J Heuristics (2006) 12:447–466
DOI 10.1007/s10732-006-7284-z

A genetic *k*-medoids clustering algorithm

Weiguo Sheng · Xiaohui Liu

Submitted in December 2004 and accepted by Zbigniew Michalewicz in January 2005 after 2 revisions
© Springer Science + Business Media, LLC 2006

Abstract We propose a hybrid genetic algorithm for *k*-medoids clustering. A novel heuristic operator is designed and integrated with the genetic algorithm to fine-tune the search. Further, variable length individuals that encode different number of medoids (clusters) are used for evolution with a modified Davies-Bouldin index as a measure of the fitness of the corresponding partitionings. As a result the proposed algorithm can efficiently evolve appropriate partitionings while making no *a priori* assumption about the number of clusters present in the datasets. In the experiments, we show the effectiveness of the proposed algorithm and compare it with other related clustering methods.

Keywords *k*-Medoids clustering · Genetic algorithms · Heuristics · Cluster validity · Davies-Bouldin index

Clustering is a tool commonly used in the data analysis. The clustering problem comes down to finding the separation of a set of objects into an *a priori* unknown number of subsets while minimizing intra-cluster variation and maximizing the inter-cluster variation. There is a large amount of literature on this subject, ranging from models, algorithms, algorithm parameter estimations to cluster validity studies (Hartigan, 1975; Jain and Dubes, 1988; Duda et al., 2001). Generally, the clustering methods can be classified into two categories (Lance and Williams, 1967): exclusive (hard) clustering where each data object belongs to only one cluster, and nonexclusive (fuzzy) methods where every data object belongs to every cluster to a certain degree. We concentrate on exclusive clustering methods, among which the *k*-centroids and *k*-medoids models are the most widely studied.

K-centroids clustering model, which adopts the mean of the corresponding cluster as the representative, aims to find *k* centroids such that, if every object in the data set is "assigned"

W. Sheng (✉)
Department of Electronics, University of Kent, Canterbury, Kent, CT2 7NT, UK
e-mail: w.sheng@kent.ac.uk

X. Liu
Department of Information System and Computing, Brunel University, UB8 3PH, London, UK
e-mail: xiaohui.liu@brunel.ac.uk

to its closest centroid, a specific clustering criterion is optimized, where the objects typically come from a real space R^d. K-medoids clustering model differs from the above in that instead of finding k centroids, the goal is to choose k members of the data set, called "medoids". In (Kaufman and Rousseeuw, 1990), a medoid was defined as the object of the cluster for which the average dissimilarity to all the objects of the cluster is minimized. In this paper, the k-medoids model will be explored. In comparison with k-centroids model, the k-medoids model has been shown to be very robust to the existence of noise or outliers and generally produces clusters of high quality (Kaufman and Rousseeuw, 1990; Ng and Han, 2002). Further more, the use of medoids for clustering is more general in that it is also applicable to situations where the objects to be clustered cannot be represented by numerical values or the mean of the objects is not defined.

Since clustering is a known NP-hard problem (Garey and Johnson, 1979), most approaches use the alternative optimization schemes in order to find a local optimum solution of their criterion function. Many stochastic optimization schemes that aim at global optimum have been reported, among which are the genetic algorithms (GAs) (Holland, 1975). GAs are search procedures that use the mechanics of evolution and natural genetics. They have been earlier reported for the clustering problem (Lucasius et al., 1993; Murthy and Chowdhury, 1996; Scheunders, 1997; Estivill-Castro and Murray, 1997; Falkenauer, 1998; Hall et al., 1999; Krishna, 1999; Maulik and Bandyopadhyay, 2000; Bandyopadhyay and Maulik, 2000; Hruschka and Nelson, 2003; Yi, 2004). However, they may take a large amount of time to converge and/or assume the number of clusters to be fixed *a priori*. Furthermore, most of these methods were proposed to optimize the k-centroids model. Accordingly, these methods may not be robust to the existence of noise or outliers in the data set.

In this paper, we suggest a hybrid genetic algorithm for k-medoids clustering called genetic k-medoids clustering algorithm (GMCA) that can efficiently evolve appropriate partitionings while making no *a priori* assumption about the number of clusters present in the datasets. For these purposes, a novel heuristic operator is designed and integrated with the global genetic search to improve the computational efficiency. The heuristic operator tries to efficiently improve the fitness of the offspring by updating the medoids encoded in them such that the total dissimilarities or distances within each cluster represented by the medoid is minimized. Further, variable length individuals that encode different number of medoids (clusters) are used for evolution with a modified version of the Davies-Bouldin validity index (Davies and Bouldin, 1979) as a measure of the fitness of the corresponding partitionings.

The outline of the paper is as follows. After defining the k-medoids clustering problem in Section 1, we briefly review related methods in Section 2. In Section 3, we present the details of the proposed algorithm—genetic k-medoids clustering algorithm (GMCA). In Section 4, data sets employed in this work are described, followed by the discussion of parameter settings of the GMCA. In Section 5, we demonstrate the effectiveness of GMCA and compare it with other related clustering methods. Finally, Section 6 contains our summary and discussion.

1 *K*-Medoids clustering problem

Given a data set $X = \{x_1, x_2, \ldots, x_n\}$ with n data objects, clusters of the k-medoids model are generated by selecting a set of k members of X as "medoids", and assigning each non-selected member of X to its closest medoid. A valid k-medoids clustering of X in a set of clusters $C = \{C_1, C_2, \ldots, C_k\}$ has the following partitioning properties:

1. $C_i \neq \emptyset,\ 1 \leq i \leq k$;

2. $\cup_{i=1}^{k} C_i = X$;
3. $C_i \cap C_j = \emptyset, i \neq j,\ 1 \leq i, j \leq k$.

Additionally, the clusters should reflect the structure of the data such that objects in the same cluster are similar to each other and objects from distinct clusters are different from each other. In order to find a solution to the k-medoids clustering problem, we need a quantitative way to distinguish between similar and dissimilar objects. In the literature various alternatives have been reported to approach this task (Jain and Dubes, 1988; Duda et al., 2001). A commonly used criterion is the sum of intra-cluster variation (SICV)

$$\text{SICV} = \sum_{i=1}^{n} \sum_{j=1, x_i \in C_j}^{k} d(x_i, m_j), \tag{1}$$

where $m_j \subset X$ represents the jth cluster medoid and $d(x_i, m_j)$ is the dissimilarity or distance between ith object x_i and jth cluster medoid m_j. Without additional constraints, Eq. (1) has a zero minimum for $k = n$. Therefore, this criterion is only applicable to the situation that the number of clusters is known *a priori.*

2 Related work

There are several well-known methods for k-medoids clustering in the literature. One of the most popular is partitioning around medoids (PAM) from Kaufman and Rousseeuw (1990). This algorithm requires the number of clusters, k, be to known *a priori*. To find the k medoids, PAM begins with an arbitrary selection of k medoids. Then, in each step, a swap between a selected object and a non-selected object is made, as long as such a swap would result in improvement of the quality of the solution. The PAM algorithm has the complexity of $O((1+\alpha)k(n-k)^2)$, where α is the number of successful swaps. Obviously, it is time consuming even for a moderate number of objects and a moderate number of clusters. A modified version of PAM called CLARA (clustering large applications) to handle large data sets was also proposed by Kaufman and Rousseeuw (1990). The CLARA is a combination of a sampling technique and the PAM algorithm. Instead of finding k medoids for the entire data set, the CLARA draws multiple samples of the data set, and applies PAM on each sample. The final solution is obtained from the best result of these multiple draws. Ng and Han (2002) proposed another variation of the CLARA called CLARANS, in which the clustering process is formalized as searching through a graph where each node is represented by a set of k medoids. Instead of using an exhaustive search strategy, the CLARANS adopts a serial randomized search. Both the CLARA and CLARANS have been proved to be much more efficient than PAM. However, they also assume the number of clusters to be known *a priori*.

Since clustering is NP-hard, genetic algorithms (GAs) that aim at a global optimum have been introduced to handle the problem. Most of the literature on GA based clustering methods pertains to optimize the k-centroids model (Scheunders, 1997; Hall et al., 1999; Krishna, 1999; Maulik and Bandyopadhyay, 2000; Bandyopadhyay and Maulik, 2000; Yi, 2004). Accordingly, these methods may not be robust to the existence of noise or outliers in the data set. On the other hand, GA clustering methods proposed to optimize the k-medoids model can be more robust. Lucasius et al. (1993) proposed a method called genetic clustering algorithm (GCA) to study the feasibility of using a GA for the k-medoids clustering. In the GCA, a GA based on the roulette selection, recombination crossover with a build-in mutation,

flip mutation (Goldberg, 1989) and fixed length coding was used with the inverse of (1) as the fitness function for the *k*-medoids clustering problem. The experimental results of the GCA are promising. However, it may take a large amount of time to converge. Estivill-Castro and Murray (1997) recommended another method called RAR_w-GA. In the RAR_w-GA, a similar GA as in the GCA was used, however, with a crossover method called "weak level of random assorted recombination (RAR_w)", which offers respect (i.e., preserving common characteristics of parents in the offspring) and proper assortment (i.e., all combinations of compatible characteristics present in the parents most have a non-zero probability of being present in the offspring) to the genetic search. RAR_w uses a parameter w to improve assortment. Like the GCA, the RAR_w-GA is still a computational intensive method to cluster large data sets.

All of the GA based clustering approaches mentioned above require an *a priori* specification of the number of clusters. Comparatively, not much work has been done for clustering that requires no such *a priori* specification. In Falkenauer (1998), an algorithm called grouping genetic algorithm (GGA), which allows for the clustering of a data set into an *a priori* unknown number of clusters, is developed. The GGA applied an integer representation to form individuals with variable length, where the integers in the first part of the individuals represent the membership of data objects and there is additional part, formed by the cluster labels. To alleviate the problems of redundancy and context insensitivity of the above representation with traditional genetic operators, genetic operators in the GGA were devised to work with groups of allele genes, instead of with the allele genes themselves. These operators are referred to as group oriented. To improve the GGA, Eduardo and Nelson (2003) developed a clustering genetic algorithm (CGA). The CGA simplified the GGA by designing a crossover operator to work with individuals of fixed length. In both GGA and CGA, partitioning is obtained explicitly based on the value of integers. These algorithms allow for the clustering of a data set into an *a priori* unknown number of clusters. However, they may take a large amount of time to converge, particularly when they are used to cluster large data sets. Furthermore, the representation used in the GGA and CGA is not a scalable encoding of clustering large data sets due to the length of the individuals. A modified version of the CGA called evolutionary algorithm for clustering (EAC) is also proposed by Hruschka et al. (2004). EAC can be more efficient than the CGA. However, it is only applicable to situations where the objects to be clustered can be represented by numerical values. In this paper, we also consider the clustering problems without making any *a priori* assumption about the number of clusters. However, unlike the above approaches with no medoids provided in the clustering results, our algorithm supplies the medoids which can be very useful for many applications and the partitioning is obtained by assigning the data objects to the medoids. More importantly, a novel heuristic operator is designed and integrated with the genetic algorithm to improve the computational efficiency. This work is an enhanced version of the work in (Sheng and Liu, 2004), which requires the number of cluster to be known beforehand.

3 Proposed algorithm

In this section, we describe a hybrid genetic algorithm for *k*-medoids clustering called genetic *k*-medoids clustering algorithm (GMCA) that can efficiently evolve appropriate partitionings while making no *a priori* assumption about the number of clusters present in the datasets. The algorithm is described below in details.

3.1 Genetic representation and population initialization

In most of the genetic clustering methods, the binary and integer representations (Cucchiara, 1998; Falkenauer, 1998) are commonly used to represent the membership or permutation of data objects and cluster assignment is done explicitly based on the value of binary codes or integers. In most cases, they suffer from problems of context insensitivity (Falkenauer, 1998). Therefore, genetic operators have to be designed to alleviate this problem. In this work, we applied a natural and simple way of representation which allows each allele gene in the individual taking value from $\{1, 2, \ldots, n\}$. In this case, each allele gene corresponds to a medoid of the selected object and the length of the individual corresponds to the number of clusters. For example, the individual (5, 27, 178, 379, 576) encodes five clusters where the 5th, 27th, 178th, 379th and 576th objects are selected as medoids. This representation can be context sensitive with most traditional genetic operators.

Each individual i in the population initially encodes k_i clusters. The initial number of clusters k_i is calculated according to *RandInt* $(2, k_{\max})$. Here, *RandInt*$(2, k_{\max})$ is a function returning an integer in the range from 2 to $k_{\max}$ (inclusive), and $k_{\max}$ is the upper bound of the number of clusters and is taken to be $\sqrt{n}$, which is a rule of thumb used by many investigators in the literature (Pal and Bezdek, 1995). The number of clusters in the population will therefore range from 2 to $k_{\max}$. Cluster assignment is done implicitly based on dissimilarity or distance of the non-selected objects to the medoids. Only valid individuals that have no duplicate indexes are considered to be included in the initial population.

3.2 Fitness function

The fitness of an individual indicates the degree of suitability of the solution it represents. In this paper, we use a modified version of the Davies-Bouldin (DB) (Davies and Bouldin, 1979) validity index for this purpose. The DB index is the ration of the sum of within-cluster scatter to between-cluster separation. This function has been popularly used to validate the clusters. However, it requires that the mean of the clusters to be validated be defined. This feature restricts its application to validate the clusters generated by the k-medoids model, which is more general in that it is also applicable to situations where the mean of the clusters is not defined. Therefore, we modify the DB index as follows to validate clusters generated by the k-medoids model. To measure the scatter within the ith cluster, S_i, we replace the centroid by the medoid and write S_i as $S_i = \frac{1}{|C_i|}\sum_{x \in C_i} d(x, m_i)$, where $|C_i|$ is the number of objects in cluster C_i and m_i represents the ith cluster medoid. Similarly, we write the dissimilarity or distance between clusters C_i and C_j as $d_{ij} = d(m_i, m_j)$. Then we computer $R_i = \max_{j, j \neq i}\{(S_i + S_j)/d_{ij}\}$. Now the modified Davies-Bouldin (MDB) index is defined as

$$\text{MDB} = \frac{1}{k}\sum_{i=1}^{k} R_i, \tag{2}$$

where k is the number of clusters. Intuitively, clusters of a good solution should be compact and well separated, thereby yielding small value of the MDB index. The objective is therefore to minimize the MDB index to achieve appropriate partitionings.

The fitness calculation of an individual is based on the clusters formed according to the medoids encoded in the individual. For each individual, the medoids encoded in it are first extracted, and then a partitioning is obtained by assigning each non-selected object of the

data set to be clustered to its closest medoid. Given the above partitioning, we then calculate MDB value according to Eq. (2) for the individual. During our experiments, sometimes the individual may represent clusters with a single object (invalid individual). To make the invalid individual less likely to enter the population pool for the next generation, we penalize its MDB value with the following equation:

$$\mathrm{MDB}' = \mathrm{MDB} + b * \mathrm{MDB}, \tag{3}$$

where $b \in \{0, \ldots, k\}$ is the number of clusters with a single object. After that the fitness of the individual is then set to $f = 1/\mathrm{MDB}'$. Note that maximization of the fitness function will ensure minimization of the MDB index.

3.3 Genetic operators

Selection: The selection operator selects parent pairs from the previous population pool for further processing. This operator plays an important role of focusing the search effort on promising regions in the search space and controlling the speed of convergence. Traditional w-fold tournament selection chooses each parent by randomly selecting a group of individuals with size w from the population and choosing the fittest one to be the parent. The size w of tournament selection controls how quickly a population is "taken over" by dominant individuals. We experienced that even with the lowest size w (i.e., 2), the population may be still quickly "taken over" by dominant individuals and converge prematurely. Goldberg and Richardson (1987) showed that assigning exponential number of mating opportunities to those members of the population that exhibit above average survival traits is not a good strategy for a thorough exploration of complex search space. Therefore, we implement a scheme with even lower selection pressure as follows. One parent is selected by 2-fold tournament selection, and its mate is selected not by the tournament selection again, but randomly from the population. This selection scheme reduces the selection pressure of traditional tournament selection and aims for better performance.

Crossover: Crossover is a probabilistic process that exchanges information between a pair of parent to generate two offspring. The crossover operator, applied stochastically with a probability p_c, must ensure that information exchange takes place in such a way that both the offspring encode at least two but no more than $k_{\max}$ medoids. Furthermore, it should always avoid producing offspring with duplicate indexes or offspring identical to their parents. For these purposes, the operator is defined as follows for given a pair of parent individuals P_1 and P_2 with k_1 and k_2 medoids, respectively.

1. Mix P_1 and P_2 to obtain P_{mix} by appending one parent individual to another and then the medoids of resulting P_{mix} are randomly scrambled.
2. Generate the number of medoids with $k_3 = \mathit{RandInt}(\max(2, k_1 + k_2 - k_{\max}), \min(k_1 + k_2 - 2, k_{\max}))$ and $k_4 = k_1 + k_2 - k_3$ for two offspring P_3 and P_4, respectively. Here, $\min(k_1 + k_2 - 2, k_{\max})$ and $\max(2, k_1 + k_2 - k_{\max})$ are functions that return the lesser of the two numbers and larger of the two numbers, respectively. (It can be verified that if the number of medoids k_3 and k_4 are calculated according to above rules, then the offspring will have at least two but no more than $k_{\max}$ medoids.)
3. Build the offspring P_3 with k_3 medoids by coping features from P_{mix} starting at the leftmost feature and going feature-wise to the right, subject to the condition that features already in P_3 are skipped.

4. Build the offspring P_4 in the same way as P_3, however starting at the rightmost feature of P_{mix} and going feature-wise to the left.

Mutation: After crossover, a probability p_m of flip mutation will be applied to the offspring. Flip mutation replaces the chosen feature by another randomly generated feature, subject to the restriction that the new feature is not presented in the current individual. The mutation operator is useful to add new features to the offspring individuals and allow them to search other areas that have not been explored by the algorithm.

3.4 Heuristic operator

GAs are a powerful set of global search techniques that have been shown to produce very good results on a wide class of problems. GAs are capable of exploring and exploiting promising regions of the search space. They can, however, take a long time to converge. Efficient optimization algorithms used to solve hard problems usually combine global and local searches to find a near optimal solution to the problem being solved. These are commonly known as hybrid methods. The theory behind the hybrid methods is not well-founded. However, they have been used successfully to solve a wide variety of problems and experimental studies show hybrid methods may greatly improve the computational efficiency (Krishna, 1999; Smith, 2000; Areibi, 2004). In this work, a new heuristic operator is designed and integrated with the global genetic search for k-medoids clustering. This operator tries to efficiently improve the fitness of the offspring by updating the medoids encoded in them such that the total dissimilarities or distances within each cluster represented by the medoid is minimized. More precisely, the operator works as follows.

For each offspring, extract the medoids encoded in the offspring and set the number of clusters k equal to the total number of medoids, then assign each non-selected object of the data set to be clustered to its closest medoid to form a set of clusters $C = \{C_1, C_2, \ldots, C_k\}$;

For $i = 1$ to k do (update the k medoids)

(a) Within the cluster C_i, choose a subset C_{subset_i} that corresponds to m_i and its p nearest neighbors (which have not been evaluated before the current iteration);
(b) Store the current medoid: $m_{\text{old}} = m_i$;
(c) Compute the new medoid:

$$q = \arg\min_{x_s \in C_{\text{subset}_i}} \sum_{x_j \in C_i} d(x_s, x_j),$$

$$m_i = m_q.$$

(d) Repeat (a) to (c) until $m_{\text{old}} = m_i$;

Endfor

For each offspring resulted from regeneration steps, the heuristic operator first assigns each non-selected object of the data set to be clustered to the cluster corresponding to its closest medoid, after which it searches for the medoids with progressively lower costs within the corresponding clusters. If the current medoid has already been compared with its p nearest neighbors and is still the lowest cost, the current medoid is declared to be a "local" minimum of the corresponding cluster otherwise it is replaced by the one with the lowest cost. This

procedure repeats until the medoid does not change any more and all k medoids are updated in this way.

In this operator, the idea of using size p is to get a group of objects around a medoid within the corresponding cluster. Within this group, we then identify the object with a lower cost as well as the direction where there may exist the object that can minimize the cost. An appropriate size p should be set to efficiently identify the medoids with the lowest cost for the corresponding clusters. A large p value will lead to a more time consuming operator and a too small p value may cause the operator to terminate prematurely. In this paper, the p value is determined experimentally in Section 5. To update the offspring, the subsets $C_{\text{subset}_i}(i = 1, 2, \ldots, k)$ can be identified during the non-selected objects' assignment. This requires the complexity of $O(pkn)$. The computation of new medoids requires the complexity of $O(tpn)$, where t is the average number of iteration to update the medoids. Therefore, the overall complexity of the heuristic operator is linear to the number of objects since both p and k are much less than n in the usual scenarios.

3.5 Overview of the GMCA

The overall algorithm can be stated as follows. The population is initialized randomly and is evolved over generations. The evolution is terminated when the fitness value of the best population individual has not changed for n generations. The output of the algorithm is the best individual encountered during the evolution.

Step 1. Randomly initialize a population of P individuals with the number of medoids (clusters) ranging from 2 to $k_{\max}$. Each individual uniquely encodes a candidate solution of the k-medoids clustering problem.
Step 2. Calculate MDB′ value according to Eq. (3) for each individual in the initial population and set the fitness as $f = 1/\text{MDB}'$.
Step 3. Repeat (a) to (e) until the *stopping criterion* is met.

(a) One parent is selected by 2-fold tournament selection, and its mate is selected randomly from the population. This procedure is repeated until $P/2$ parent pairs are selected.
(b) Perform crossover with the parent pairs to generate offspring with the number of clusters ranging from 2 to $k_{\max}$ and apply flip mutation to the offspring.
(c) Run the heuristic operator on the offspring and update the offspring.
(d) Calculate MDB′ value according to Eq. (3) for each offspring and set the fitness as $f = 1/\text{MDB}'$.
(e) Create a new generation of size P from the one best individual of the previous generation and the best offspring.

Step 4. Provide the individual from the terminal population with the best fitness.

4 Data sets and parameter settings

This section provides a description of the data sets and parameter settings of the proposed algorithm. Both simulated data sets and real gene expression data sets were used in our experiments. In order to allow graphically display the clustering results, the simulated data sets, *Simulate15* and *Simulate30*, are generated according to a bivariate normal distribution. Figure 1 shows these two data sets. The *Simulate15* and *Simulate30* comprise 3100 data

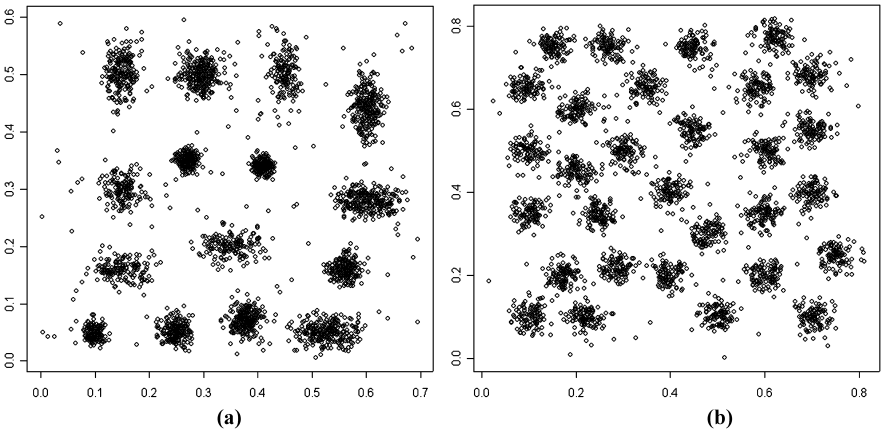


Fig. 1 (a) *Simulate15* data set and (b) *Simulate 30* data set

objects with fifteen and thirty clusters, respectively. In the *Simulate15*, the clusters can be seen to be with different sizes and different volumes. In the *Simulate30*, there are a large number of clusters. We randomly generate 100 noise data objects from a uniform distribution for both the *Simulate15* and *Simulate30*. The dissimilarity or distance measure between two objects for the experiments is based on the Euclidean metric.

The real data sets used in this paper are gene expression produced by a DNA microarray hybridization experiment representing the ratio of expression levels of n genes under two different experimental conditions. The result, on a single chip, is series of n expression-level ratios. Typically, the numerator of each ratio is the expression level of the gene in the varying condition of interest, whereas the denominator is the expression level of the gene in some reference condition. The data from a series of m such experiments may be represented as a gene expression matrix, in which each of the n rows consists of an m-element expression vector for a single gene. A crucial step in the analysis of such data is the detecting of gene groups that have similar expression profiles. Most current methods for gene expression data analysis rely on the use of clustering algorithms (Tavazoie et al., 1999; Dembele and Kastner, 2003; Wu et al., 2004).

In this paper, we consider two subsets of the yeast cell cycle data set (Cho et al., 1998). The yeast cell cycle data set contains time-course expression profiles for more than 6220 genes, with 17 time series for each gene taken at 10 min intervals covering nearly two yeast cell cycles (160 min). We used two different subsets of this data set. The first subset *Yeast384* consists of 384 genes whose expression levels peak at different time series corresponding to the five phases (early G1, late G1, S, G2 and M) of cell cycle. We expect clustering results to approximate this five-cluster partitioning. Another subset *Yeast2945* is from (Tavazoie et al., 1999). We used the same selection of 2945 genes, which are most variable genes out of 6220. In that selection, the data for the time points 90 min and 100 min are excluded because less efficient labelling of their mRNA during the original chip hybridizations. There is no *a priori* knowledge about this subset. Both subsets are normalized into z-scores by subtracting the row-mean and dividing the row-wise standard deviation so that every gene has an average expression value of zero and a standard deviation equal to one. Normally, to measure the dissimilarity or distance between two genes one tends to choose correlation coefficients which capture the similarity of the "shapes" of two expression profiles, and ignore differences between their magnitudes. However, Euclidean distance metric is used here since

it has been shown that the correlation coefficients and Euclidean distance are equivalent on a standardized gene expression data set (Yeung, 2001).

All parameter values of the GMCA for experiments on the above data sets were determined experimentally. The mutation rate was set at 0.01 for *Simulate15* and *Yeast384*, and 0.015 for *Simulate30* and *Yeast2945*. To establish these values, all variables were held constant with only the mutation rate changing. Ten runs were completed for a wide range of values of the mutation rate. The best solutions from each of the ten runs were averaged and the best average was selected. The outcomes of the ten runs were averaged to ensure that the GMCA performs well consistently over a series of runs. The crossover rate and the nearest neighbor size were determined in a similar way. First, crossover rate was varied as all other parameters were held constant, then the nearest neighbor size was determined using the established parameter values. Both the best fitness values and consistency of results were used in determining the crossover rate and the nearest neighbor size. The crossover rate of 0.85 for *Yeast384*, 0.9 for *Simulate15*, and 0.95 for *Simulate30* and *Yeast2945* with the nearest neighbor size of 3 for *Simulate15* and *Yeast384*, and 4 for *Simulate30* and *Yeast2945* were established. For the population size, we found $2 \times k_{\max}$ for each data set to be acceptable for the GMCA approach. A population size too small would retard the search ability of the GMCA. The stopping criterion was that the fitness value of the best population individual has not changed for 20 generations. It is possible that there are nonlinear relationships between the parameters so there may be a more effective set of parameter values that would result in even better performance.

5 Experiments

In this section, we examined the properties of the algorithm to evolve the appropriate partitioning of the data. After that, we did some experiments in which we compared the GMCA with other related clustering methods.

5.1 GMCA experiments

(*a) Performance of the GMCA.* We applied the GMCA to the analysis of all the above data sets. For the two simulated data sets, the solutions are demonstrated in Figures 3(a) and 4(a), which indicate appropriate partitionings with correct number of clusters were recovered on both data sets. For *Yeast384*, a partitioning with five clusters corresponding to the five phases of cell cycle was also provided by the GMCA. It is evident from the results that, the GMCA is able to evolve appropriate partitionings with correct number of clusters. For *Yeast2945*, our algorithm provides a partitioning with 23 clusters, which is rather different from the 30 clusters practically set in (Tavazoie et al., 1999). Our algorithm indicates 30 clusters in this data set may be overestimated. It was found that the MDB index for the 23 clusters (5.19) was indeed smaller than that of 30 clusters (5.45) and hence the former should be preferred over the latter.

(*b) Significance of the Heuristic Operator.* We examined the significance of the heuristic operator in the proposed algorithm. For this purpose, GMCA with and without the heuristic operator were applied on the *Simulate30*. Figure 2 shows the MDB measure over time corresponding to this experiment. It can be observed from the Figure 2 that GMCA without the heuristic operator becomes very slow after an initial period of the evolution and the heuristic operator significantly increases the speed of convergence of the algorithm. Similar

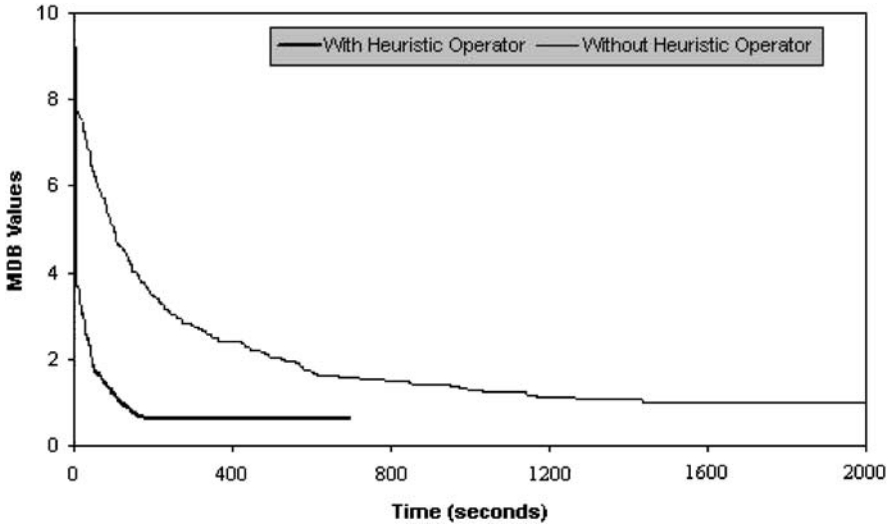


Fig. 2 GMCA with and without the heuristic operator

results can be also observed in the experiments with other data sets employed in this work. In fact, this is the main reason for using the heuristic operator in the GMCA.

5.2 Comparison with other methods

In the next experiments, we compared the proposed algorithm with the well-known CLARA (Kaufman and Rousseeuw, 1990) and two other GA *k*-medoids clustering methods, namely, the GCA (Lucasius et al., 1993) and RAR_w-GA (Estivill-Castro and Murray, 1997). To examine the robustness of our algorithm, we also compared the proposed algorithm with the genetic guided clustering algorithm (GGCA) (Hall et al., 1999), which was proposed to optimize the *k*-means function: mean squared error (MacQueen, 1967). In the GGCA, tournament selection, two-point crossover, and flip bit mutation were applied with a binary gray code representation, which was used to represent the cluster centers, for clustering. To represent clustering solution with the binary gray code, the real numbers of cluster centers were first converted into integers, which capture three decimal places of precision, and then to binary strings. Since these four methods require an *a priori* specification of the number of clusters, we run these methods on each of the data sets based on the *a priori* known or estimated number of clusters and their results are then compared with the results found by our algorithm. In order to alleviate the implementation bias, we encode all the four GA based clustering methods using the same programming language. We compare the performance with respect to the criteria of (1) *solution quality* and (2) *computational efficiency*. All results reported in this section are obtained on a PC with AMD Athlon™ 1800 running Windows™ 2000 operation system.

To make the comparisons among the four GA based clustering methods more meaningful, we used the same stopping criterion (i.e., the fitness value of the best population individual has not changed for 20 generations) and the same population size on each data set for all experiments. The rest parameters of GCA, RAR_w-GA and GGCA were specified according to the original papers for the best performance. The configuration of GCA was set according

to the Plackett-Burman design (Plackett and Burman, 1946), which leads to 0.5, 0.125 and 0.2 for the crossover, built-in mutation and flip mutation rate, respectively. In (Estivill-Castro and Murray, 1997), the RAR_w-GA was found working well with the parameter $w = 3$ and mutation rate of 0.01. The parameter values of the GGCA were determined experimentally. Hall et al. (1999) found the crossover rate of 0.9, tournament order of 2 with mutation rate of $1.75/(P \times \sqrt{(bits)})$ (P is the population size and *bits* is the number of bits in an individual) offered the best results. The configuration of CLARA has been optimized in (Kaufman and Rousseeuw, 1990), which indicates that setting the sample size as $40 + 2k$ (k is the number of clusters) and the number of samples as 5 would give satisfactory performance. Here, we adopt this standard configuration for our experiments.

To compare the performance with respect to the solution quality, we quantified the quality of the solutions in terms of both the MDB and SICV values and reported their averages and corresponding standard deviations found from the experiments. We also reported the classification error on the simulated data since we know the "true" cluster. To compute the classification error, each simulated data set was first randomly divided into two halves: one for training, another for testing. The algorithm was run on the training set. We assign each object to its most likely cluster, and assign each cluster to a class based on examining the class labels of the training data assigned to each cluster and choosing the majority class. We can then compute the error rates on the test data (excluding the noise data objects). We stress that the class labels are not used during training (i.e., in discovering clustering). They are only intended to provide independent verification of the performance of clustering. By using more than one quality measures, we are trying to better quantify the quality of the solutions. Note, for the GGCA, we only reported the classification error on the simulated data since the MDB and SICV are not applicable to this method. To compare the computational efficiency, we reported the average run time of each method on each data set. The results are shown in Tables 1 and 2 for simulated data and real data, respectively. The entries in the tables give the means (and standard deviations) in the form *mean*(±*standard deviation*). They are averaged over ten trials of each method.

Tables 1 and 2 indicate, in comparison with the GCA and RAR_w-GA, the GMCA is able to efficiently identify solutions of comparable quality on both simulated data sets and real gene expression data sets. For example, on the *Simulate30* data set, the GCA needs 851.1 seconds to identify the solution with MDB $= 0.51(\pm 0.02)$, SICV $= 87.2(\pm 0.95)$ and *percent classification* error $= 4.1(\pm 0.4)$, and the RAR_w-GA needs 877.4 seconds to identify the solution with MDB $= 0.52(\pm 0.03)$, SICV $= 89.1(\pm 1.05)$ and *percent classification* error $= 4.6(\pm 0.7)$ while the GMCA only takes 196.4 seconds to identify the solution with MDB $= 0.49(\pm 0.01)$, SICV $= 88.7(\pm 0.83)$ and *percent classification* error $= 4.9(\pm 0.4)$. The improvement in computational efficiency of the proposed algorithm is mainly coming from the use of heuristic operator which significantly increases the speed of convergence. In both cases of the simulated data, the GMCA was found outperforming the GGCA. This can be the evidence about the robustness of GMCA to the cluster number, k, and existing outliers. In comparison with the CLARA, the results uncover an interesting trade-off between the solution quality and computational efficiency. The GMCA produces solutions of better quality as compared with the CLARA. However, the GMCA takes considerably more time to find its better answer.

It is of interest to extend the above comparison with a visual inspection of the partitionings found by different methods. For this purpose, we show a typical result for each method on the two simulated data sets. The results are shown in Figures 3 and 4 (different clusters represented by different symbols) for *Simulate15* and *Simulate30* data sets, respectively.

Table 1 Comparing the performance of the five algorithms (GCA, RAR_w-GA, CLARA, GGCA and GMCA) on the simulated data sets. The results are averaged over the ten trials

	Simulate15				Simulate30			
Algorithm	MDB	SICV	% Classification Error	Run time (seconds)	MDB	SICV	% Classification Error	Run time (seconds)
GCA	0.38(±0.02)	76.7(±1.10)	3.6(±0.3)	212.4	0.51(±0.02)	87.2(±0.95)	4.1(±0.4)	851.1
RAR_w-GA	0.39(±0.03)	78.2(±1.23)	4.1(±0.6)	219.2	0.52(±0.03)	89.1(±1.05)	4.6(±0.7)	877.4
CLARA	0.50(±0.06)	102.7(±8.65)	8.7(±1.9)	0.48	0.69(±0.07)	108.9(±7.32)	11.4(±3.0)	2.52
GGCA	N/A	N/A	9.9(±2.1)	459.8	N/A	N/A	12.4(±3.3)	1553.9
GMCA	0.36(±0.02)	77.6(±1.04)	3.4(±0.2)	76.5	0.49(±0.01)	88.7(±0.83)	4.9(±0.4)	196.4

Table 2 Comparing the performance of the four algorithms (GCA, RAR_w-GA, CLARA, and GMCA) on the real data sets. The results are averaged over the ten trials

	Yeast384			Yeast2945		
Algorithm	MDB	SICV	Run time (seconds)	MDB	SICV	Run time (seconds)
GCA	2.42(±0.02)	910.4(±1.48)	62.3	5.23(±0.04)	7563.3(±15.1)	2551.6
RAR_w-GA	2.45(±0.03)	917.5(±4.95)	64.7	5.31(±0.06)	7572.2(±21.2)	2643.0
CLARA	2.56(±0.05)	1007.9(±49.6)	0.16	7.05(±0.78)	8396.1(±296.5)	2.07
GMCA	2.40(±0.01)	912.0(±1.66)	27.9	5.19(±0.02)	7572.5(±15.1)	582.7

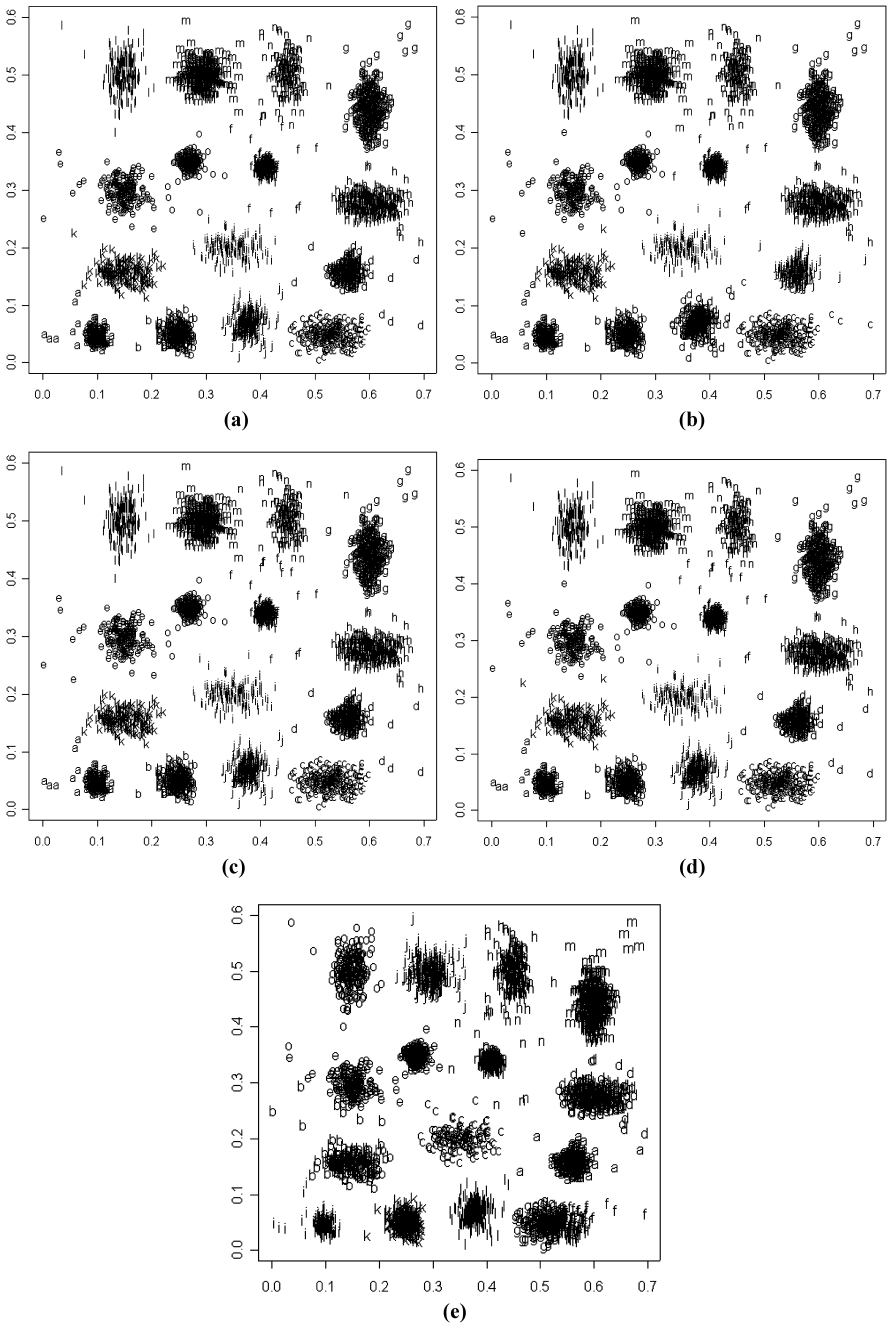


Fig. 3 Partitioning with fifteen clusters found on *Simulate15* data by the (a) GMCA, (b) CLARA, (c) GCA, (d) RAR_w-GA and (e) GGCA

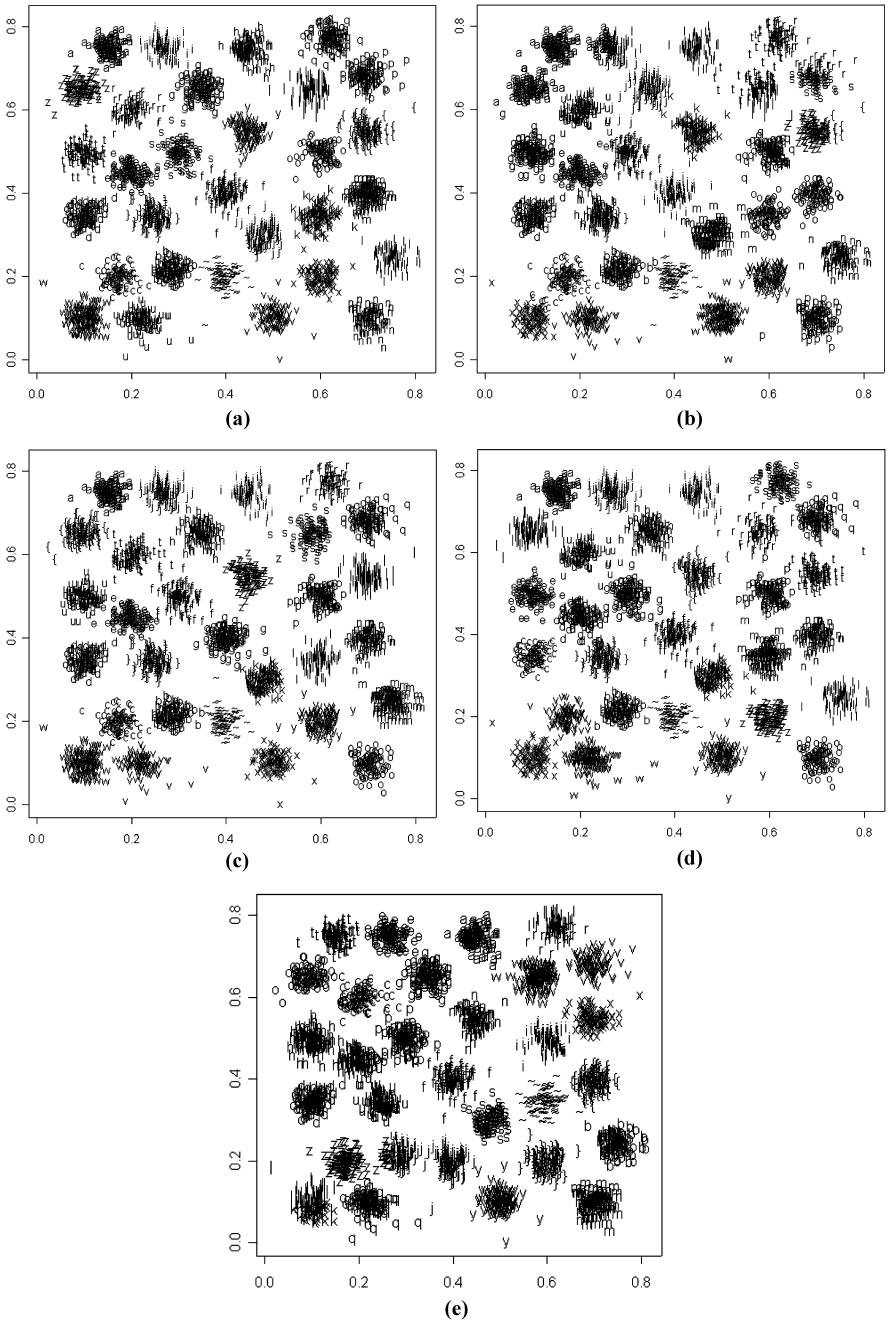


Fig. 4 Partitioning with thirty clusters found on *Simulate30* data by the (a) GMCA, (b) CLARA, (c) GCA, (d) RAR_w-GA and (e) GGCA

Figure 3 shows, to cluster the *Simulate15* data set (with a comparatively small search space), all the *k*-medoids based methods generally manage to find all the clusters correctly. The CLARA seems to have somewhat more difficulty with objects that lie between clusters. The GGCA tends to join some clusters together and split others. To cluster the *Simulate30* data set (with a comparatively large search space), the distinction among partitionings found by different methods are becoming clearer. Figure 4 shows that both the CLARA and GGCA cannot succeed in finding all clusters in *Simulate30*, as some clusters are joined together and some clusters are split. In contrast, all the three GA based *k*-medoids methods are successfully in finding all the underlying clusters.

5.3 Further comparisons

From our experiments it follows that the GMCA requires considerably more running time to produce solutions of better quality as compared with the CLARA. In order to make a more fair comparison between the GMCA and CLARA, we further examine whether the CLARA would perform better when it is configured to run roughly the same time as the GMCA. Recall that in the CLARA, PAM is performed on S number of samples, each having $n' = Ak + B < n$ objects with 5, 2 and 40 as the standard values for S, A and B, respectively. Therefore, increasing the run time of CLARA can be done by increasing S or n'. We summarize our findings as below.

Large S values, which lead to a linear increase in the computation time, generally do not improve the results significantly. Indeed, for S values that correspond to computation time approximately equal to the run time of GMCA, the latter still performs better on all data sets. On the other hand, large n' values, which lead to a quadratic increase in the computation time, generally improve the results, which agrees with findings of (Hopke and Kaufman, 1990). For n' values that correspond to computation time of the GMCA, the CLARA can perform as good as the GMCA on the *Simulate15* and *Yeast384* data sets, however, it still performs worse than the GMCA for the data sets of *Simulate30* and *Yeast2945*, which have comparatively large search space.

It is also of interest to compare the GMCA with multiple runs of the CLARA with standard configuration but different number of clusters. To make the comparison more meaningful, we process CLARA with $k = 2$ to $k = k_{\max}$, where $k_{\max}$ is taken to be $\sqrt{n}$, and evaluate the result in each case. This procedure is repeated until the same amount of time as the GMCA is spent and the best result is selected. In this case, our results indicate that the GMCA outperformed CLARA in every case of the data sets.

6 Summary and discussion

In this paper, we considered the *k*-medoids clustering problem. Since it is a NP-hard problem, approaches for *k*-medoids clustering based on alternative optimization schemes may result in local optimum solutions. Genetic algorithms, which are stochastic optimization schemes and aim at global optimum, have been reported for this problem in the literature. However, earlier GA *k*-medoids clustering approaches may take a large amount of time to converge and/or assume the number of clusters to be known *a priori*. To address these drawbacks, we suggest a hybrid genetic algorithm for *k*-medoids clustering that can efficiently evolve appropriate partitionings while making no *a priori* assumption about the number of clusters present in the datasets.

We performed experiments comparing our algorithm with the well-known CLARA and three other GA based clustering methods, namely, the GCA, RAR_w-GA and GGCA. In addition to the advantage of making no *a priori* assumption about the number of clusters present in the data sets, the GMCA has the following properties. In comparison with the GCA, RAR_w-GA and GGCA, the GMCA is able to efficiently identify appropriate partitionings. The GMCA was also found more robustness than the GGCA, which was proposed to optimize the *k*-centroids model. In comparison with the CLARA, the GMCA produces solutions of better quality.

The use of medoids for clustering is more general than centroids in that it is also applicable to situations where the objects to be clustered cannot be represented by numerical values or the mean of the objects is not defined. For example, we can use the proposed algorithm for Web mining, which can be viewed as a special case of the more general problem of knowledge discovery in databases (Agrawal and Srikant, 1994). The process of identifying a structure in an unlabeled data set in terms of categories or components plays a central role in Web mining. Consider, for example, the problem of determining categories of users with "similar" interests, or the problem of grouping together a set of pages with similar content, and so on. In Web mining, *a prior* knowledge about the number of clusters is rarely available. In addition, outliers and incomplete data can occur in the data set, due to a wide variety of reasons inherent to Web browsing and logging. Furthermore, Web objects (pages, URLs, etc.) are non-numeric in nature. Thus, Web mining techniques should be able to determine the appropriate number of clusters automatically and handle non-numeric data in the presence of noise and outliers. The algorithm proposed in this work can satisfy all these requirements. Therefore it is suitable to be applied for Web miming.

The proposed algorithm can also be used for facility location problems. The facility location problems (Agarwal et al., 1997) have many applications in telecommunications, industrial transportation and distributions, etc. One of the most well-known facility location problems is the *p*-median problem. The problem consists of locating *p* facilities in a given space which satisfy *n* demand points in such a way that the total sum of distances between each demand point and its nearest facility is minimized. The proposed GMCA can be suitably applied to the facility location problems, which may require to determine the appropriate number of facilities automatically and to be robust with outlier facilities.

There are several other possible extensions to the proposed algorithm. First, a detailed time and sensitivity analysis of the algorithm can be performed. Next, several issues like the use of other validity indexes, use of other dissimilarity metrics and comparisons with other *k*-medoids clustering algorithms can be carried out in the future. Additionally, simultaneous search for the effective set of parameter values of the developed algorithm can be attempted. Finally, parallel computer architecture of the GMCA can be exploited to further improve the computational efficiency.

References

Agarwal, P., M. Sharir, and E. Welzl. (1997). "The Discrete 2-Center Problem." In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pp. 147–155.

Agrawal, R. and R. Srikant. (1994). "Fast Algorithms for Mining Association Rules." In *Proceedings of the 20th VLDB Conference*, pp. 487–499.

Areibi, S. and Z. Yang. (2004). "Effective Memetic Algorithms for VLSI Design Automation = Genetic Algorithms + Local Search + Multi-Level Clustering." *Evolutionary Computation* 12(3), 327–353.

Bandyopadhyay, S. and U. Maulik. (2002). "An Evolutionary Technique Based on *k*-Means Algorithm for Optimal Clustering in R^N." *Information Science* 146(1–4), 221–237.

Cho, R.J., M. Campbell, E. Winzeler, L. Steinmetz, A. Conway, L. Wodicka, T. Wolfsberg, A. Gabrielian, D. Landsman, D. Lockhart, and R. Davis. (1998). "A Genome-Wide Transcriptional Analysis of the Mitotic Cell Cycle." *Molecular Cell* 2(1), 65–73.

Cucchiara, R. (1998). "Genetic Algorithms for Clustering in Machine Vision." *Machine Vision and Applications* 11(1), 1–6.

Davies, D.L. and D. W. Bouldin. (1979). "A Cluster Separation Measure." *IEEE Trans. Pattern Analysis and Machine Intelligence* 1, 224–227.

Dembele, D. and P. Kastner. (2003). "Fuzzy c-Means Method for Clustering Microarray Data." *Bioinformatics* 19(8), 973–980.

Duda, R.O., P.E. Hart, and D.G. Stork. (2001). *Pattern Classification* New York, Wiley.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass, Addison-Wesley.

Estivill-Castro, V. and A.T. Murray. (1997). "Spatial Clustering for Data Mining with Generic Algorithms." Technical Report FIT-TR-97-10, Queensland University of Technology, Australia.

Falkenauer, E. (1998). *Genetic Algorithms and Grouping Problems*. Boston: John Wiley & Sons.

Garey, M. and D. Johnson. (1979). *Computers and Intractability—A Guide to the Theory of NP-Completeness*. San Francisco, W.H. Freeman.

Goldberg, D.E. and J. Richardson. (1987). "Genetic Algorithms with Sharing for Multimodal Function Optimization." In *Proceedings of the 2nd International Conference Genetic Algorithms*, pp. 41–49.

Hall, L.O., I.B. Ozyurt, and J. C. Bezdek. (1999). "Clustering with a Genetically Optimized Approach." *IEEE Transactions Evolutionary Computation* 3(2), 103–112.

Hartigan, J.A. (1975). *Clustering Algorithms*, Wiley.

Hartigan, J.A., and M.A. Wong. (1979). "A *k*-Means Clustering Algorithm." *Applied Statistics* 28, 100–110.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, University of Michigan Press.

Hopke, P.K. and L. Kaufman. (1990). "The Use of Sampling to Cluster Large Data Sets." *Chemom. Intelligence Laboratoire System* 8, 195–204.

Hruschka, E.R., L.N. de Castro, and R.J. G.B. Campello. (2004). "Evolutionary Algorithms for Clustering Gene-Expression Data." In *Proceedings of the IEEE International Conference on Data Mining*, pp. 403–406.

Hruschka, E.R. and F.F.E. Nelson. (2003). "A Genetic Algorithm for Cluster Analysis." *Intelligent Data Analysis* 7, 15–25.

Jain, A.K. and R.C. Dubes. (1988). *Algorithms for Clustering Data*. Englewood Cliffs, N.J., Prentice Hall.

Kaufman, L. and P.J. Rousseeuw. (1990). *Finding Groups in Data: an Introduction to Cluster Analysis*. N.Y., John Wiley & Sons.

Krishna, K. and M.N. Murty. (1999). "Genetic *k*-Means Algorithm," *IEEE Transactions on Systems, Man and Cybernetics*, Part B: Cybernetics, Vol. 29, No. 3.

Lance, G.N. and W.T. Williams. (1967). "A General Theory of Classificatory Sorting Strategies: II Clustering Systems." *Computers Journal* 10, 271–277.

Lucasius, CB., A.D. Dane, and G. Kateman. (1993). "On *k*-Medoid Clustering of Large Data Sets with the Aid of a Genetic Algorithm: Background, Feasibility and Comparison." *Analytical Chimica Acta* 282, 647–669.

MacQueen, J. (1967). "Some Methods for Classification and Analysis of Multivariate Observations." In *Proceedings of the 5th Berkeley Symp. Mathematical Statistics and Probability*, pp. 281–297.

Maulik, U. and S. Bandyopadhyay. (2000). "Genetic Algorithm-based Clustering Technique." *Pattern Recognition* 33(9), 1455–1465.

Murthy, C.A. and N. Chowdhury. (1996). "In search of Optimal Clusters using Genetic Algorithms." *Pattern Recognition Letters* 17, 825–832.

Ng, R. and J. Han. (2002). "CLARANS: A Method for Clustering Objects for Spatial Data Mining." *IEEE Transactions Knowldge of Data Engineering* 14(5), 1003–1016.

Pal, N.R. and J.C. Bezdek. (1995). "On Cluster Validity for the Fuzzy c-Means Model." *IEEE Transactions on Fuzzy Systems* 3(3), 370–379.

Plackett, R.L. and J.P. Burman. (1946). "The Design of Optimum Multifactorial Experiments." *Biometrika* 33, 305–325.

Scheunders, P. (1997). "A Genetic c-Means Clustering Algorithm Applied to Color Image Quantization." *Pattern Recognition* 30(6), 859–866.

Sheng W. and X. Liu. (2004). "A Hybrid Algorithm for *k*-Medoids Clustering of Large Data Sets." In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 77–82.

Smith, G.D., J.C.W. Debuse, M.D. Ryan, and L.M. Whittley. (2000). "An Effective Genetic Algorithm for the Fixed Channel Assignment Problem." *Telecommunications Optimisation: Heuristic and Adaptive Techniques*, John Wiley and Sons, pp. 357–371.

Tavazoie, S., D. Hughes, J.M.J. Campbell, R.J. Cho, and G.M. Church. (1999). "Systematic Determination of Genetic Metwork Architecture." *Nature Genetics* 22, 281–285.

Wu, S., A.W.C Liew, H. Yan, and M. Yang. (2004). "Cluster Analysis of Gene Expression Database on Self-Splitting and Merging Competitive Learning." *IEEE Transactions on Information Technology in Biomedicine* 8(1).

Yeung, K.Y. (2001). "Clustering Analysis of Gene Expression data." PhD Thesis, University of Washington.

Yi, L., S. Lu, F. Fotouhi, Y. Deng, and S. Brown. (2004). "Incremental Genetic *k*-Means Algorithm and Its Application in Gene Expression Data Analysis." *BMC Bioinformatics* 5, 172.