Expert Systems with Applications 37 (2010) 4966–4973

Contents lists available at ScienceDirect

Expert Systems with Applications

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

# A quantum-inspired genetic algorithm for *k*-means clustering

Jing Xiao [a,b], YuPing Yan [b], Jun Zhang [b], Yong Tang [a,*]

[a] School of Computer Science, South China Normal University, Guangzhou 510631, PR China
[b] Key Lab of Machine Intelligence and Sensor Network (Sun Yat-sen University), Ministry of Education, Guangzhou 510006, PR China

ARTICLE INFO

*Keywords:*
*k*-means clustering
Genetic algorithms
Quantum-inspired genetic algorithms

ABSTRACT

The number of clusters has to be known in advance for the conventional *k*-means clustering algorithm and moreover the clustering result is sensitive to the selection of the initial cluster centroids. This sensitivity may make the algorithm converge to the local optima. This paper proposes a quantum-inspired genetic algorithm for *k*-means clustering (KMQGA). In KMQGA, a Q-bit based representation is employed for exploration and exploitation in discrete 0–1 hyperspace using rotation operation of quantum gate as well as the typical genetic algorithm operations (selection, crossover and mutation) of Q-bits. Different from the typical quantum-inspired genetic algorithms (QGA), the length of a Q-bit in KMQGA is variable during evolution. Without knowing the exact number of clusters beforehand, KMQGA can obtain the optimal number of clusters as well as providing the optimal cluster centroids. Both the simulated datasets and the real datasets are used to validate KMQGA, respectively. The experimental results show that KMQGA is promising and effective.



## 1. Introduction

Clustering plays an important role in many unsupervised learning areas, such as pattern recognition, data mining and knowledge discovery. Clustering problem can be summarized as: Given $n$ points in $R^d$ space and an integer $k$, find a set of $k$ points, called centroids, such that the sum of the distances of each of the $n$ points to its nearest centroid is minimized. Generally speaking, conventional clustering algorithms can be grouped into two main categories, namely hierarchical clustering algorithms and partitional clustering algorithms. A hierarchical clustering algorithm outputs a dendrogram, which is a tree structure showing a sequence of clusterings with each clustering being a partition of the dataset (Leung, Zhang, & Xu, 2000). Unlike the hierarchical clustering algorithm, the partitional clustering algorithms partition the data set into a number of clusters, and the output is only a single partition of the data set. The majority of partitional clustering algorithms obtain the partition through the maximization or minimization of some criterion functions. Recent researches show that the partitional clustering algorithms are well suited for clustering a large dataset due to their relatively low computational requirements (Steinbach, Karypis, & Kumar, 2000). And the time complexity of the partitional algorithms is almost linear, which makes them widely used (Abraham, Das, & Konar, 2006).

Among the partitional clustering algorithms, the most famous one is *k*-means clustering (Hartigan, 1975). *K*-means clustering algorithm first randomly generates $k$ initial cluster centroids. After several iterations of the algorithm, data can be classified into certain clusters by the criterion function, which makes the data close to each other in the same cluster and widely separated among clusters. However, the traditional *k*-means clustering algorithm has two drawbacks. The one is that the number of clusters has to be known in advance, and the other is that the clustering result is sensitive to the selection of initial cluster centroids and this may lead the algorithm converge to the local optima. Different datasets have different number of clusters, which is difficult to known beforehand, and the initial cluster centroids are selected randomly, which will make the algorithm converge to the different local optima. Therefore, a lot of research efforts have been conducted on mitigating the two drawbacks of the conventional *k*-means clustering algorithm. The genetic algorithm (GA) is one of the methods to avoid local optima and discover good initial centroids that lead to superior partitions under *k*-means. In Krishna and Murty (1999), the authors proposed a novel hybrid genetic algorithm that finds a globally optimal partition of a given dataset into a specified number of clusters for *k*-means algorithm. Lu, Lu, Fotouhi, Deng, and Brown (2004) proposed a fast genetic *k*-means algorithm which is an improved version of the work in Krishna and Murty (1999) with faster convergence to the global optima. Laszlo and Mukherjee (2007) presented a genetic algorithm using a novel crossover operator that exchanges neighboring centers for selecting centers to seed the *k*-means method for clustering.

* Corresponding author. Tel.: +8620 85211532 201.
*E-mail addresses:* jingsxiao@gmail.com (J. Xiao), yuson_yan@163.com (Y. Yan), junzhang@ieee.org (J. Zhang), ytang@scnu.edu.cn (Y. Tang).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.12.017

Recently, the work of combining quantum computing and evolutionary computing has stimulated the studies of quantum-inspired evolutionary algorithms and their applications. The most important work to classical computer was done by Han and Kim (2002). They proposed a quantum-inspired evolutionary algorithm (QEA) which was used to solve combinational problem on classical electronic computer. QEA used a Q-bit as a probabilistic representation and a Q-bit individual is defined by a string of Q-bits. Since the Q-bit representation in quantum computing can represent linear superposition of states probabilistically, it has a better characteristic of population diversity than binary, numeric, or symbolic representations. The ensemble of quantum-inspired algorithms has been successfully employed in a series of optimization problems due to its effectiveness. In Wang, Wu, Tang, and Zheng (2005), a quantum-inspired genetic algorithm was proposed for flow shop scheduling problem with a single objective. And in Li and Wang (2007), the authors proposed a hybrid quantum-inspired genetic algorithm for multi-objective flow shop scheduling problem. The concept of quantum computing was also applied to some other applications, such as solving the travelling salesman problem (Talbi, Draa, & Batouche, 2004), image segmentation (Karima, Mouloud, Yacine, & Nadjib, 2006), and face detection (Jang, Han, & Kim, 2004).

In this paper, we attempt to utilize the quantum-inspired genetic algorithm to alleviate the drawbacks in the *k*-means clustering method. An improved *K*-means clustering algorithm based on quantum-inspired genetic algorithm (KMQGA) is proposed. KMQGA uses Q-bits as its chromosome's representation and Davies–Bouldin rule index (Bouldin, 1979) as criterion function. Every chromosome in KMQGA denotes cluster centroids of a partition. Different chromosomes can have different string lengths of Q-bits, that is, different chromosomes denote different partitions. Before performing the GA operations (selection, crossover, and mutation) and the quantum operation (rotation), the Q-bit representation firstly has to be encoded into the binary representation, and then the real-coded representation. When the GA operations and the quantum operation are conducted during the iterations of KMQGA, the chromosome may change its string length, that is, the cluster centroids of a partition are changed. After several iterations of KMQGA, the improved algorithm can find the optimal number of clusters as well as the initial cluster centroids, due to the string length variation of chromosomes. The simulated datasets and the real datasets are used to verify the effectiveness of KMQGA and to compare KMQGA with a *K*-means clustering algorithm based on the famous variable string length genetic algorithm (KMVGA) (Bandyopadhyay & Mauilk, 2001; Mauilk & Bandyopadhyay, 2002) respectively.

The rest part of this paper is organized as follows: Section 2 briefly introduces the background of quantum computing. The implementation details of KMQGA are described in Section 3. Experimental results and analysis are presented in Section 4. We conclude this paper in Section 5.

## 2. Quantum computing

Before describing KMQGA, we introduce the quantum computing briefly. Unlike the two-state (0/1) representation in conventional computing, the smallest information representation in quantum computing is called quantum bit (Q-bit) (Hey, 1999). The state of Q-bit may be "0" state, "1" state or any superposition of the two. So the state of Q-bit can be represented in formula (1):

$$|\psi>=\alpha|0>\beta|1> \tag{1}$$

where $\alpha$ and $\beta$ are complex numbers that specify the probability amplitudes of the corresponding states. Thus, $|\alpha|^2$ and $|\beta|^2$ denote probabilities that the Q-bit will be found in the "0" state and the "1" state, respectively. Normalization of the state to the unity guarantees in formula (2):

$$|\alpha|^2+|\beta|^2=1 \tag{2}$$

Thus, a Q-bit can be represented as the linear superposition of the two conventional binary genes (0 and 1). A Q-bit individual as a string of $m$ Q-bits is defined in formula (3):

$$\begin{bmatrix}\alpha_1 & \alpha_2 & \cdots & \alpha_m\\ \beta_1 & \beta_2 & \cdots & \beta_m\end{bmatrix} \tag{3}$$

If there is a string of $m$ Q-bits, then the string can represent $2^m$ states at the same time. The state of a Q-bit can be changed by the operation with a quantum gate, such as the NOT gate, the rotation gate, etc. However, the superposition of "0" state and the "1" state must collapse to a single state in the action of observing a quantum state, that is, a quantum state have to be the "0" state or "1" state. In the evolutionary computing, the Q-bit representation has a better characteristic of population diversity than other representations, because it can represent linear superposition of states probabilistically.

Here is an example that can explain the essence of quantum's Q-bit. If there is a Q-bit individual with three-Q-bits in formula (4)

$$\begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{2}\\ \frac{1}{\sqrt{2}} & \frac{\sqrt{2}}{\sqrt{3}} & \frac{\sqrt{3}}{2}\end{bmatrix} \tag{4}$$

Then the state of Q-bit individual can be formulated as formula (5)

$$\frac{1}{2\sqrt{6}}|000\rangle+\frac{1}{2\sqrt{2}}|001\rangle+\frac{1}{2\sqrt{3}}|010\rangle+\frac{1}{2\sqrt{6}}|100\rangle+\frac{1}{2\sqrt{2}}|101\rangle+\frac{1}{2\sqrt{3}}|110\rangle+\frac{1}{2\sqrt{2}}|111\rangle \tag{5}$$

That is to say, the probabilities of the Q-bit individual being 000, 001, 010, 011, 100, 101, 110, 111 are $\frac{1}{24},\frac{3}{24},\frac{2}{24},\frac{6}{24},\frac{1}{24},\frac{3}{24},\frac{6}{24}$, respectively. Obviously, a single Q-bit individual in the above contains eight states with a certain probability to get each state while the conventional evolutionary algorithms just contain a single state in a single individual.

More examples of Q-bit representation can be found in Han and Kim (2002). Inspired by the concept of the representation of Q-bit in quantum computing, KMQGA is designed with this novel Q-bit representation and the quantum rotation operation is one of the various operations in the algorithm.

## 3. KMQGA

In this section, we first propose the overall algorithm of KMQGA. Then the chromosome representation and the fitness function of this algorithm will be presented. At last, four main operations and an accessional operation (GA selection, GA crossover, GA mutation, quantum rotation operation and quantum catastrophe operation) will be interpreted.

### 3.1. The overall flowchart of KMQGA

KMQGA uses Davies–Bouldin (DB) rule index as its criterion function, which is also called fitness function, to evaluate whether the partition of a dataset is good or not. One of the important features of Davies–Bouldin run index is that it justifies a partition of a dataset based on both the inter-cluster and the intra-cluster. It means that a partition is good enough if all the data in the same cluster are similar, and data in different cluster are far too similar (Bouldin, 1979).

The overall procedure of this algorithm is shown in Fig. 1.

The algorithm begins with initializing the population randomly. Each chromosome (also called individual) in the population denotes a certain partition of a dataset. More specifically, each chromosome is represented by the Q-bit representation at first. Then the Q-bit string is collapsed into a certain state, which is a binary string. After the collapse operation, there is another operation to change this binary string into real-coded string. Then a Q-bit individual consists of a group of real-coded numbers, and each real number in the real-coded string denotes a pattern of the dataset. That is to say that a Q-bit individual consists of several centroids and it can be represented some kinds of partition accordingly. Then the algorithm runs conventional *K*-means clustering algorithm on each chromosome just one time and evaluates each chromosome by the DB rule index fitness function. The fitness value and the probabilities which are used in the later GA selection operation are calculated accordingly. The performance of a certain partition of a dataset is evaluated by the fitness value. Based on the probabilities, the algorithm can produce a new population through the roulette selection and the elite selection operations. The GA crossover operation affects each chromosome in terms of the crossover probability. In the process of GA crossover operation, the length of each chromosome may be changed. Due to this change, the partition denoted by the chromosome is changed accordingly, and after several iterations of this algorithm, the better chromosome may be shown up. Then the GA mutation and quantum rotation operation are performed, both of which may make the searching space of KMQGA more diversified. In order to avoid the degeneration of the chromosome and prematurity of the algorithm, KMQGA uses elite selection operation and quantum catastrophe operation. The overall procedure of KMQGA is summarized as follows:

(a) Randomly generating an initial population using Q-bit representation.

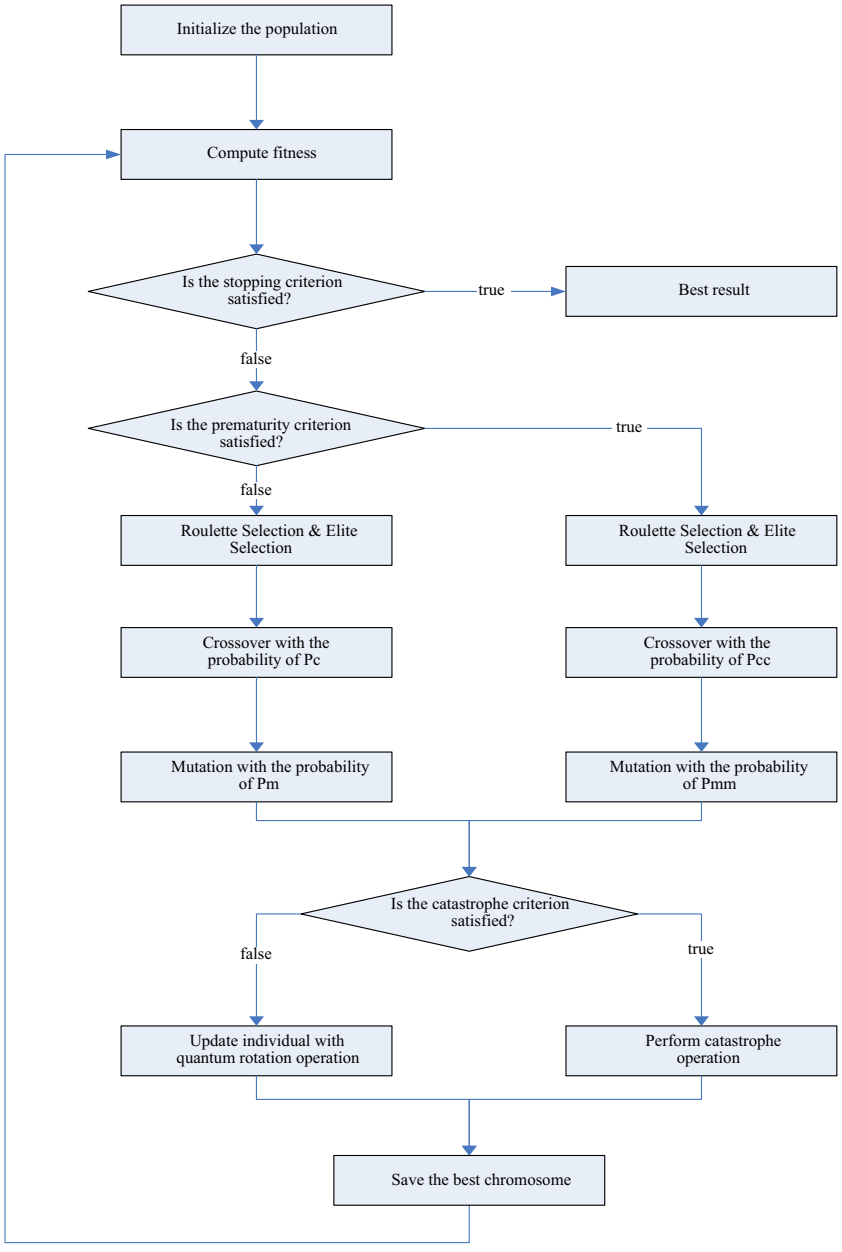


**Fig. 1.** The overall flowchart of KMQGA.

(b) Collapsing the Q-bit representation into the binary representation, and then the real-coded representation. Using the DB fitness function to evaluate each chromosome.
(c) If the stopping criterion is satisfied, then output the best result; otherwise save the best chromosome and go to the next steps.
(d) If the prematurity criterion is satisfied, then go to step (f); otherwise go to step (e).
(e) Adjusting the probabilities of crossover and mutation as $P_c$ and $P_m$, respectively. Then go to step (g).
(f) Adjusting the probabilities of crossover and mutation as $P_{cc}$ and $P_{mm}$, respectively. Then go to step (g).
(g) Performing selection operation, crossover operation and mutation operation.
(h) If the catastrophe criterion is satisfied, then go on step (j); otherwise go to step (i).
(i) Look up table of rotation angle (refer to Table 1), and then perform rotation operation. Go back to step (b).
(j) Perform catastrophe operation. Go back to step (b).

KMQGA sets a maximal iteration number $N_{max}$ as its stopping criterion, which should be as big as possible in order to find the optimal solution. The prematurity criterion is satisfied when the best chromosome saved by the algorithm does not change after $n_{max}$ iterations, where $n_{max} < N_{max}$. That is to say, if the best chromosome in current iteration is the same as the best one which is $n_{max}$ iterations ago, then the probabilities of the crossover ($P_c$) and the mutation ($P_m$) should be adjusted to $P_{cc}$ and $P_{mm}$, respectively, for the sake of jumping out of the local trap. Since the larger probability of the crossover can exchange more information among a couple of chromosomes, and the larger probability of the mutation can enlarge the diversity of the population, the algorithm adjusts $P_c$ and $P_m$ to larger probabilities ($P_{cc}$ and $P_{mm}$). The catastrophe criterion (Wang et al., 2005) is similar to the prematurity criterion. We think it is trapped in local optima if the best solution does not vary in $m_{max}$ ($m_{max} < n_{max}$) consecutive number of generations. Then the best solution is reserved and the others will be replaced by randomly generated solutions.

The four probabilities $(p_c, p_m, p_{cc}, p_{mm})$ are real numbers less than 1.0, where $p_{cc} > p_c, p_{mm} > p_m, p_{cc} = random(p_c, 1)$, and $p_{mm} = random(p_m, 2 \times p_m)$, and $p_{mm} = random(p_m 2 \times p_m)$ Here, we set $p_m < 0.5$ and $random(a, b)$ is a function that generates a random real number between $a$ and $b$.

### 3.2. Representation of chromosome in KMQGA

Inspired by the Q-bit representation and the modified variable string length genetic algorithm (Song & Park, 2006) (MVGA), KMQGA employs a modified variable Q-bit string length representation. Before coding the chromosomes, the range of $K$ (number of clusters) $[K_{min}, K_{max}]$ should be defined first, where $K_{min} \geqslant 2$ and $K_{max} \leqslant N$ ($N$ is the number of instances). As long as the $K_{max}$ is not less than the exact number of clusters, the algorithm could obtain the optimal solution. However, if $K_{max}$ is much greater than the exact number of clusters, the cost of both the computation time and the storage space will be high. Research shows that the optimal $K$ is not larger than $\sqrt{N}$ (Yang, Li, Hu, & Pan, 2006). In this paper, the range of $K$ is set to $[2, \sqrt{N}+1]$.

**Table 1**
The lookup table of rotation angle.

| $r_i$ | $b_i$ | $\mathscr{F}(r) < F(b)$ | $\Delta\theta_i$ | $s(\alpha_i, \beta_i)$ | | | |
|---|---|---|---|---|---|---|---|
| | | | | $\alpha_i\beta_i > 0$ | $\alpha_i\beta_i < 0$ | $\alpha_i = 0$ | $\beta_i = 0$ |
| 0 | 0 | False | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | True | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | False | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | True | $0.05\pi$ | −1 | +1 | ±1 | 0 |
| 1 | 0 | False | $0.01\pi$ | −1 | +1 | ±1 | 0 |
| 1 | 0 | True | $0.025\pi$ | +1 | −1 | 0 | ±1 |
| 1 | 1 | False | $0.005\pi$ | +1 | −1 | 0 | ±1 |
| 1 | 1 | True | $0.025\pi$ | +1 | −1 | 0 | ±1 |

Knowing the amount $N$, we are also aware of the number of binary string length $B$ to denote a pattern ID, that is, $2^{B-1} < N \leqslant 2^B$. For example, if the number of instances is 178, then $B = 8$. When coding the chromosome, a number $\ell$ is randomly generated first, where $\ell \in [K_{min}, K_{max}]$. Thus, the Q-bit string length of this chromosome is $\ell \times B$. In the conventional QGA or QEA, the lengths of all Q-bit individuals are the same. After several operations of selection, crossover, mutation and quantum gate, the length of every Q-bit individual do not change and still are the same. In KMQGA, we propose a quite different method from the conventional QGA and QEA. The length of a Q-bit individual depends on the number $\ell$, and the $\ell$ is a random number ranging in $[k_{min}, k_{max}]$. Therefore, every Q-bit individual may be different from each other and the crossover operation which we will mention in the following section may change the length of Q-bit individual. Because of the variable length representation, a Q-bit individual denotes different number of centroids. And after several iterations, KMQGA may find the number of centroids in the best partition. The chromosome representation is presented as in formula (6).

$$\mathscr{R} = \left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_{\ell\times B} \\ \beta_1 & \beta_2 & \cdots & \beta_{\ell\times B} \end{array}\right] \tag{6}$$

Then the Q-bit string will be collapsed into a certain state, which is a binary string. That is, randomly generating a float number $\eta$, where $\eta \in (0, 1)$. If $\eta \leqslant |\alpha_i|^2$ the corresponding binary state is 0, otherwise is 1.

After the collapse operation, there is another operation to change this binary string into real-coded string. The algorithm converts every $B$ binary string into a real number. Each real number in the real-coded string denotes a pattern of the dataset.

### 3.3. Fitness function of KMQGA

In order to evaluate whether the quality of a certain partition is good or not, we should use a criterion function (also called fitness function). A good fitness function can give a high value to a good partition solution and give a low value to a bad partition. There are many fitness functions to evaluate clustering nowadays, such as Euclidean distance, Davies–Bouldin rule index (Bouldin, 1979), Dunn index (Dunn, 1974), and Turi (Turi, 2001), S_DBW (Halkidi & Vazirgiannis, 2001), CDBW (Halkidi & Vazirgiannis, 2002).

Since the DB rule index can evaluate the distance of intra-cluster and the distance of inter-cluster, KMQGA employs DB as its fitness function. We describe DB as follows (Zhang, Liu, & Guan, 2006) in formulas (7) and (8):

$$S_i = \frac{1}{N_i}\sum_{X\in G_i} \|X - Z_i\| \tag{7}$$

$$Z_i = \frac{1}{N_i}\sum_{X\in G_i} X \tag{8}$$

where $G_i$ is the set the $i$th cluster, $z_i$ is the amount of instances in the cluster $G_i$, $N_i$ is the centroids of $G_i$, $\|X - Z_i\|$ is the Euclidean distance between pattern $X$ and $Z_i$, $S_i$ means the average cohesion of $G_i$.

$$d_{ij} = \|Z_i - Z_j\| \tag{9}$$

In formula (9), $d_{ij}$ denotes the distance between $Z_i$ and $Z_j$ and formula (11) denotes the DB rule index.

$$R_i = max_{j,j\neq i}\left\{\frac{S_i + S_j}{d_{i,j}}\right\}, i\neq j \tag{10}$$

$$DB = \frac{1}{K}\sum_{i=1}^{k} R_i \tag{11}$$

The smaller the DB value is, the better the partition is. So we let $\frac{1}{DB}$ be the KMQGA's fitness function as formula (12):

$$\mathscr{F} = \frac{1}{\mathrm{DB}} \tag{12}$$

Since the Q-bit representation cannot denote the expression of the pattern ID directly, the Q-bit representation has to be collapsed to binary string representation first. A Q-bit string $\mathscr{R}$ with length $\ell \times B$ is firstly collapsed to a certain state (0-1 state) according to the probability amplitudes of the individual. For $i = 1, 2, 3, \ldots, \ell \times B$, we generate a random number $\eta$ between $[0, 1]$. If $\alpha_i$ of chromosome $\mathscr{R}$ satisfies $|\alpha_i|^2 > \eta$, then set $\mathscr{R}_i$ to 0, otherwise set it to 1. After the binary string is constructed, we encode every $\mathscr{B}$-length binary string into a real number, which is a pattern ID. That is to say, KMQGA first changes the Q-bit string to binary string, and then converts it to real number. Thus, a Q-bit string with length $\ell \times B$ denotes a partition with $\ell$ centroids. By doing this, the DB fitness function can evaluate every partition.

### 3.4. Selection operations in KMQGA

Roulette selection and elite selection are employed in KMQGA. Roulette selection runs according to the selected probability. Usually, the chromosome with high fitness value will be chosen to the next iteration with a high probability. However, in order to avoid the case where the optimum chromosome with good fitness value is not selected occasionally, KMQGA performs the elite selection after the roulette selection operation. The elite selection guarantees that the best chromosome in a certain generation would not be lost in the evolutionary process. The basic idea of elite selection is as follows: if a certain individual in the former generation is better than the best individual in the current generation, some individuals in the current generation will be replaced by the better ones in the former generation. The pseudo code of elite selection is expressed as follows:

For the $t$th generation, the population is $\{P(t) = \{P(t,1), P(t,2), \ldots, P(t,N)\}$;
and $F_{max}(t)$ is the best individual in the $t$th generation.
$F_{max}(t) = max\{F(P(t,1)), F(P(t,2)), \ldots, F(P(t,N))\}$;
$F_{max}(t+1) = max\{F(P(t+1,1)), F(P(t+1,2)), \ldots, F(P(t+1,N))\}$;
if $F_{max}(t) > F_{max}(t+1)$ then
replicate
$\{P(t,k) | F(P(t,k)) > F_{max}(t+1), P(t,k) \in P(t)\}$;
replace randomly
$\{P(t+1,j) \in P(t+1)\}$ with $\{P(t,k) | F(P(t,k)) > F_{max}(t+1), p(t,k) \in P(t)\}$;
enf if

### 3.5. Crossover operation in KMQGA

KMQGA uses a special crossover operation which can change the lengths of parental chromosomes. For each chromosome, the crossover point is randomly chosen according to its own string length. Take for an instance, there are two chromosomes ($\mathscr{R}_1$ and $\mathscr{R}_2$) in formulas (13) and (14) with length 8 and 5, respectively:

$$\mathscr{R}_1: \left[\begin{array}{c|c|c|c|c|c|c|c} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 & \alpha_8 \\ \beta_1 & \beta_2 & \beta_3 & \beta_4 & \beta_5 & \beta_6 & \beta_7 & \beta_8 \end{array}\right] \tag{13}$$

$$\mathscr{R}_1: \left[\begin{array}{c|c|c|c|c} \alpha'_1 & \alpha'_2 & \alpha'_3 & \alpha'_4 & \alpha'_5 \\ \beta'_1 & \beta'_2 & \beta'_3 & \beta'_4 & \beta'_5 \end{array}\right] \tag{14}$$

A random integer between 1 and its length is generated as the crossover point for each chromosome. For example, the crossover point of $\mathscr{R}_1$ are 6 and 2, respectively. Then, the new chromosomes ($\mathscr{R}'_1$ and $\mathscr{R}'_2$) are shown as in formulas (15) and (16):

$$\mathscr{R}'_1: \left[\begin{array}{c|c|c|c|c|c|c|c|c} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & \alpha'_3 & \alpha'_4 & \alpha'_5 \\ \beta_1 & \beta_2 & \beta_3 & \beta_4 & \beta_5 & \beta_6 & \beta'_3 & \beta'_4 & \beta'_5 \end{array}\right] \tag{15}$$

$$\mathscr{R}'_2: \left[\begin{array}{c|c|c|c} \alpha'_1 & \alpha'_2 & \alpha_7 & \alpha_8 \\ \beta'_1 & \beta'_2 & \beta_7 & \beta_8 \end{array}\right] \tag{16}$$

Now, we can see both of the chromosomes' lengths are changed. Due to this change, the partition solution denoted by the chromosome is changed accordingly. Thus, the search space is larger and the optimal solution could be found.

### 3.6. Mutation operation in KMQGA

For the diversity, mutation is employed. Based on the mutation probability, mutation point is generated randomly according to the chromosome's length. For example, there is a chromosome ($\mathscr{R}_3$) as in formula (17) with the length of 7.

$$\mathscr{R}_3: \left[\begin{array}{c|c|c|c|c|c|c} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \\ \beta_1 & \beta_2 & \beta_3 & \beta_4 & \beta_5 & \beta_6 & \beta_7 \end{array}\right] \tag{17}$$

The mutation point is a number between 1 and its string length 7. If $\mathscr{R}_3$ mutation point is 5, then KMQGA changes the position of $\alpha_5$ and $\beta_5$. The new chromosome is $\mathscr{R}'_3$ is shown as formula (18).

$$\mathscr{R}'_3: \left[\begin{array}{c|c|c|c|c|c|c} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \\ \beta_1 & \beta_2 & \beta_3 & \beta_4 & \beta_5 & \beta_6 & \beta_7 \end{array}\right] \tag{18}$$

### 3.7. Rotation operation in KMQGA

A rotation gate $U(\theta)$ is employed to update a Q-bit individual as formula (19):

$$\begin{bmatrix} \alpha'_i \\ \beta'_i \end{bmatrix} = U(\theta_i) \times \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \times \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \tag{19}$$

where $\begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}$ is the $i$th Q-bit and $\theta_i$ is the rotation angle of each Q-bit toward either 0 or 1 state depending on its sign.

The rotation operation used in KMQGA is to adjust the probability amplitudes of each Q-bit. According to the rotation operation in formula (19), a quantum gate $U(\theta_i)$ is a function of $\theta_i = s(\alpha_i, \beta_i) \times \Delta\theta_i$, where $s(\alpha_i, \beta_i)$ is the sign of $\theta_i$ which determines the direction and $\Delta\theta_i$ is the magnitude of rotation angle (Li & Wang, 2007). The lookup table of $\Delta\theta_i$ is shown in Table 1, where $b_i$ and $r_i$ are the $i$th bit of the best solution $b$ and a binary solution of $a$, respectively. In particular, if the length of $b$ is not the same as the length of $b$, an additional correcting performance will be employed. In this paper, we increase/delete the $r$'s Q-bits randomly if the length of $r$ is less/more than the length of $b$. Because $b$ is the best solution in KMQGA, the use of quantum gate rotation is to emphasize the searching direction towards $b$.

### 3.8. Catastrophe operation in KMQGA

We consider that it is trapped in local optima if the best solution does not change in a certain number of consecutive generations. KMQGA records the fitness of best solution for each iteration and compares the best fitness in current iteration with the best fitness

**Table 2**
Details of clusters' ranges in sds1 dataset.

| Coordinate | Range of sds1 | | |
|---|---|---|---|
| | Cluster1 | Cluster2 | Cluster3 |
| *X* | [0,20] | [40,60] | [80,100] |
| *Y* | [0,20] | [40,60] | [80,100] |

**Table 3**
Details of clusters' ranges in sds2 dataset.

| Coordinate | Range of sds2 | | | |
|---|---|---|---|---|
| | Cluster1 | Cluster2 | Cluster3 | Cluster4 |
| *X* | [0,20] | [40,60] | [80,100] | [0,20] |
| *Y* | [0,20] | [40,60] | [80,100] | [80,100] |

in the last previous iteration. To avoid premature convergence, a catastrophe operation (Li & Wang, 2007) is used in KMQGA. If the algorithm obtains the same best fitness in a certain number of consecutive generations, we consider KMQGA traps in premature convergence and perform the catastrophe operation. KMQGA keeps the best individual and the other individuals are re-generated randomly.

## 4. Experimental evaluation of KMQGA

In this section, we test the performance of the proposed KMQGA. First, we test KMQGA's effectiveness using three simulated datasets. Then four datasets from famous UCI machine learning repository (Blake, Keogh, & Merz, 1998) are utilized to compare the performance and the effectiveness of KMQGA with those of KMVGA. In both the simulated data and the real data experiments, we empirically set the size of population to 100, $\mathscr{P}_c = 0.9, \mathscr{P}_m = 0.01, \mathscr{n}_{max} = 15, \mathscr{m}_{max} = 25$. We set $\mathscr{N}_{max} = 100$ and $\mathscr{N}_{max} = 300$ in simulated data and real data respectively. $\mathscr{N}_{max} = 100$ and $\mathscr{N}_{max} = 300$ are randomly generated as described in Section 3.1.

### 4.1. Simulated datasets

We randomly generate three simulated datasets (sds1, sds2, sds3). There are three, four and eight clusters in sds1, sds2, sds3, respectively. Each cluster in simulated datasets has 100 data vectors. To get the direct vision from the coordinate easily, we define each data vector as two dimensions. Details of clusters' ranges of sds1, sds2 and sds3 are given in Tables 2–4 respectively.

After a certain range is given, these data vectors are generated in uniform probability distribution within the given range. We run KMQGA with the three simulated datasets for 30 times, and each time we can obtain the number of clusters 3, 4, 8 for sds1, sds2 and sds3, respectively.

KMQGA also obtains the best initial cluster centroids of each dataset. One of the 30 runs experiments' results are shown as follows. Figs. 2–4 depict the partitions solutions of sds1, sds2, sds3, respectively.

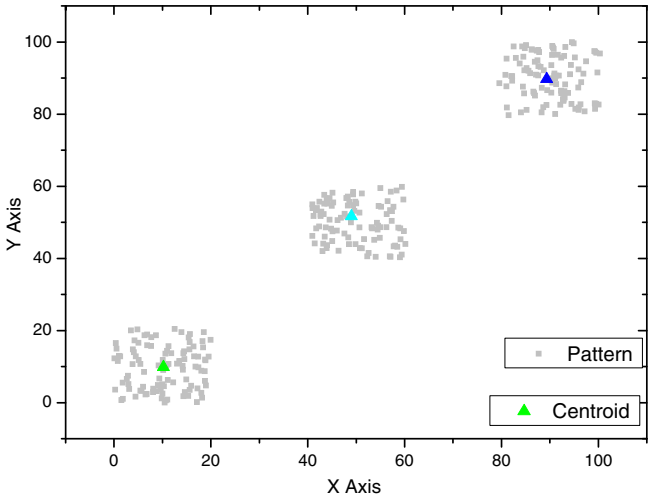


**Fig. 2.** Initial centroids obtained by KMQGA for sds1 dataset.

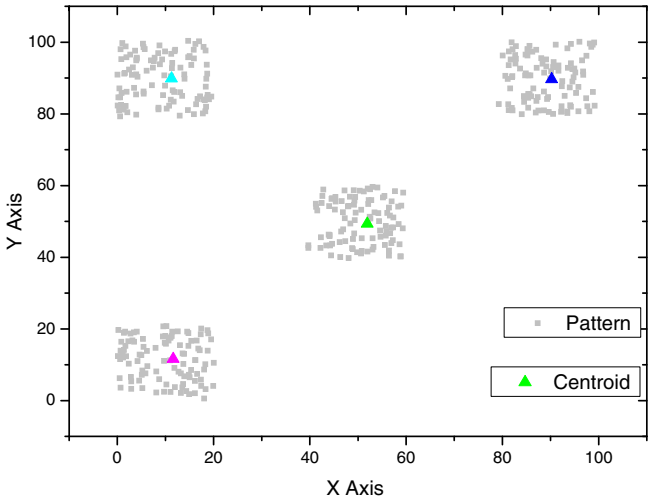


**Fig. 3.** Initial centroids obtained by KMQGA for sds2 dataset.

In Figs. 2–4, the initial cluster centroids obtained by KMQGA are reasonable since each centroid is almost in the center of corresponding data vectors from a visual point of view. Thus, the correctness of KMQGA is proved by the simulated datasets in our experiment.

### 4.2. Real datasets (UCI datasets)

The four real datasets from UCI machine learning repository are Glass, Wine, SPECTF-Heart and Iris. The details of the four datasets, which can be found in the .names file of every dataset's fold, are summarized in Table 5.

**Table 4**
Details of clusters' ranges in sds3 dataset.

| Coordinate | Range of sds3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Cluster1 | Cluster2 | Cluster3 | Cluster4 | Cluster5 | Cluster6 | Cluster7 | Cluster8 |
| *X* | [0,20] | [40,60] | [80,100] | [80,100] | [0,20] | [180,200] | [180,200] | [180,200] |
| *Y* | [0,20] | [40,60] | [80,100] | [0,20] | [180,200] | [0,20] | [80,100] | [180,200] |

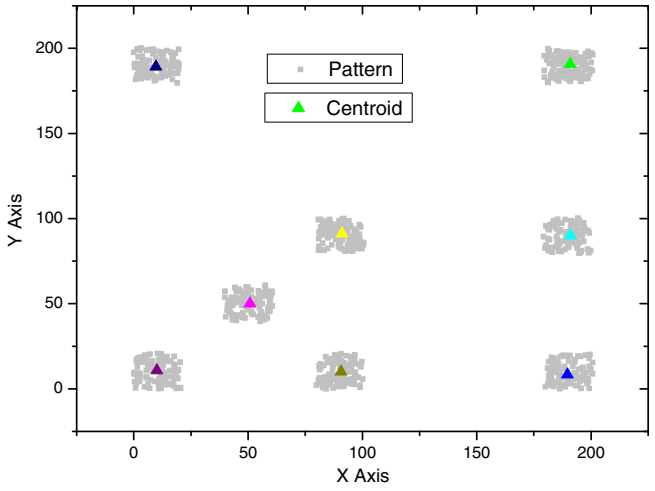


**Fig. 4.** Initial centroids obtained by KMQGA for sds3 dataset.

**Table 5**
Details of UCI real datasets.

| Dataset | Number of instances | Number of attributes | Number of clusters |
|---|---|---|---|
| Glass | 214 | 9 | 2 |
| Wine | 178 | 13 | 3 |
| SPECTF-Heart | 187 | 44 | 2 |
| Iris | 150 | 4 | 3 |

**Table 6**
Comparison between KMQGA and KMVGA.

| Dataset | KMQGA | | | KMVGA | | |
|---|---|---|---|---|---|---|
| | Clusters | Ave. fitness | Std | Clusters | Ave. fitness | Std |
| Glass | 2 | 64.106 | 2.682 | 2 | 35.492 | 2.439 |
| Wine | 3 | 11.186 | 2.932 | 12 | 4.124 | 3.035 |
| SPECTF-Heart | 2 | 10.496 | 3.204 | 2 | 7.162 | 2.766 |
| Iris | 3 | 14.131 | 3.549 | 6 | 5.864 | 3.428 |

For each dataset, KMQGA and KMVGA run 30 times, respectively. For each dataset, we record the best fitness values KMQGA and KMVGA get for each run. And after 30 runs, we calculate the average best fitness value. And the average results are shown as follows in Table 6.

We can see from the results from Table 6 that, for all the four real datasets, KMQGA obtains the exact *K*. But the KMVGA just gets the optimal result in Glass dataset and SPECTF-Heart dataset and cannot get the exact *K* in another two datasets. For the average best fitness value, KMQGA gets the higher value than KMVGA does. In Glass dataset, KMQGA's best fitness value is 1.8 times higher than KMVGA's. In Wine dataset, KMQGA's best fitness value is 2.7 times higher than KMVGA's. In SPECTF-Heart dataset, KMQGA's best fitness value is 1.5 times higher than KMVGA's. In Iris dataset, KMQGA's best fitness value is 2.4 times higher than KMVGA's.

And we also calculated the standard deviation in the experiments of real datasets. In each dataset, the standard deviation that KMQGA obtains is similar to those of KMVGA, which means the stability of KMQGA is similar to that of KMVGA. Therefore, with the same stability, the better fitness that KMQGA gets means the better performance KMQGA makes.

In the real datasets experiment, we set both the same population size and the same generation numbers for KMQGA and KMVGA. KMQGA can achieve the better results in terms of the number of *K* and the average best fitness. Therefore, we can say that KMQGA is better than KMVGA in real datasets in our experiments.

From the above results of both the simulated datasets and the UCI real datasets, the effectiveness of KMQGA is proved. And the effectiveness of KMQGA is better than that of KMVGA.

## 5. Conclusion

In this paper, we propose an improved *K*-means clustering algorithm based on quantum-inspired genetic algorithm (KMQGA). This algorithm employs the Q-bit representation and the concept of quantum computing. Four main operations and an accessional operation (GA selection, GA crossover, GA mutation, quantum rotation operation and quantum catastrophe operation) are performed to search the optimal partition solution for a certain dataset. And the Q-bit string length can be changed in crossover operation. Due to this change, the partition denoted by a chromosome is adjusted, accordingly. Thus, the algorithm's searching space is large enough to get the optimal solution after several iterations of evolution. The simulated datasets in our experiment proved the correctness of KMQGA, and the UCI real datasets are performed to compare the difference between KMQGA and KMVGA in the effectiveness. The experiment results show that KMQGA is better than the KMVGA. Our future work is to investigate how to explore the search space using small number of individuals (even using only one individual).

## Acknowledgement

This work was supported in part by the NSFC Joint Fund with Guangdong under Key Project U0835002, the National High Technology Research and Development Program ("863"Program) of China, No. 2009AA01Z208 and the NSFC Funds with Nos. 60970044, 60736020, and 60673135.

## References

Abraham, A., Das, S., & Konar, A. (2006). Document clustering using differential evolution. In *Proceedings of 2006 IEEE congress on evolutionary computation* (pp. 1784–1791).

Bandyopadhyay, S., & Mauilk, U. (2001). Nonparametric genetic clustering: Comparison of validity indices. *IEEE Transactions on System, Man, and Cybernetics – Part C Applications and Reviews, 31*(1), 120–125.

Blake, C., Keogh, E., & Merz, C. J. (1998). *UCI repository of machine learning databases*. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

Bouldin, D. (1979). A cluster separation measure. *IEEE Transactions Pattern Analysis Machine Intelligence, 1*(2), 224–227.

Dunn, J. C. (1974). Well separated clusters and optimal fuzzy partitions. *Cybernetics and Systems, 4*(1), 95–104.

Halkidi, M., & Vazirgiannis, M. (2001). Clustering validity assessment: Finding the optimal partitioning of a data set. In *Proceedings of ICDM conference, CA, USA* (pp. 187–194).

Halkidi, M., & Vazirgiannis, M. (2002). Clustering validity assessment using multi representative. In *Proceedings of SETN conference, Thessaloniki, Greece* (pp. 237–248).

Han, K. H., & Kim, J. H. (2002). Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Transactions on Evolutionary Computation, 6*(6), 580–593.

Hartigan, J. A. (1975). *Clustering algorithms*. New York, NY: John Wiley and Sons, Inc..

Hey, T. (1999). Quantum computing : An introduction. *Computing and Control Engineering Journal, 10*(3), 105–112.

Jang, J. S., Han, K. H., & Kim, J. H. (2004). Face detection using quantum-inspired evolutionary algorithm. In *Proceedings of 2004 IEEE congress on evolutionary computation* (pp. 2100–2106).

Karima, B., Mouloud, K., Yacine, B., & Nadjib, B. (2006). Image segmentation using quantum genetic algorithms. In *Proceedings of IEEE 2006 – 32nd annual conference on industrial electronics* (pp. 3556–3563).

Krishna, K., & Murty, M. (1999). Genetic *K*-means algorithm. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics, 29*(3), 433–439.

Laszlo, M., & Mukherjee, S. (2007). A genetic algorithm that exchanges neighboring centers for *k*-means clustering. *Pattern Recognition Letters, 28*, 2359–2366.

Leung, Y., Zhang, J., & Xu, Z. (2000). Clustering by space–space filtering. *IEEE Transactions Pattern Analysis Machine Intelligence, 22*(12), 1396–1410.

Li, B. B., & Wang, L. (2007). A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling. *IEEE Transactions on System, Man and Cybernetics, Part B, Cybernetics*(37), 576–591.

Lu, Y., Lu, S., Fotouhi, F., Deng, Y., & Brown, S. J. (2004). FGKA: A fast genetic *k*-means clustering algorithm. In *Proceedings of the 2004 ACM symposium on applied computing* (pp. 622–623).

Mauilk, U., & Bandyopadhyay, S. (2002). Performance evaluation of some clustering algorithms and validity indices. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(12), 1650–1654.

Song, W., & Park, S. C. (2006). Genetic algorithm-based text clustering technique: Automatic evolution of clusters with high efficiency. In *Proceedings of the seventh international conference on web-age information management workshops* (pp. 17–24).

Steinbach, M., Karypis, G., & Kumar, V. (2000). A comparison of document clustering techniques. In *TextMining workshop, KDD*. <http://citeseer.ist.psu.edu/steinbach00comparison.html>.

Talbi, H., Draa, A., & Batouche, M. (2004). A new quantum-inspired genetic algorithm for solving the travelling salesman problem. In *Proceedings of 2004 IEEE international conference on industrial technology* (Vol. 3, pp. 1192–1197).

Turi, R. H. (2001). *Clustering-based colour image segmentation*. PhD thesis, Monash University, Australia.

Wang, L., Wu, H., Tang, F., & Zheng, D. Z. (2005). A hybrid quantum-inspired genetic algorithm for flow shop scheduling. *Lecture Notes in Computer Science, 3645*, 636–644.

Yang, S. L., Li, Y. S., Hu, X. X., & Pan, P. (2006). Optimization study on *k* value of *K*-means algorithm. *System Engineering – Theory and Practice, 26*(2), 97–101.

Zhang, D. X., Liu, X. Z., & Guan, Z. H. (2006). A dynamic clustering algorithm based on PSO and its application in fuzzy identification. In *Proceedings of the 2006 international conference on intelligent information hiding and multimedia signal processing* (pp. 232–235).