: P. Arabie, L. Hubert (Eds.), Clustering and Classification, World Scientific Publishing Co., Inc., River Edge, NJ., 1992.
[12] M.B. de Almeida, A. de Padua Braga, J.P. Braqa, Svm-km: speeding svms learning with a priori cluster selection and *k*-means, in: Proceedings of Sixth Brazilian Symposium on Neural Networks, 2000, pp. 162–167.
[13] I.S. Dhillon, Y. Guan, B. Kulis, Kernel *k*-means, spectral clustering and normalized cuts, in: Proceedings of the Tenth ACM SIGKDD, 2004, pp. 551–556.
[14] C. Elkan, Using the triangle inequality to accelerate *k*-means, in: Proceedings of the Twentieth International Conference on Machine Learning, 2003, pp. 147–153.
[15] S. Eschrich, J. Ke, L.O. Hall, D.B. Goldgof, Fast accurate fuzzy clustering through data reduction, IEEE Transactions on Fuzzy Systems 11 (2) (2003) 262–270.
[16] M. Ester, H. Peter Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of International Conference on Knowledge Discovery and Data Mining, 1996, pp. 226–231.
[17] F. Farnstrom, J. Lewis, C. Elkan, Scalability for clustering algorithms revisited, SIGKDD Explorations 2 (1) (2000) 51–57.
[18] P. Ferragina, A. Gulli, A personalized search engine based on web-snippet hierarchical clustering, in: International World Wide Web Conference (WWW2005), 2005, pp. 801–810.
[19] R.A. Fisher, Iris data set, 1936. Available from: <http://archive.ics.uci.edu/ml/datasets/Iris>.
[20] T. Fromherz, P. Stucki, M. Bichsel, A survey of face recognition, MML Technical Report, 1997.
[21] G. Getz, H. Gal, I. Kela, D.A. Notterman, E. Domany, Coupled two-way clustering analysis of breast cancer and colon cancer gene expression data, Bioinformatics 19 (2003) 12079–12084.
[22] A. Ghosh, A. Halder, M. Kothari, S. Ghosh, Aggregation pheromone density based data clustering, Information Sciences 178 (13) (2008) 2816–2831.
[23] F. Giannotti, M. Nanni, D. Pedreschi, F. Samaritani, Webcat: automatic categorization of web search results, in: Proceedings of SEBD'2003, 2003, pp. 507–518.
[24] S. Guha, A. Meyerson, N. Mishra, R. Motwani, Clustering data streams: theory and practice, IEEE Transactions on Knowledge and Data Engineering 15 (3) (2003) 515–528.
[25] K.M. Hammouda, M.S. Kamel, Efficient phrase-based document indexing for web document clustering, IEEE Transactions on Knowledge and Data Engineering 16 (10) (2004) 1279–1296.
[26] C. Hua, Q. Chen, H. Wu, T. Wada, RK-means clustering: *k*-means with reliability, IEICE Transactions on Information and Systems E91-D (1) (2008) 96–104.
[27] A.K. Jain, R.C. Dubes, Algorithms for Clustering Data, Prentice Hall, Englewood Cliffs, NJ, 1988.
[28] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review, ACM Computing Surveys 31 (3) (1999).
[29] M.F. Jiang, S.S. Tsent, C.M. Su, Two-phase clustering process for outliers detection, Pattern Recognition Letters 22 (6–7) (2001) 691–700.
[30] J. Kogan, Introduction to Clustering Large and High-Dimensional Data, Cambridge University Press, NY, USA, 2007.
[31] K. Krishna, M.N. Murty, Genetic *k*-means algorithm, IEEE Transactions on System, Man and Cybernetics – Part B:Cybernetics 29 (3) (1999) 433–439.
[32] R.J. Kuo, L.M. Ho, C.M. Hu, Integration of self-organizing feature map and *k*-means algorithm for market segmentation, International Journal of Computers and Operations Research 29 (11) (2002) 1475–1493.
[33] R.J. Kuo, H.S. Wang, T.-L. Hu, S.H. Chou, Application of ant *k*-means on clustering analysis, Computers and Mathematics with Applications 50 (10–12) (2005) 1709–1724.
[34] M. Laszio, S. Mukherjee, A genetic algorithm using hyper-quadtrees for low-dimensional *k*-means clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (4) (2006) 533–543.
[35] H. Lawrence, Face recognition: a literature survey, Review of Financial Studies 4 (3) (1991) 389–415.
[36] C.-H. Lee, O.R. Zäiane, H.-H. Park, J. Huang, R. Greiner, Clustering high dimensional data: a graph-based relaxed optimization approach, Information Sciences 178 (23) (2008) 4501–4511.
[37] A. Leuski, Evaluating document clustering for interactive information retrieval, in: Proceedings of the Tenth International Conference on Information and Knowledge Management, 2001, pp. 33–40.
[38] D.D. Lewis, Reuters-21578 text categorization collection, 1987. Available from: <http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html>.
[39] Y. Li, S.M. Chung, Parallel bisecting *k*-means with prediction clustering algorithm, Journal of Supercomputing 39 (1) (2007) 19–37.
[40] A. Likas, N. Vlassis, J.J. Verbeek, The global *k*-means clustering algorithm, Pattern Recognition 36 (2) (2003) 451–461.
[41] B. Liu, Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications), Springer, 2007.
[42] Y. Liu, Z. Yi, H. Wu, M. Ye, K. Chen, A tabu search approach for the minimum sum-of-squares clustering problem, Information Sciences 178 (12) (2008) 2680–2704.
[43] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, S.J. Brown, Fgka: A fast genetic *k*-means clustering algorithm, in: Proceedings of Symposium on Applied Computing, 2004, pp. 622–623.
[44] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, S.J. Brown, Incremental genetic *k*-means algorithm and its application in gene expression data analysis, International Journal of BMC Bioinformatics 5 (172) (2004).
[45] J.B. McQueen, Some methods of classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.
[46] T. Mitchell, 20 newsgroups, 1999. Available from: <http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html>.
[47] M.K. Ng, J.C. Wong, Clustering categorical data sets using tabu search techniques, Pattern Recognition 35 (12) (2002) 2783–2790.
[48] R.T. Ng, J. Han, Clarans: a method for clustering objects for spatial data mining, IEEE Transactions on Knowledge and Data Engineering 14 (5) (2002) 1003–1016.
[49] C. Ordonez, E. Omiecinski, Efficient disk-based *k*-means clustering for relational databases, IEEE Transactions on Knowledge and Data Engineering 16 (8) (2004) 909–921.
[50] N.R. Pal, J.C. Bezdek, On cluster validity for the fuzzy *c*-means model, IEEE Transactions on Fuzzy System 3 (3) (1995) 370–379.
[51] D. Pelleg, A.W. Moore, *X*-means: extending *k*-means with efficient estimation of the number of clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, 2000, pp. 727–734.
[52] B. Schölkopf, A. Smola, K.-R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, Neural Computation 10 (1998) 1299–1319.
[53] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.
[54] M. Steinbach, G. Karypis,V. Kumar, A comparison of document clustering techniques, in: Sixth ACM SIGKDD International Conference on Data Mining (KDD'00), 2000, pp. 109–110.
[55] M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques, University of Minnesota Technical Report #00-034, 2000.
[56] M.-C. Su, H.-T. Chang, Fast self-organizing feature map algorithm, IEEE Transactions on Neural Networks 11 (3) (2000) 721–733.

[57] C.-F. Tsai, Z.-C. Chen, C.-W. Tsai, MSGKA: an efficient clustering algorithm for large databases, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, vol. 5, 2002.
[58] D. van der Merwe, A.P. Engelbrecht, Data clustering using particle swarm optimization, in: Proceedings of the 2003 Congress on Evolutionary Computation, 2003, pp. 215–220.
[59] J. Vesanto, E. Alhoniemi, Clustering of the self-organizing map, IEEE Transactions on Neural Networks 11 (3) (2000) 586–600.
[60] R.W. Zhao, R. Chellappa, P.J. Philips, A. Rosenfeld, Face recognition: a literature survey, ACM Computing Surveys (CSUR) 35 (4) (2003) 399–458.
[61] W. Wang, J. Yang, R. Muntz, STING: a statistical information grid approach to spatial data mining, in: Proceedings of Conference on Very Large Databases, 1997, pp. 186–195.
[62] R. Xu, D. Wunsch II, Survey of clustering algorithms, IEEE Transaction on Neural Netowrks 16 (3) (2005) 645–678.
[63] R. Xu, D.C. Wunsch, Clustering, Wiley, John & Sons, Inc., 2008.
[64] W. Xu, X. Liu, Y. Gong, Document clustering based on non-negative matrix factorization, in: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003, pp. 267–273.
[65] H.J. Zeng, Q.C. He, Z. Chen, W.Y. Ma, J. Ma, Learning to cluster web search results, in: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'04), 2004, pp. 210–217.
[66] R. Zhang, A.I. Rudnicky, A large scale clustering scheme for kernel *k*-means, in: 16th International Conference on Pattern Recognition, 4, 2002, p. 40289.
[67] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: an efficient data clustering method for very large databases, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 1996, pp. 103–114.

Ming-Chao Chiang received the B.S. degree in Management Science from National Chiao Tung University, Hsinchu, Taiwan in 1978 and the M.S., M.Phil., and Ph.D. degrees in Computer Science from Columbia University, New York, NY, U.S.A. in 1991, 1998, and 1998, respectively. He had over 12 years of experience in the software industry encompassing a wide variety of roles and responsibilities in both large and start-up companies before joining the faculty of the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan in 2003, where he is currently an Assistant Professor. His current research interests include image processing, evolutionary computation, and system software.

Chun-Wei Tsai was born in 1978. He received the M.S. degree in Management Information System from National Pingtung University of Science and Technology, Pingtung, Taiwan in 2002, and the Ph.D. degree in Computer Science from National Sun Yat-sen University, Kaohsiung, Taiwan in 2009. He was a postdoctoral fellow with the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan before joining the faculty of the Department of Applied Geoinformatics, Chia Nan University of Pharmacy & Science, Tainan, Taiwan in 2010, where he is currently an Assistant Professor. His research interests include web information retrieval, evolutionary computation, and combinatorial optimization.

Chu-Sing Yang received the B.S. degree in Engineering Science, and the M.S. and Ph.D. degrees in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan in 1976, 1984 and 1987, respectively. He joined the faculty of the Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, as an Associate Professor in 1988. Since 1993, he has been a professor of the Department of Computer Science and Engineering, National Sun Yat-sen University. He was a Chair of Department of Computer Science and Engineering, National Sun Yat-sen University from August 1995 to July 1999. He was a director of Computer Center, National Sun Yat-sen University from August 1998 to October 2002. He was a Program Chair of ICS-96 and Program Co-Chair of ICPP-2003. He joined the faculty of the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, as a Professor in 2006. His research interests include embedded system, network management, web mining, and wireless communication.