STUDY ON THE IMPROVEMENT OF THE FCM ALGORITHM BASED ON EVOLUTIONARY ALGORITHMS

A Thesis

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

ZHENBANG JU

James Keller, Thesis Supervisor

May 2016

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

STUDY ON THE IMPROVEMENT OF THE FCM ALGORITHM BASED ON EVOLUTIONARY ALGORITHMS

presented by Zhenbang Ju,

a candidate for the degree of Master of Science

and hereby certify that, in their opinion, it is worthy of acceptance.

James Keller

Alina Zare

Mihail Popescu

ACKNOWLEDGEMENTS

I would like to express my infinite gratitude to my advisor, Professor James Keller, for his continuous support of my master's study and related research. His great patience, enthusiasm, and immense knowledge were the most important factors in the completion of my studies. If it were not for Prof. Keller going over my thesis so many times and offering so many valuable suggestion, I would not have been able finish this thesis. Furthermore, I would like to thank Dr. David Fogel for his encouragement and guidance throughout my study of evolutionary algorithms. My sincere thanks also goes to my roommates, who gave me much encouragement and help in both my studies and life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

STUDY ON IMPROVEMENT OF FCM BASED
ON EVOLUTIONARY ALGORITHMS

Zhenbang Ju

Dr. James Keller, Thesis Supervisor

ABSTRACT

The Fuzzy C-Means (FCM) is a widely used clustering algorithm in unsupervised learning. It always converges to an optimum solution very quickly, thanks to its alternating optimization (AO) strategy. However, this AO strategy cannot guarantee the solution is global optimal. Hence, FCM may produce some counterintuitive results. On the other hand, evolutionary algorithms (EAs) are good at finding global optima in optimization problems. Hence, this thesis studied how to improve the FCM with the help of EAs.

The evolving FCM (EFCM) was first proposed. It had a slight higher performance in term of the Rand Index (RI) result measurement, but took much longer time compared with FCM. An improved EFCM approach was then proposed, which had a better performances and used less time than EFCM. What's more, the improved EFCM found better results which was not found by FCM. Finally, another related approach, modified genetic FCM (modified GFCM) was proposed by optimizing the existing improved GFCM. The modified GFCM greatly shorten the runtime compared to improved GFCM, but with no performance loss.

CHAPTER 1. INTRODUCTION

1.1 Evolutionary algorithms

Evolutionary algorithms (EAs) are adaptive population-based metaheuristic optimization algorithms. EAs are actually a family of algorithms, which are often used to find solutions to difficult optimization problems. EAs have been categorized according to their type. These types include genetic algorithms, evolutionary strategies, and evolutionary programming to name a few [1]. Each one of them may have different characteristics, so their use is flexible. But one thing is clear, based on the "no free lunch" principle, no single best type of evolutionary algorithm can solve a sufficiently broad range of difficult problems [2], [3]. Each algorithm will perform more efficiently in a particular subclass of difficult problems. That is, no one algorithm is worse than others and no one is better in all aspects. All depend on the differences in problems.

EAs are all developed by mimicking biology's evolving process, based on Darwin's evolution theory of natural selection and the genetic mechanism underlying the biological evolution process [4]. Therefore, they all follow a similar process, which normally includes reproduction and selection. EAs start with a population of random solutions, which represents a set of potential solutions. Each time the current set of solutions will generate new solutions by crossover and mutation. All previous solutions

along with the newly generated solutions form a new population. According to the "survival of the fittest" principle, not all current solutions will survive to the next generation, as the stronger algorithms replace them based on their fitness. This reproduction and selection processes results in later generations of algorithms that better fit the environment. Finally, the best solution in population will be a very good answer to the problem.

Usually, traditional optimization approaches start with a single initial solution and the optimal solution is obtained by iteration. However, this solution is sometimes easily trapped in a local optimum, and can never come out. EAs are also iterative algorithms, but they start with a set of solutions, instead of a single solution. A great difference separates EAs from traditional optimization approaches. The risk of falling into local optimum is reduced by processing a set of solutions at the same time in EAs. Basically, EAs do not require extra knowledge or information of how to process the objective functions. EAs only require the objective functions to evaluate the fitness of solutions. There's no need of requiring the continuity nor differentiability of objective functions. These characteristics largely extend the application scope of EAs. Due to EAs' high robustness, they can be widely applied to solve different types of problems for different disciplines such as engineering, computer science, physics, and chemistry [5].

Fuzzy C-Means (FCM) is a frequently used clustering algorithm in pattern recognition, which was developed by Dunn in 1973 [6] and improved by Bezdek in 1981 [7]. FCM

is a very fast clustering algorithm. However, its main criticism comes on occasions when a problem has multiple optimal solutions, and FCM is unable to find the best one. Hence, the goal of this thesis was to study how EAs can improve its performance when using FCM. The thesis is constructed as follows. Chapter 2 will show an overview of the concepts of each EA component. Chapter 3 will study the basic features of the EAs in terms of the parameters. Chapter 4 will propose a basic evolving FCM (EFCM) approach to use EAs to optimize FCM's objective functions; Chapter 5 proposes a new multi-level strategy of selecting the parameters of evolving FCM; Chapter 6 proposes an improved EFCM, and a modified genetic FCM (modified GFCM) which is modified as the basis of an existing improved GFCM algorithm.

CHAPTER 2. EVOLUTIONARY ALGORITHMS OVERVIEW

Historically, the first use of Darwinian principles to solve problems appeared in the 1950s. Then three different ideas, which have had the most impact on the field as we see it today, were developed different places in the 1960s [1]. Evolutionary programming was first introduced by Lawrence J. Fogel [8]. John Henry Holland called his method a genetic algorithm [9]. The third algorithm was introduced by Ingo Rechenberg [10] and expanded by Hans-Paul Schwefel [11] and is known as evolution strategies. In the 1990s, the fourth idea, genetic programming, came out [12]. At the same time, all of these nature-inspired algorithms were unified as one technology–evolutionary computing. From that time on, with the dramatic increases in the power of computers, evolutionary computation entered a high-speed development period. Evolutionary computation theory and application of evolutionary computation have both become very popular research topics. Now, evolutionary algorithms can be widely used to solve multi-dimensional optimization problems more efficiently than off-the-shelf software. Evolutionary algorithms are also used to optimize the system designs [13].

Even though EAs may have different forms, most of them still have a common evolving model based on the Darwinian evolutionary system. Normally, EAs start with a population, which makes up the potential solution set for the optimization problems. A

population consists of a certain number of solutions (called individuals). Each solution is encoded into a series of genes. Each series of genes can be seen as a chromosome that can represent all characteristics of a solution. These initial solutions are usually randomly allocated within the search space. The solutions are evaluated by an objective function of the problem. Part of the solutions are chosen to survive based on selection methods. These survivors then generate their offspring using some reproduction operators, such as crossover and mutation. All offspring together with their parents form a new population for the next generation. This evaluating, selecting and reproducing process repeats until a stopping condition is met. The following sections in this chapter will introduce each part of this process in detail.

2.1 Representation

The first step in solving an optimization problem using EA is to determine the encoding method of solutions. All needed characteristics are encoded into a series of genes in a certain structure, so the framework can be accessed and processed via computer. The characteristics can not only include numerical parameters, but also any kind of factors that are needed to describe the solutions. This thesis focuses only on the numerical optimization problems. This encoding of characteristics process is referred to be as representation.

Usually, all possible candidate solutions for the problem should be able to be represented by chromosomes. The binary representation is a very famous approach. However, with the development of computer power and the changing of coding styles, the real-value representation method has also been used a lot. For example, since this thesis focuses only on numerical optimization problems, a solution $\mathbf{x}$ of a d-dimension numerical optimization problem can be represented by $[x_1, x_2, ..., x_d]$, where x_1 is the real- valued number of $\mathbf{x}$ in the i-th dimension. One variable can map directly into one dimension, and so, this approach makes solutions easier to understand, reducing the work needed to encode and decode. My experiments in this thesis also used this real-valued representation structure.

2.2 Fitness

Every solution in the population is assigned a fitness value. This fitness is a measure that tells how well the solution fits the objective function, that is, how well a solution achieves our expectation in term of objective function. The task of EAs is to find solutions that have the highest fitness. The more fit some solutions are, the more likely they will be selected from the current population. This leads the population toward better and better. However, the fitness values of all solutions must be calculated at each generation, which means that, in most EAs problems, fitness evaluation is one of the most important factors; however, it also increases computational complexity.

2.3 Genetic operators

The population repeats reproduction and elimination in order to evolve and get better. This happens over and over again. Typically, three main genetic operators are applied in this process. They are mutation, crossover and selection. Mutation and crossover are operators used for generating new solutions, and the selection operator is used to eliminate bad and undesirable solutions in the current population.

Mutation is an operator that mimics the role of mutation in natural evolution. In real-valued EAs, mutation is usually performed by adding a Gaussian random variable with a zero mean and a certain standard deviation to the genes. Mutation yields new variations in the solution space. Hence, mutation is a main approach that can give some help when the population is trapped in a local minimum. In real-valued EAs, mutation may also be regarded as a main searching tool.

Crossover is another important operator that mimics sexual reproduction to generate offspring from multiple parents. Crossover is completed by mixing genes from one solution with genes from another solution. Some commonly used crossover approaches are one-point crossover, two-point crossover, and uniform crossover. The crossover process tries to communicate and spread good genes from some solutions to the whole population so all solutions can move toward the best position quickly. However, there are some problems here. First, crossover speeds up the process of convergence. But this

sometimes causes solutions to fall into local optima instead of the global optimum. Besides, crossover only spreads genes; it doesn't create new ones. If some desired genes do not exist in the current population, of course, they will never appear later. So mutation is usually needed to bring fresh genes into the population, and this happens during the crossover operation.

Different from mutation and crossover, selection is an operator that simulates the "survival of the fittest" to eliminate solutions. The key idea of selection is to give preference to better solutions [14]. Some common selection approaches are plus/comma selection, proportional selection and tournament selection. All of these approaches try to make better solutions more likely to pass on their genes to the next generation, and to prevent worse solutions from getting into the next generation. Hence, population solutions should get fitter and fitter as generations proceed.

2.4 Termination

EAs need some conditions that signal when to terminate the evolving process. Normally, the process stops when:

- A solution is good enough to satisfy the purpose of the problem.
- The best solution's fitness becomes stable after no better results are produced in many generations

- The number of generations reaches a certain value
- A combination of the above

2.5 Benchmarks

To learn and evaluate any expected aspect of tested algorithms, we also use some benchmarks, that is, some different objective functions. This thesis first applied two objective functions in the following study on EAs. They are sphere function and Rastrigin's function.

2.5.1 F1: Sphere Function

The sphere function is a smooth, unimodal and symmetric function. It has only one optimum, which can be found at $f_1(\mathbf{0}) = 0$. This is a very simple objective function that is often used. The function can be described as:

$$f_1 = \sum_{i=1}^{D} x_i^2 \tag{6.1}$$

where D stands for the dimensionality of the function, and $x_i \in [-10,10]$ is used in this thesis. A surface of the function where D = 2 is shown in Figure 2.1.

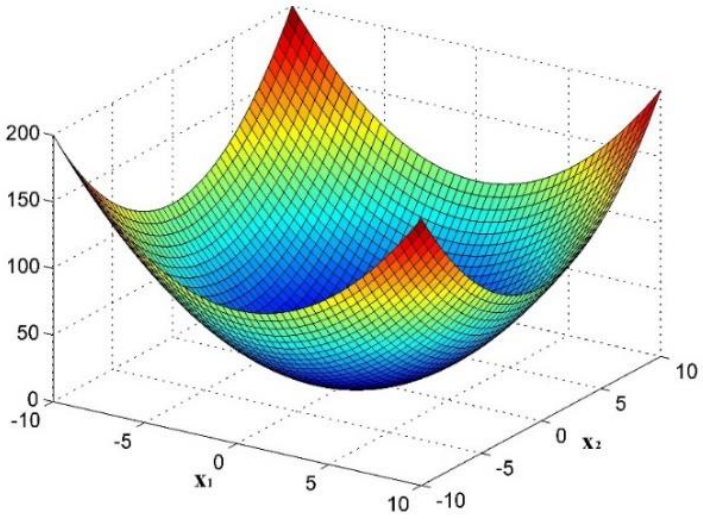


Figure 2.1 Surface of sphere function where D = 2.

2.5.2 F2: Rastrigin's Function

Rastrigin's function is considered to be a very difficult problem for optimization because of its large number of local optimums. The function is:

$$f_2 = \sum_{i=1}^{d} (x_i^2 - 10\cos(2\pi x_i) + 10) \tag{6.2}$$

where d stands for the dimensionality of the function, and $x_i \in [-10,10]$. Its global optimum can be found at $f_2(\mathbf{0}) = 0$. A surface of the function where D = 2 is shown in Figure 2.2.

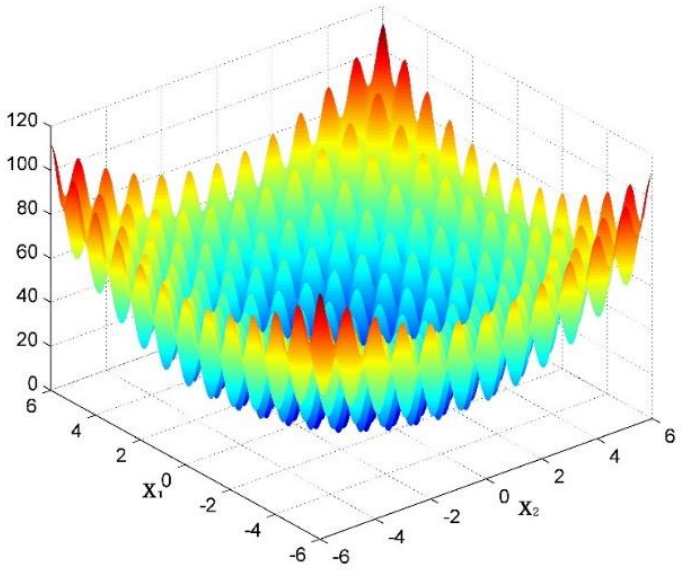


Figure 2.2 Surface of Rastrigin's functions where D=2.

CHAPTER 3. EVOLUTIONARY OPTIMIZATION

3.1 Basic Approach

3.1.1 Method

This chapter starts with one basic evolution approach, which can be seen as the cornerstone from which all following studies in this chapter are extended. This basic evolution approach for optimization can be described as expressed in the steps below [15], [16].

I. Begin, and set g (Generation) = 1.

II. Create the initial solutions. Totally, $\mu + \lambda$ solutions are created and randomly distributed within the whole search space. All these $\mu + \lambda$ solutions form a population. For the D-dimension numerical optimization problems in this thesis, every solution is simply encoded into a vector with D elements. Each element is a gene, which stands for a real-valued variable in its corresponding dimension in search space.

III. Evaluate the current population, which is based on the objective function. Rank all the solutions in the current population, keep the best μ solutions and eliminate all other solutions from the current population. Then increase $g = g + 1$.

IV. Generate offspring step 1 – Crossover. Each time, uniformly random select two

solutions from the current population (with replacement). They are the parents, who will generate a new child solution. Each gene of this child solution has a 50–50 chance to inherit the allele from either one of its parents. This process is called the uniform crossover. Next, two more solutions are selected to generate another child solution, and this generation process stops when λ child solutions are generated.

V. Generate offspring step 2 – Mutation. Next, all λ new generated child solutions get into the mutation process. This thesis applied one common used mutation strategy where each gene of a child solution has a chance (referred to as the mutation rate, P_{mutate}) to mutate; otherwise it should stay in place (at its point of origin). If a gene is determined to mutate, it will be added to a Gaussian random variable with zero mean and a certain standard deviation, σ. For example, if the i-th gene, x_i, of a solution, $\mathbf{x} = [x_1, x_2, ..., x_d]$, is going to mutate, then the mutated new gene x_i' will be:

$$x_i' = x_i + N(0, \sigma) \tag{3.1}$$

where N(0, σ) represents a Gaussian random variable with zero mean and a certain standard deviation, σ. After all λ solutions finish their mutation process, they are put into the current population together with previous solutions.

VI. Repeat from Step III to Step V until the best solution's fitness is stabilized so that there is no change for the best obtained solution to occur in t generations, where t is called the waiting generations.

The flow chart for this process is shown in Figure 3.1.

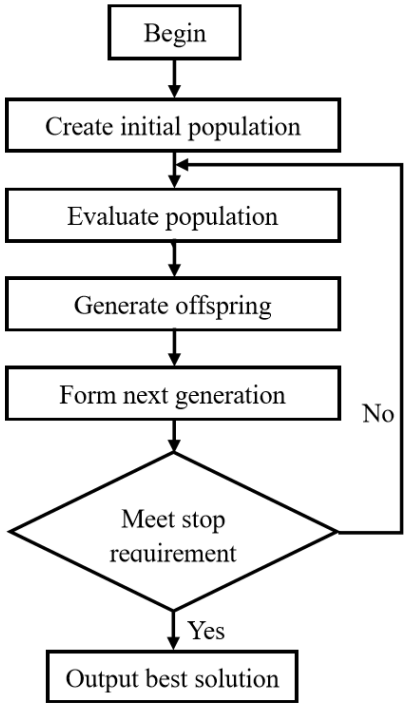


Figure 3.1 A flow chart of evolutionary algorithm construction

3.1.2 Experiments and Results

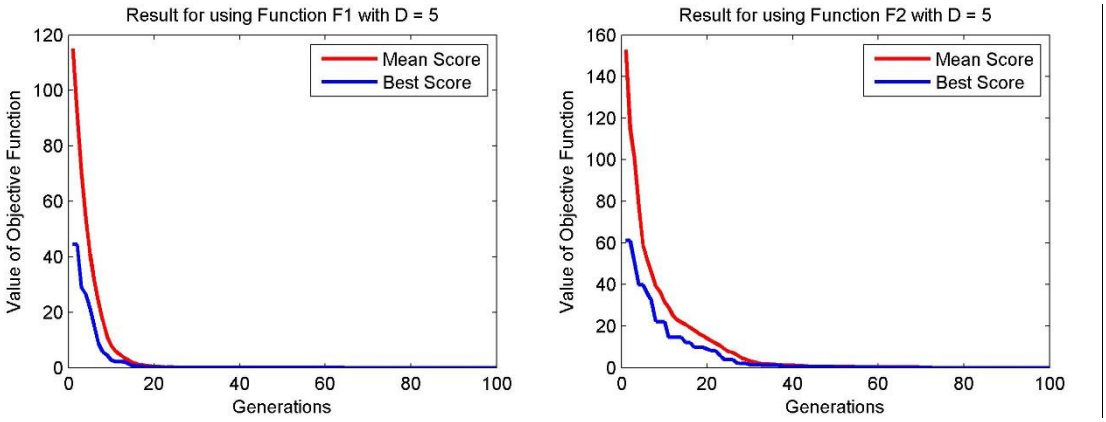
To show the process of this evolutionary algorithm, experiments are performed to find the minimum value of objective Function F1 and F2, respectively. The best score found for each generation and mean score of all solutions at each generation are shown in Figure 3.2 by using the blue and red curves. In the experiments, the following parameters were used:

Population size: $N = (\mu + \lambda) = (30 + 30)$

Mutation rate: $P = 0.1$

Mutation variance: $\sigma = 1$

Dimensionality: D = 5



(a) (b)

Figure 3.2 Plots of best and mean score found at each generation for the two objective functions: (a) optimization process to function F1 with D = 5 & (b) optimization process to function F2 with D=5

It is encouraging to see how the best solution's score at each generation descended very quickly as expected. It needs only about 20 generations to get the minimal score as it relates to the objective function F1, and less than 40 generations for the objective function F2. At the same time, the mean score at each generation, which represents the overall level of all solutions at that generation, also evolved showing better and better results throughout the process. Finally, the whole population of solutions achieved zero score, for each objective function.

With the prior knowledge, we know that both objective functions' global minimums are 0 and can be obtained at $(0, 0, \ldots, 0)_D$ where D is the dimensionality of the objective functions. For Function F2, it also has many other local minima values, which are larger than zero, such as $f_2(\mathbf{x}) = 0.995$, where $\mathbf{x} = (0.995, 0, 0, \ldots, 0)_D$, for example, is a

secondary minima of Function F2. This means this evolutionary approach found the global optimal solution successfully.

3.2 Further Study on EA

The previous section gave a first impression of a basic evolutionary algorithm, which can find the global optimal solution for both benchmarks. To learn the main characteristics of this evolutionary algorithm in detail, this chapter presents a step by step delineation, in order to establish the foundation needed to understand the FCM applications in the following chapters.

3.2.1 Crossover Only

As previously mentioned, only two operators serve in the reproduction process of EAs, crossover and mutation. Either one of them is actually enough to be used to generate new different children. Hence, a very interesting question arises as to whether only one of them is enough to make the algorithm work. First, let's consider the crossover only approach, which throws away the mutation process and all child solutions are generated only by crossover. This crossover only approach is described as:

I. Begin, and set g (Generation) = 1.

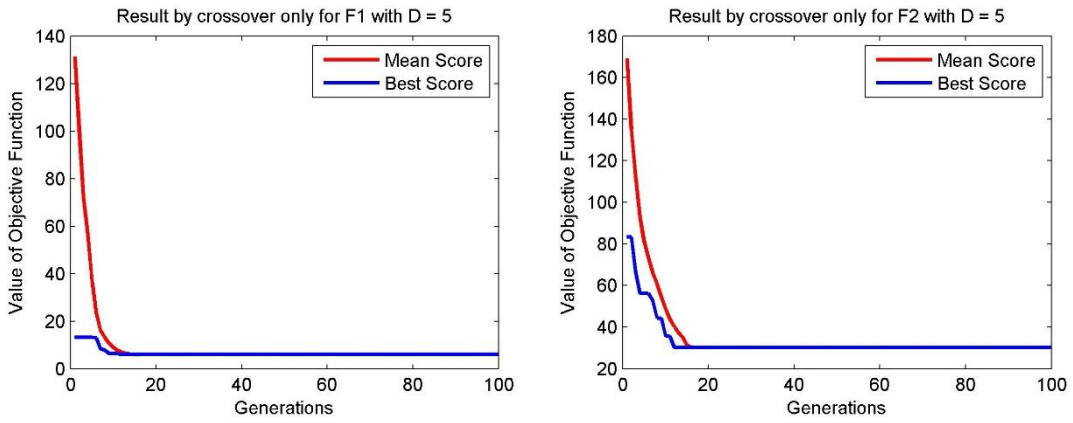
II. Create the initial solutions. Totally, $\mu+\lambda$ solutions are created and randomly distributed within the whole search space. All these $\mu+\lambda$ solutions form a population.

III. Evaluate the current population, which is based on the objective function. Rank all the solutions in the current population, keep the best μ solutions and eliminate all other solutions from the current population. Then increase g = g + 1.

IV. Generate offspring–crossover. Each time, uniformly random select two solutions from current population (with replacement). They are the parents, who will generate a new child solution. Each gene of this child solution has a 50–50 chance to inherit the allele from either one of its parents. This process is called the uniform crossover. Next, two more solutions will be selected to generate another child solution, and this generation process then stops until λ child solutions are generated.

V. Repeat Steps III and IV until the best solution's fitness is stabilized and no change is needed to obtain the best solution for the t generations, where t is known to represent waiting generations.

Perform this approach on both objective functions with D = 5. The curves of best score and mean score at each generation are shown in Figure 3.3 as blue and red curves, respectively. This time, the evolving process stops much earlier, about 10 generations for F1 and less than 20 generations for F2. However, the solutions' scores found this time are much worse than previous experiments, which had scores that were all zero. That is, the solutions found this time are not well optimized when the evolving processes stop.



(a) (b)

Figure 3.3. Best and mean score are found at each generation obtained by crossover only approach. (a) optimization process to function F1 with D=5 & (b) optimization process to function F2 with D=5

This experiment gives one characteristic to crossover. The crossover only spreads genes from parent solutions to child solutions, which means no new type of genes is entering the population. Hence, the solutions found by the crossover only approach can only be one of the possible combinations of existing genes. In other words, if the genes needed by a global optimal solution are not created in the initialization process, they will never appear during the evolving process, and the global optimal solution will not be found.

3.2.2 Mutation Only

Recalling the process of mutation, genes are added to a random variable, which always makes them different from before. Hence, if the crossover is spreading genes, the mutation process will bring a lot of new features. This leads to another question: How about the mutation only approach? Since the mutation process only changes one internal solution, the reproduction process of the algorithm should also have some modification. The modified approach is shown below:

I. Begin, and set g (Generation) = 1.

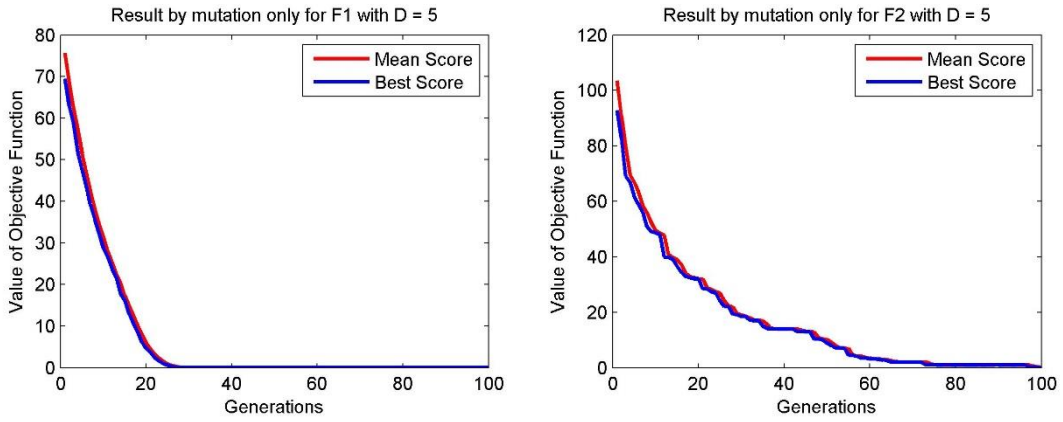
II. Create the initial solutions. Totally, $\mu+\lambda$ solutions are created randomly distributed within the whole search space. All these $\mu+\lambda$ solutions form a population. For the d-dimension numerical optimization problems in this thesis, every solution is simply encoded into a vector with d elements. Each element is a gene, which stands for a real-valued variable in its corresponding dimension in search space.

III. Evaluate the current population, which is based on the objective function. Rank all the solutions in the current population, keep the best μ solutions and eliminate all other solutions from the current population. Then increase g = g + 1.

IV. Generate offspring mutation. Each time, randomly select a solution from the current population with a replacement. Perform a mutation on this solution, and the mutated solution will be seen as its child solution. Then select another solution to generate another child solution. Repeat this breeding process until λ child solutions are generated.

V. Repeat Step III and Step IV until the best solution's fitness is stabilized that there is no change for the best obtained solution in t generations, where t represents waiting generations.

Again, perform this approach to optimize the two-objective function with D = 5, and the curves of best score and mean score at each generation are shown in Figure 3.4 as blue and red curves, respectively,



(a) (b)

Figure 3.4 Best and mean score found at each generation obtained by the mutation only approach. (a) optimization process to function F1 with D = 5 & (b) optimization process to function F2 with D=5

This time, the value of both objective functions also achieved zero, which is the global optimal value we expected for both functions. That is, this mutation-only approach also solved the two optimization problem. However, if we take another closer look at this mutation-only approach in Figure 3.4, two big differences stand between it and previous crossover plus mutation approach (see also Figure 3.2).

First, it takes some more generations for the mutation only approach to achieve zero score, especially for the function F2, where it used only about 40 generations in the crossover plus mutation approach, while getting almost the same score at Generation 100 here. Second, the score of best solution obtained at each generation is much smaller than the mean score of the whole population in crossover plus the mutation approach,

but the best score at each generation is obtained by the mutation-only approach, which is almost the same as the mean score. This means that the crossover and mutation approach may generate some solutions, which are much better than themselves, but the mutation-only approach is not able.

The reason for this is simple. In the mutation-only approach, each child inherits genes only from its single parent, but in crossover plus mutation approach, each child gets genes from two parents. As a result, in the in-mutation only approach, each solution must find all fit genes by itself. However, in the crossover plus mutation approach, if a solution has a good gene, and another solution has another good gene, the exchange of good genes can help both of them omit the process of looking for genes, which are also owned by each other.

For example, assume two solutions exist for function F1, $x_1 = (1,4)$ and $x_2 = (4,1)$. In the mutation only approach, both solutions can only do the searching work separately, until they find ways from $(1,4)$ or $(4,1)$ to $(0,0)$. On the contrary, in crossover plus mutation approach, these two solutions may crossover and generate a powerful combined child $x_3 = (1,1)$, who takes advantages from both of its parents and becomes much more evolved. Much time is saved and its following searching task only need to start from (1,1) to (0,0).

3.2.3 Mutation Rate

From the experiments in previous sections we can know that, the crossover can speed up the evolving process by recombining the existing genes, and mutation brings new different genes into population so that the population can continue to evolve better. The cooperation of crossover and mutation makes the evolving process able to quickly achieve a good solution to the optimization problem. So my following study on evolutionary algorithms will only focus on the basic evolutionary algorithm, which consists of both crossover and mutation (described in Section 3.1), even though the mutation-only approach may also complete the optimization task, and is also used by many people.

In natural sexual reproduction, the mutation rate is very low, which can make the living things stable and evolve slowly. This is because we have already more or less adapted our current environment and the environment is changing very slowly. But, it is unsuitable here for our evolutionary algorithm, which tries to find the best solution quickly. Then the problem comes: How much of mutation rate should be used for gene mutation?

To study this, the basic crossover plus mutation-based evolutionary algorithm is applied, with different mutation rates $\{1.00, 0.50, 0.10, 0.05, 0.01\}$. Twenty independent trials for each mutation rate were performed, where every trial for all experiments started with a different random initialized population set. The number of generations used and the

score of the best solution obtained for each trial were recorded and are shown in Figure 3.5 with x-axis and y-axis, respectively. Each point in the figure is a solution obtained from a trial, the different legends represent the different mutation rates used for that solution. The other parameters used are:

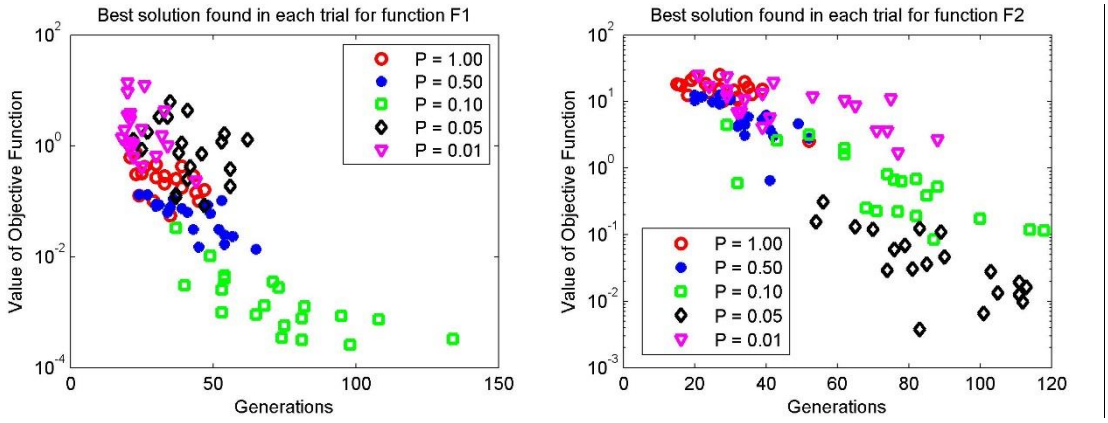
Population size: N = (μ+ λ) = (30 + 30)

Maximum generations: 500

Number of waiting generations: t = 10

Mutation variance: σ = 1

Dimensionality: D = 5.



(a) (b)

Figure 3.5 Best solution obtained at each trial by using different mutation rates
(a) solutions obtained each trial for F1 and (b) solutions obtained for each trial for F2

For function F1, generally, the mutation rate 0.10 is the best choice, since in each trial, the evolution algorithm can find a solution, whose score is smaller than 10^{-2}, as the green squares show. The solutions obtained by using the mutation rate 1.00 and 0.50 are much worse; their values in term of function F1 vary from about 10^{-2} to 10^{0}, that is

0.01 and 1. These may be seen as good results, since they are a bit far away from the expected point (0, 0). The solutions obtained by using mutation rate 0.05 and 0.01 are even worse, with scores varying from 0.1 to 10.

The reasons for both high and low mutation rates resulting in bad outcomes are not the same. For high mutation rate, like 1.00, most genes of a child will change to be different from its parents, and the changes are all random. If any change in its genes is bad, then this child might not be better than its un-mutated status. In other words, the high mutation rate makes it much harder for the child solutions to evolve better, so that the evolving process will stop early if no better solutions were found in several generations.

On the other hand, the evolving process with low mutation rates also stopped early since no better solutions were found. Imagine the mutation rate was 0.01 and had a solution with 5 genes. That is, each gene in this solution has 0.01 chance to mutate, and the total probability of this solution not changing after mutation is high—up to 0.99^5 (=0.951). It is very likely that, all solutions in the population will not change after the mutation process. Hence, the low mutation rate makes the solutions harder to change, let alone find solutions that will become better.

Furthermore, the mutation rate of 0.10 achieved the best performance for function F1, but 0.05 was better for function F2. That means the fittest mutation rate may be different for different problems. This increases the difficulty in determining the best value for

mutation rates. To solve this, one method is to try some different values and choose the ones that most fit in term of objective function value. However, as Figure 3.5 shows, with the same mutation rate, the best obtained solutions are not the same. So this method needs a lot of repeated trials for each candidate mutation rate, which is very inconvenient and time consuming. Later in Chapter 5, a multi-level strategy will be proposed, which can successfully overcome this problem in evolving FCM.

3.2.4 Mutation Step Length

Besides the mutation rate, another parameter, mutation step length, σ, is also very important in evolutionary algorithms. It decides to what degree the mutation will perform. This section studies this parameter by reapplying the evolutionary algorithm with different mutation steps, $\{10, 1, 0.1, 0.01\}$. Twenty independent trials for each mutation step were performed. Each trial started with a different initialized population set. The number of generations used and the score of the best obtained solution for each trial were recorded and are shown in Figure 3.6 with x-axis and y-axis, respectively. Each point in the figure is a solution obtained from a trial. The different legends represent the different mutation rates used for that solution. The other parameters used are shown as:

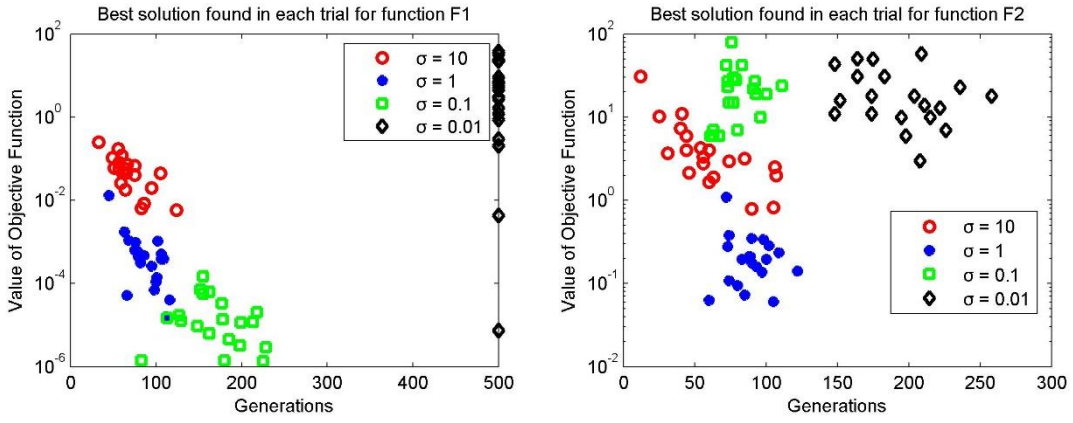
Population size: $N = (\mu + \lambda) = (30 + 30)$

Maximum generations: 500

Number of waiting generations: t = 10

Mutation rate: $P_{mutate} = 0.1$

Dimensionality: D = 5



(a) (b)

Figure 3.6 Best solution obtained at each trial by using different mutation steps as shown by (a) solutions obtained each trial for F1 & (b) solutions obtained each trial for F2

Optimization results are shown for function F1 in Figure 3.6(a). This figure also shows a situation where either large mutation steps or small steps will reduce the performances of the algorithm. For the large steps, the solutions were harder to mutate as improved especially when they were close to the optimal points. For example, assume we want to find the minimum value of function F1 with D = 1, and have a solution x = 0.1. If this solution mutates by adding a Gaussian random variable with a zero mean and standard deviation σ = 10, the probability of a mutated solution x + N(0, σ) belonging to [−0.1, 0.1] is obviously much smaller than if we use the step σ= 0.1. Hence, a large

mutation step will make the mutation hard to get better solutions, especially when the solutions are already close to the optimum point. This is what made the evolving process stop early. In other words, small mutation steps could have a stronger local search ability, so they can still continue the searching when solutions are close to the target.

On the other hand, the solutions obtained by applying a very small mutation step, 0.01, are also obtained at Generation 500, which is the maximum generation. That is, these solutions did not finish their evolving process, but only because the maximum threshold was met. Hence, one disadvantage of the small mutation step appears. It takes a very long time for solutions to complete evolving with a small mutation step because they change very little each time.

For the output for function F2 in Figure 3.6(b), first the large mutation step makes solutions worse following the previous pattern. Then for the small mutation steps, solutions obtained were also very bad, but this time they were not caused by n unfinished evolving process. Every trial stopped before Generation 300, which means, they were unable to improve their evolution when they stopped. That is, they were actually stuck at some local optimal positions and could not find a way out with such a small searching step. This is the second disadvantage of small mutation steps.

In summary, larger mutation steps may cause solutions to not optimize as they should, while the smaller steps take a much longer time to get into the optimal positions, where a high risk exists that the position is at a local optimal position. What's worse, similar to the mutation rate, the fittest mutation step can also be uncertain for different problems. It seems the value of 0.1 is better for F1, but F2 needs a higher value than 1. This problem was solved by a proposed multi-level strategy associated with an evolving FCM in Chapter 5.

CHAPTER 4. EVOLVING CLUSTERING

4.1 Introduction

Clustering is a task that groups a set of objects, such that objects in the same group (called a cluster) are more similar than objects in different clusters, which are less similar [17]. It is usually used in unsupervised learning. Many existing clustering algorithms try to find their expected clustering results by optimizing some defined objective functions. Normally, the clustering algorithms optimize objective functions by applying iterative approaches. For example, some famous clustering algorithms, K-Means, Fuzzy C-Means (FCM), and Possibilistic C-Means (PCM), use an iterative technique called alternating optimization (AO), which updates cluster memberships and cluster centers alternatively to find an optimum solution. Hence, the optimization processes are very simple and fast.

However, this kind of alternating optimization has an underlying defect. The optimization process is very sensitive to initial status, so that sometimes it cannot find the global optimal solution and, therefore, can only stop in a local optimal solution. This may result in a case where the clustering result we get is not expected or ideal. Hence, this chapter utilizes evolutionary algorithms to solve the problem of getting stuck in a local optimum, focusing on the FCM.

4.1.1 Fuzzy C-Means

Fuzzy C-Means is a frequently used clustering algorithm in pattern recognition, which was developed by Dunn in 1973 [6] and improved by Bezdek in 1981 [7]. FCM is a very fast clustering algorithm. The process of FCM can be described as follows. A set of objects $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ is given, and each object $\mathbf{x}_i$ is a d-dimensional real-valued vector. FCM tries to cluster all objects into C clusters, i.e., $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_C\}$. FCM introduces a notion of "Fuzzy Logic", so that, each object $\mathbf{x}_i$ is described by a factor, u_{ji}, called membership, which indicates the degree of how much this object $\mathbf{x}_i$ belongs to cluster S_j, where $0 \le u_{ji} \le 1$. Together with all memberships, $U_{C\times n} = \{u_{ji}\}$, is called the partition matrix, and it is subject to a constraint wherein the sum of an object's memberships belonging to all clusters should be one:

$$\sum_{j=1}^{C} u_{ji} = 1 \text{ for all objects } \mathbf{x}_i \tag{4.1}$$

Then, the task of FCM is to minimize the following objective function:

$$\begin{aligned} J_{FCM} &= \arg\min \sum_{j=1}^{C}\sum_{i=1}^{n} u_{ji}{}^{m} \left\|\mathbf{x}_i - \mathbf{v}_j\right\|^2 \\ &\text{subject to } \sum_{j=1}^{C} u_{ji} = 1 \end{aligned} \tag{4.2}$$

where C is the number of clusters and n is the number of objects. $\mathbf{x}_i$ is the i-th object, and $\mathbf{v}_j$ is the j-th cluster's center, and m is another introduced concept, the fuzzifier, where $m \ge 1$. This fuzzifier determines the degree of cluster fuzziness. A larger m yields

more similar degrees of membership to objects within all clusters. Usually, m is set to 2 and this is what is used for all experiments in this thesis. In addition, $\left\|\mathbf{x}_i - \mathbf{v}_j\right\|^2$ is the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{v}_j$. This is what is usually used and what is used in this thesis, but other distance functions can be used as well. To optimize Eq. (4.2), one common approach is to construct a new equation with the help of the Lagrange multiplier, and thereby obtain:

$$J_{FCM} = \arg\min \sum_{j=1}^{C}\sum_{i=1}^{n} u_{ji}{}^{m} \left\|\mathbf{x}_i - \mathbf{v}_j\right\|^2 - \sum_{j=1}^{C} \lambda \left(\sum_{i=1}^{n} u_{ji} - 1 \right) \tag{4.3}$$

where the newly added factor is always equal to 0 as Eq. (4.1) describes. To find the minimum value of J_{FCM}, the partial derivative of every variable in Eq. (4.3) must be zero, which is a necessary condition for a minimum in a function. Hence, all partial derivatives of Eq. (4.3), which should be met first, are listed below:

$$\frac{\partial}{\partial \lambda} J_{FCM} = \sum_{j=1}^{C} u_{ji} - 1 = 0 \tag{4.4}$$

$$\frac{\partial}{\partial u_{ji}} J_{FCM} = m u_{ji}{}^{m-1} \left\|\mathbf{x}_j - \mathbf{v}_i\right\|^2 - \lambda = 0 \tag{4.5}$$

$$\frac{\partial}{\partial \mathbf{v}_i} J_{FCM} = \sum_{j=1}^{n} u_{ji}{}^{m} \times 2(\mathbf{x}_j - \mathbf{v}_i) = 0 \tag{4.6}$$

Summarizing Eq. (4.4) and Eq. (4.5), we get:

$$u_{ji} = \frac{1}{\sum_k \left(\frac{\|\mathbf{x}_i - \mathbf{v}_j\|}{\|\mathbf{x}_i - \mathbf{v}_k\|} \right)^{\frac{2}{m-1}}} \tag{4.7}$$

Solving Eq. (4.6), we obtain:

$$\mathbf{v}_j = \frac{\sum_i u_{ji}^m \mathbf{x}_i}{\sum_i u_{ji}^m} \tag{4.8}$$

Finally, we can update the memberships u_{ji} and cluster centers $\mathbf{v}_j$ alternately until both of them do not change. Then they could be considered as a candidate which may be a minima of the objective function. The complete process is written as:

Step 1. Randomly generate an initial set of cluster centers $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_C\}$. To keep the consistency with other approaches, I decided to initialize all initial cluster centers $\mathbf{v}_j$ uniformly at random within the domain of a data set, and also for all later experiments in this thesis.

Step 2. Update every membership u_{ji} by applying Eq. (4.7).

Step 3. Update every cluster center $\mathbf{v}_j$ by applying Eq. (4.8).

Step 4. Repeat Step 2 and Step 3, until the change of J_{FCM}, ΔJ_{FCM}, is less than a preset convergence threshold, ε, where ε is set 10^{-8} for every experiment in this thesis.

In order to give a preliminary impression of FCM, a simple data set, called the butterfly data, is used for testing. This data set consists of 15 two-dimensional points, which are shown in Table 4.1. Apply the FCM algorithm described above with C = 2 and m = 2 to the butterfly data set, and results like those found in Figure 4.1 can be obtained where the centers shown as two red stars are found. The result of Figure 4.1 satisfies our expectation very well: one center is at the left and one is at the right, which will result, when the final partition matrix is hardened, in all points at left forming one cluster and the others forming another one. It should be noticed that, the point in the center of data space that is, (3, 2) actually has the same membership, 0.5, to both clusters.

Table 4.1 Fifteen points from butterfly data set

POINT	X	Y
1	0	0
2	0	2
3	0	4
4	1	1
5	1	2
6	1	3
7	2	2
8	3	2
9	4	2
10	5	1
11	5	2
12	5	3
13	6	0
14	6	2
15	6	4

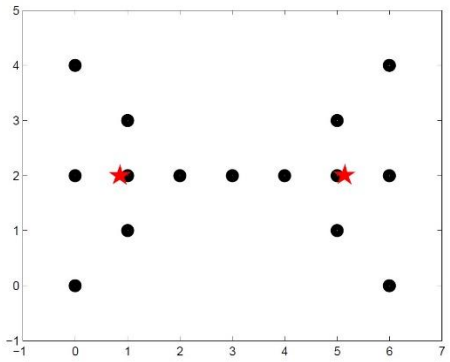


Figure 4.1 The clustering result obtained by FCM for butterfly data set.

4.1.2 Unexpected Situation

Before, FCM showed a pretty good clustering result. However, as previously mentioned, these alternating update algorithms are very sensitive to the initial status. The previous data set is very simple, but it also truly has a local optimum solution. Consider the data set shown in Figure 4.2, which is another very interesting data set that can help the operator to learn more about local optima, where there are seemingly four clusters, each of which is represented by a different color.

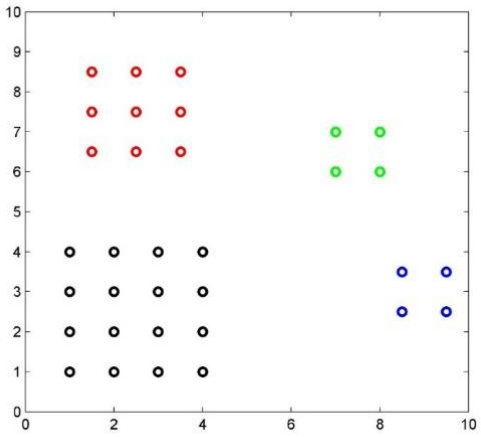
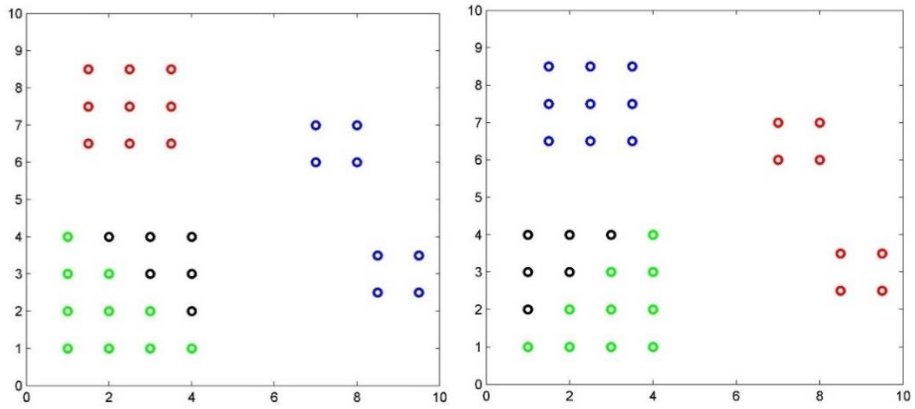


Figure 4.2 The new Data Set 1 used for studying local minima

For this data set, 100 independent trials of the FCM were run with parameters C = 4 and m = 2. Since the FCM began with a different set of C uniformly at random initialized centers in each trial, many different outcomes were noted when the partition matrices were hardened. First, of course, is the case replicated in Figure 4.3. However, there are only 9 out of 100 trials that obtained the correct, or expected, solution, results like Figure 4.3. As for all other trials, they have the same problem where the right two small clusters, green and blue, were grouped to be in one cluster, while the biggest black cluster at left bottom is divided into 2 clusters, the two most common unexpected results as shown in Figure 4.3.

To be more precise, the value of the objective functions for both algorithms are calculated for these cases, and shown in Table 4.2. From this table, it is very clear that the expected clustering results have the least value of objective function. This means the clusters shown in Figure 4.2 not only satisfy our expectation, but also represent the real optimum solution in terms of the objective function.



(a) (b)

Figure 4.3 Unexpected results obtained from clustering shown here as (a) Case 1 and (b) Case 2.

Table 4.2 Values of objective functions for different cases

	JFCM
Expected (Figure 4.4)	44.45
Unexpected (Figure 4.5a)	49.77
Unexpected (Figure 4.5b)	49.48
Other results	≥49.78

To conquer the local optimum problem above, we can run the FCM many times to choose the best results in terms of the objective function's value. However, it cannot guarantee how many times the FCM can obtain a desirable result. What's worse, sometimes the FCM can only find some secondary results, but not the truly best one (which will be shown later in Section 5.4). Another better way is to apply evolutionary algorithms to optimize the objective function of FCM. Since evolutionary algorithms can successfully find the global optimum in many different situations, EAs are introduced in later chapters, to help find better results for FCM.

4.2 Preparation Work

4.2.1 Data Sets

To study the performance of evolving clustering approaches and have some comparisons, four data sets will be used in the later experiments in this chapter. One is a simple approach used earlier in Section 4.1.2, in Figure 4.2, referred to as Data Set 1.

The second data set consists of four rectangular blocks, each of which contains150 data sets distributed uniformly at random distribution points. This is called Data Set 2 and as shown in Figure 4.5.

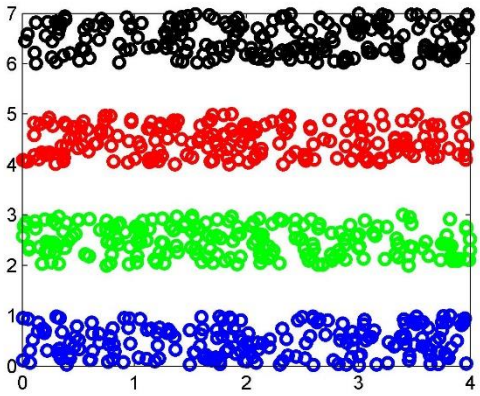


Figure 4.5 Plotting of Data Set 2. Points with same color are the same cluster

Data sets 3 and 4 are more complex. Both have a higher dimensionality, 7, and are created by mixing several Gaussian clouds. Each cloud is considered as one cluster. Table 4.3 shows the parameters of each cloud. The Data Set 4 is even more interesting and difficult, since the first center could be a bit confusing. This center has almost the same coordinates with the third center in Dimension 1, 2, and 4; has a similar coordinate with the other 2 centers in Dimension 5. In other words, the plotting of Data Set 4 in term of any three dimensions shows only 2 or 3 clusters seemingly because of overlapping.

Table 4.3 Parameters used for Data Set 3 and Data Set 4.

	Number of clusters, C	Dimen-sionality	Number of points	Mean of cluster centers	Standard deviation
Data Set 3	3	7	60	$(-1,-1,\ 2,-3,-4,-1,\ 3)$	2I
			60	$(-5,\ 2,\ 4,-1,-3,\ 5,-3)$	2I
			120	$(-1,-5,-4,\ 1,\ 3,-4,-5)$	4I
Data Set 4	4	7	60	$(\ 5,-2,-2,\ 0,\ 4,-2,\ 1)$	2I
			60	$(-5,\ 2,\ 2,\ 5,\ 4,\ 2,-3)$	2I
			60	$(\ 4,-2,\ 2,\ 0,-2,\ 0,\ 4)$	2I
			90	$(-4,\ 4,\ 5,-5,\ 5,\ 2,-5)$	3I

4.2.2 Result Measurement

To assess how well the different approaches perform, the value of objective functions is a good measurement. However, the objective function values shown in Table 4.2 cannot exhibit the clustering results very clearly. Since all data sets are synthesized, I chose the Rand Index to measure the similarity between the original synthesized clusters and new clusters obtained by clustering algorithms [19].

The Rand Index can be described as first presented in [20]. Given a set of n objects, $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, and two partitions to compare, $\mathbf{A} = \{A_1, A_2, ..., A_r\}$ and $\mathbf{B} = \{B_1, B_2, ..., B_s\}$, where **A** is a partition of **X** separated into r subsets and **B** is another partition of **X** separated into s subsets. Define:

a is the number of objects in **X** that are in the same set in **A** and in the same set in **B**.

b is the number of objects in **X** that are in different sets in **A** and in different sets in **B**.

c is the number of objects in **X** that are in different sets in **A** and in the same set in **B**.

d is the number of objects in **X** that are in the same set in **A** and in different sets in **B**.

The Rand Index, R, is:

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}} \tag{4.9}$$

where $a+b$ is the number of agreements between **A** and **B**, and $c+d$ is the number of disagreements between **A** and **B**. The value of RI varies from 0 to 1, with 0 showing that the two partitions do not agree on any pair of objects and 1 shows that the clusters in **A** and **B** are completely the same.

4.3 Implementation of Evolving Clustering

4.3.1 A Basic Approach

To apply EAs, some terms must be established. The objective function is the most important term in EAs, and fortunately FCM already has its own objective function, J_{FCM}, and is described in Eq. (4.3). Hence, this function can be used directly and optimized as objective function by EAs.

Second, the construction of individuals should also be defined. Intuitively, considering the objective function, there are two simple ways to define the construction: one method is to convert the cluster centers and the other method is to convert the membership to

serve as the chromosomes. However, some considerations are important while constructing the chromosomes in this case. If the chromosomes have too many dimensions, generally, the optimization process becomes harder. Also, the coupling among all memberships is strong. For example, assume some points are very close to each other. Once the memberships of one of these points are changed, all other point memberships can be relevantly modified. However, the probability of all these points evolving toward the same direction is very low, which means the probability of success will be low. Hence, in this thesis, I chose to convert cluster centers to be the chromosomes for the evolution process; furthermore, the i-th solution in the population at Generation g is defined as:

$$\text{solution}_i^{(g)} = \{\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_C\} \tag{4.10}$$

where $\mathbf{v}_j$ is a vector that describes the j-th cluster's center, and C is the number of clusters. A summary of the steps needed to implement EAs for optimization of clustering approaches follows.

Step 1. Randomly generate μ initial solutions, using the structure mentioned in Eq. (4.11) uniformly at random distribution points within the search space, and put them in the current population.

Step 2. Uniformly at random choose 2λ solutions from the current population with a replacement, so that some solutions may be chosen more than once. Pair each of the

two chosen solutions in order and each pair parent solution will apply a uniform crossover to generate a child solution. Totally λ solutions will be generated by λ pairs of parents at each generation.

Step 3. For every new generated solution, its every dimension has a mutation rate, P_{mutate}, to mutate by adding a Gaussian random variable with zero mean and standard deviation σ, i.e., $N(0, \sigma)$. These λ solutions are then added into current population.

Step 4. Compute the fitness of every new generated solution, i.e., value of objective function J_{FCM}. Select only the best μ solutions in terms of fitness from all current population ($\mu + \lambda$ solutions), and generate a new population.

Step 5. Repeat Step 2 to Step 4, until the maximum generation, G_{max}, is achieved or the fitness of the best solution in population has not changed for t generations; t will be called waiting generations later in this thesis.

Fortunately, it is not necessary to solve the objective function. Just knowing how to compute the objective function is enough for EAs. Updating equations as shown in Eqs. (4.3) to (4.8) are for the sake of illustration and is not required.

4.3.2 Experiments and Results

The five-step approach to optimizing clustering delineated in 4.3.2 is very fundamental. To see how well it performs, it was independently performed 100 times for clustering in each one of the four data sets and results were compared with the results obtained by traditional FCM described in Section 4.1. Table 4.4 shows the statistical results, where the number of clusters, C, is equal to the same number that is already known for each data set. The other parameters for the evolving clustering approach are as follows:

Population size: $N = (\mu + \lambda) = (30 + 30)$

Mutation rate: $P_{mutate} = 0.1$

Mutation step length: $\sigma = 0.1R_{max}$

Waiting generation, $t = 10$

Maximum generation: $G_{max} = 500$.

Note that R_{max} is the maximum value among every dimension's longest distance between the minimum value and maximum value over the data set.

In Table 4.4, it is very clear that basic evolving FCM, in general, can obtain the results with higher RI values for all data sets compared with traditional FCM, which is expected. This is what we expected based on our knowledge of EA benefits, especially the benefit of being more likely to find global optimal solutions.

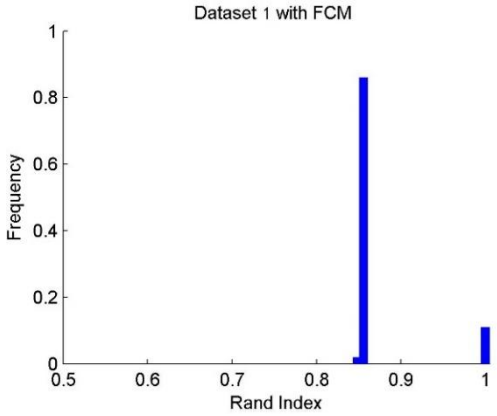
Table 4.4 Statistical clustering result optimized by both traditional FCM and evolving FCM

		Average runtime (ms)	Average Rand Index	Rand Index variance	Max Rand Index	Min Rand Index
Data Set 1	Traditional FCM	8.3	0.868	0.049	1.000	0.848
	Evolving FCM	576	0.963	0.064	1.000	0.848
Data Set 2	Traditional FCM	63.5	0.940	0.005	0.943	0.927
	Evolving FCM	1831	0.941	0.011	0.954	0.895
Data Set 3	Traditional FCM	39.1	0.785	0.081	0.925	0.738
	Evolving FCM	1460	0.821	0.106	0.936	0.723
Data Set 4	Traditional FCM	19.2	0.934	0.046	0.970	0.851
	Evolving FCM	1680	0.941	0.049	0.981	0.834

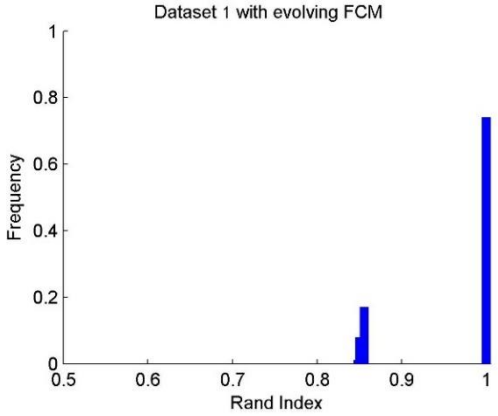
However, it should be noted that the variance of Rand Index obtained by evolving FCM was also higher than the alternative optimized traditional FCM; furthermore, the maximum Rand Index obtained by evolving FCM is bigger than the traditional FCM and minimum Rand Index obtained by evolving FCM is smaller. That is, the evolving FCM may find a few more solutions than traditional FCM.

For a more detailed and straightforward investigation of the Rand Index value, Figure 4.4 shows the statistical results, the frequency of every Rand Index value out of the 100

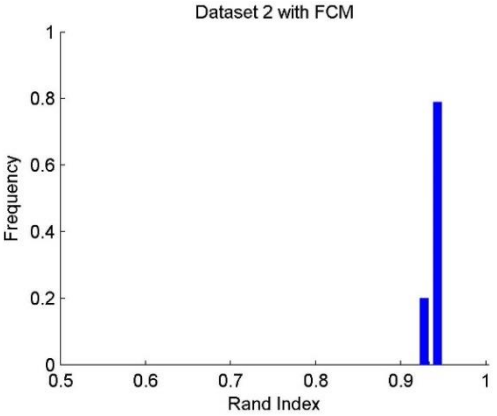
trials for each experiment respectively, where the x-axis represents the value of Rand Index obtained and y-axis represents the frequency of that Rand Index value.



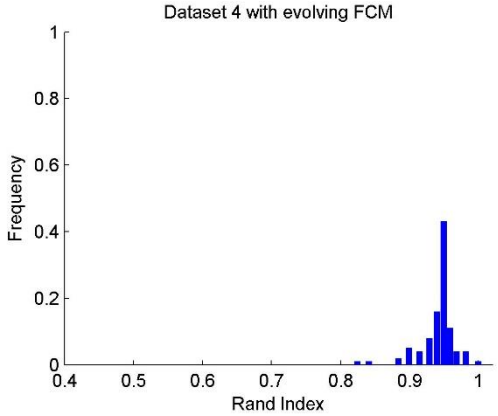
(a1) Traditional FCM with Data Set 1



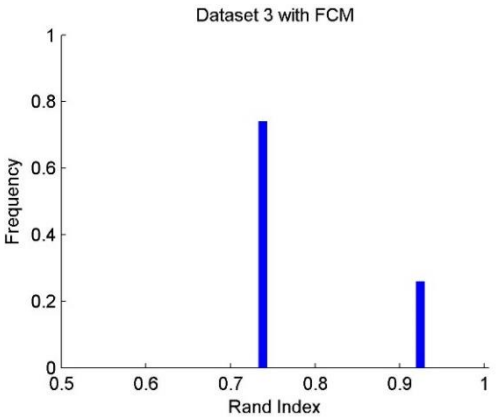
(b1) Evolving FCM with Data Set 1



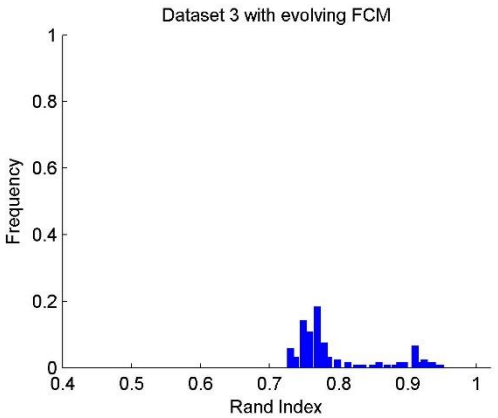
(a2) Traditional FCM with Data Set 2



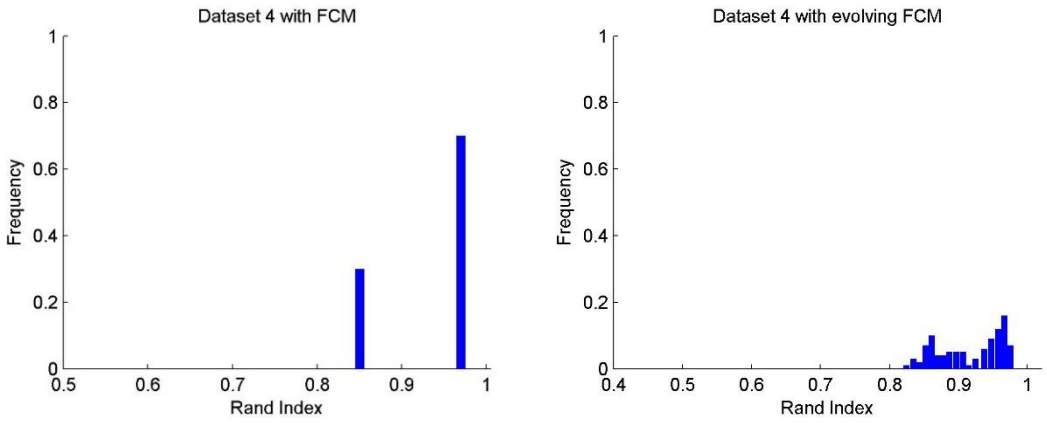
(b2) Evolving FCM with Data Set 2



(a3) Traditional FCM with Data Set 3



(b3) Evolving FCM with Data Set 3



(a4) Traditional FCM with Data Set 4 (b4) Evolving FCM with Data Set 4

Figure 4.4 The Rand Index distribution for each experiment.

First, from Figure 4.4 a1 – a4, it is obvious that the FCM can find only a few possible optimal solutions, three for Data Set 1 and two for each of the other data sets. However, the results obtained by evolving FCM are more scattered. Intuitively, it is easier to accept that there are only a few optimal solutions, including both local and global optimums, but not as many as can be found by evolving FCM.

Take a closer look at the results obtained by evolving FCM, b1 – b4. It is very interesting that the distribution of Rand Index looks like a mixture of some Gaussian distributions. For example, comparing a3 and b3, solutions found by FCM had two different RI values, and values of RI obtained by EFCM were distributed around those two values. Considering the character of EAs, the answer is very obvious. The seemingly abundant solutions are not the real local optimal solutions, but are the less developed optimized solutions. This is because the mechanisms of FCM and EAs are not the same, even though both of them stop when the solutions cannot be optimized any more. FCM uses the alternatively updating methods; hence, each step finds a better

solution until the solution reaches the sink mode of an optimal position and no better solution can be found. However, EAs use the heuristic way to find better solutions. That is, the new generated solutions in each generation cannot be guaranteed to be better than their parent solutions. This problem becomes much more serious near the end of the evolving process. Therefore, it is very possible that a population of solutions cannot find a better one in generations, even though the currently found best solutions are not well optimized, that is, they are close to the optimal position but are not there yet.

To prove that the solutions obtained by evolving FCM are not the newly found optimums but are only the solutions which are not well optimized, is simple. If an obtained solution has less difference than another solution when considering the sum of the squared Euclidean distance of all clusters, both solutions can be considered similar. After putting all similar solutions into the same group, several groups of solutions will form. The solutions in different groups should be very different, and the solutions in the same group should be considered as one. The difference between two obtained solutions is the sum of the squared Euclidean distance, $s = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_C\}$ and $s' = \{\mathbf{v}'_1, \mathbf{v}'_2, \ldots, \mathbf{v}'_C\}$, which is derived as:

$$d(s, s') = \sum_{j=1}^{C} \left\| \mathbf{v}_j - \mathbf{v}'_j \right\|^2 \tag{4.11}$$

where C is the number of clusters set, $\mathbf{v}_j$ is the j-th cluster's center obtained by solution s and $\mathbf{v}'_j$ is the j-th cluster's center obtained by solution s', and the order of

clusters are modified. The modification is in case the different ordered but same cluster centers get a large difference instead of zero. Table 4.5 shows the grouped information for both FCM and evolving FCM, which used the same results obtained in Table 4.4.

Table 4.5 Groups of solutions obtained for each case

	Group ID	FCM		Evolving FCM	
		Frequency number	Rand Index	Frequency number	Mean Rand Index
Data Set 1	Group 1	8	1.000	75	1.000
	Group 2	89	0.856	17	0.854
	Group 3	3	0.848	8	0.849
Data Set 2	Group 1	83	0.943	86	0.944
	Group 2	17	0.927	14	0.923
Data Set 3	Group 1	25	0.925	42	0.917
	Group 2	34	0.738	20	0.743
	Group 3	41	0.738	38	0.764
Data Set 4	Group 1	71	0.970	72	0.961
	Group 2	29	0.851	30	0.853

It should be clear that a group of solutions are solutions obtained from different trials of experiments but their clustered centers are very similar. Each row in Table 4.5 represents a group of solutions, which are similar to each other, whether they come from FCM or EFCM.

Finally, there are several groups of optimized results for each data set as expected. More precisely, the solutions in the same group obtained by FCM are exactly the same, and those solutions obtained by evolving FCM are very similar to each other. This means the previous statement that solutions obtained by evolving FCM are not well optimized

but are only close to the optimum positions, is correct. Meanwhile, it is encouraging to see that the frequency of solutions belonging to a better group obtained by an evolving FCM is higher than the traditional FCM. This shows the benefit of EAs as being good at overcoming the local optimum solutions during optimization, which confirmed our expectations.

However, two problems remain. First, the solutions found by evolving FCM are unstable. They cannot always be guaranteed to be the most optimal ones, whether they are formed locally or globally. They usually just stop close to the optimal positions, and cause a big variance of results as shown in Figure 4.5 b1 – b5. This is a serious problem, which reduces the performance of evolving FCM. Besides, the time consuming process of evolving FCM is many dozens of times greater than that of traditional FCM as shown in Table 4.4. Both the variance in results and time disadvantage when compared to traditional FCM lower the evolving FCM's cost efficiency.

CHAPTER 5. IMPROVED INTERNAL EAS

The previous chapter presented two problems of evolving FCM: 1) The solutions may not be well optimized and stop just short of their optimal points, and 2) completion time is too long. Actually, both problems can be addressed together. The problem is that solutions cannot reach the optimal position because during the end of the evolving process, it's hard to generate better solutions while comparing current ones. The lengthy time it takes to complete the evolution process is also caused by too few good solutions generated during each generation, which delays the speed of evolving. Both problems can be solved by increasing the success rate by generating a higher quality of offspring every generation.

One of the best ways is to give the most suitable set of parameters for each data set. However, considering the character of EAs, we cannot guarantee that a time reduction in the evolving process is caused by either the improvement of tuning a parameter correctly or a failed tuning so that it is hard for solutions to find better ones. Hence, they stopped early even though they were much further away from any optimal position, especially when there was no prior knowledge of the data set condition. To overcome this dilemma, I proposed a multi-level strategy to solve the problem, which will be introduced in this chapter.

5.1 Multi-level strategy

Throughout the whole process of improving the evolving FCM, there were only two parameters that concerned me, the mutation rate and the mutation step length. Their values have a great effect on the quality of the mutation process. Either too many or too few of them produce bad solutions, and for different data sets, the fittest values are unknown and differ, even within the same data set. The fittest parameters are also changing during the evolving process. It was hard to set the proper values directly, unless we conducted some experiments to study them. Hence, I proposed this multi-level strategy, which assigned a parameter to a few different values at the same time, to try to overcome the problem.

This strategy can be described as follows. In each generation, there will be some offspring generated by crossover. When they are mutating, some of them use one set of parameters and some use a different set, and some use another set, etc. This strategy has three benefits.

First, it is more likely that a set of parameters will be found that is better than the one set of parameters serving as a bottleneck now due to their restrictions. Choosing different parameters is determined by experience or even randomly, such as that demonstrated in Section 4.3.2. Second, the operator is able to recognize that some parameters need different values during the whole evolving process, such as the

mutation step length σ. Third, the division of labor model is always better and more effective than one handling all. The following sections will show how this strategy works.

5.2 Multi-rate Mutation

The first attempt to improving the evolving FCM is to focus on the mutation rate P_{mutate}. Usually, this rate is set at a small value, like 0.1. However, it's not good enough to set P_{mutate} so it will always equal a certain value. Recall from Chapter 3, if the mutation rate is either too high or too low, the evolving speed is reduced. But the standard of high or low is not same for all data sets, or more apparently, for the different dimensionalities of the search space. For example, for a 2-dimensional optimization problem, the mutation rate of 0.1 seems to be too low, since each child will have up to $0.9 \times 0.9 = 0.81$ chance of not being changed (0.9 chance for each dimension of this child). But this may be too high for a 1000 dimensional optimization problem, since the expectation number of changed dimensions of a child is 100. The more the dimensions are changed, the harder it is for that child mutate to a better one. Therefore, it will be good to apply the multi-level strategy for mutation rates. The new modified mutation process can be called the multi-rate mutation. The details of this multi-rate mutation are described as follows.

Given a set of λ children solutions, which are already generated by crossover and waiting for mutation, and a pre-set value P_{max}, which indicates that the maximal mutation rate will be used, mutation will start from the 1st to the λ^{th} solution within the set. For the i-th solution, its component in each dimension has a probability $P_i = (i/\lambda) \times P_{max}$ to mutate by adding a Gaussian random variable with zero mean and a standard deviation σ, where σ is a pre-set mutation step length.

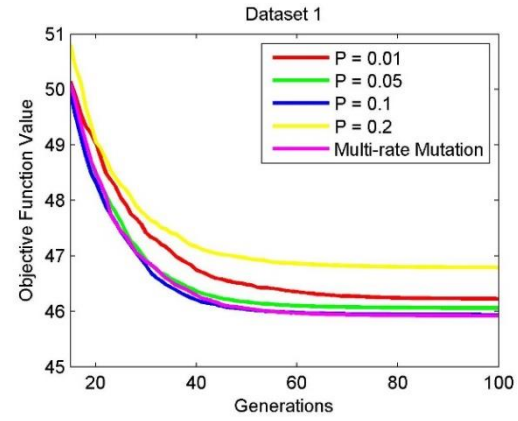
Compare the results obtained by applying this multi-rate mutation to the previous normal mutation (described in Section 4.3.2) with a certain rate. These results are averaged among 100 independent trials for each case. Figure 5.1 (a–d) shows the average value of the objective function, J_{FCM}, at each generation (extracted from Generation 15 to Generation 100 for a better view), where each curve represents a case as the legend says, for the four data sets. All other parameters are the same and include population size:

$N = (\mu + \lambda) = (30 + 30)$;

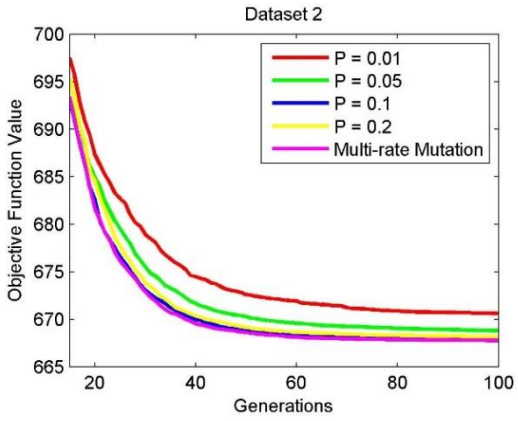
Mutation step length: $\sigma = 0.1R_{max}$;

Waiting generation: $t = 10$;

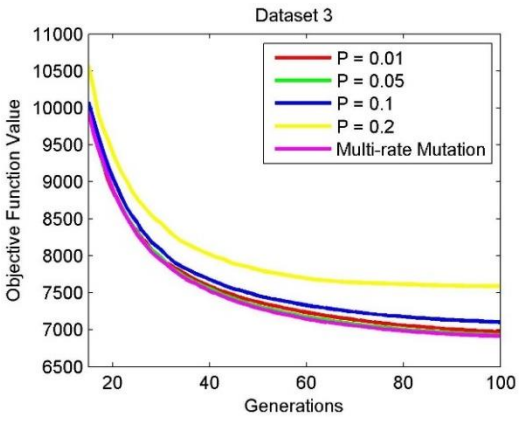
Maximum generation: $G_{max} = 500$.



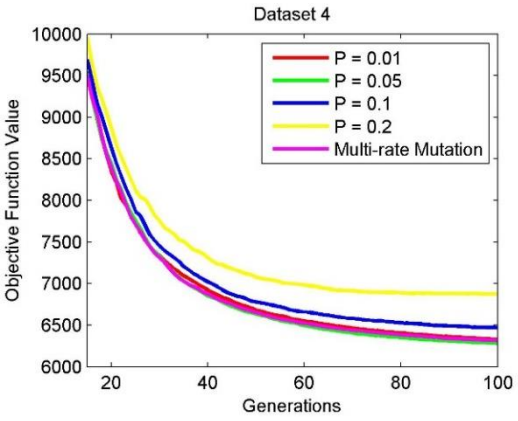
(a) Result used Data Set 1



(b) Result used Data Set 2



(c) Result used Data Set 3



(d) Result used Data Set 4

Figure 5.1 Comparison of multi-rate mutation with singe rate mutation.

Four selected mutation rates, {0.01, 0.05, 0.1, and 0.2}, were applied to each data set. It can be seen from the figures above, choosing either a high mutation rate, like 0.2 in (a), (c) and (d), or a low rate, like 0.01 in (a) and (b), is suffering a risk that the evolving process may result in a very bad solution. For the remaining two rates, rate 0.1, represented by the blue curve, seems to be the best choice in Data Set 1 and Data Set 2, but it cannot guarantee to find the best solutions in Data Set 3 and Data Set 4. However, the mutation rate of 0.05 is the opposite; it got the best performance in Data Set 3 and Data Set 4, but not in Data Set 1 and Data Set 2.

However, the multi-rate mutation solved this problem as expected. It performed best in Data Set 2 and Data Set 3, and was very close to having the best curves in the other two even though they were not the best. In other words, the multi-rate mutation can be seen as successful in tuning the mutation rate.

5.3 Mutation Step Length

The choice of mutation step length, that is, σ, is even more critical than the mutation rate. As mentioned in Chapter 3, the smaller mutation step makes the evolving process more like a gradient descent approach, so that the solution cannot jump out of the local optima, and a larger mutation step makes the population harder to get to an effective (better) solution during the end of the evolving process, which makes the solutions stop far away from optimal positions. More accurately, the EAs need a larger mutation step to enable a wider exploration and freedom from the local optimal areas. Meanwhile, a much smaller step is also needed during the end of evolving process to provide solutions a better opportunity and more speed to get to the real optimal point from nearby places.

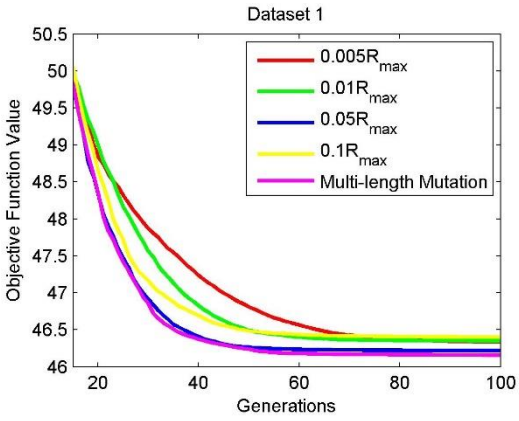
The mutation step length is another good candidate to apply the multi-level strategy to, getting a multi-length mutation. The details of assigning the multiple mutation step lengths are described as:

I. For the i-th dimension of the data set, find the longest distance between the maximum value and minimum value in this dimension, which is recorded as $dist_i$.

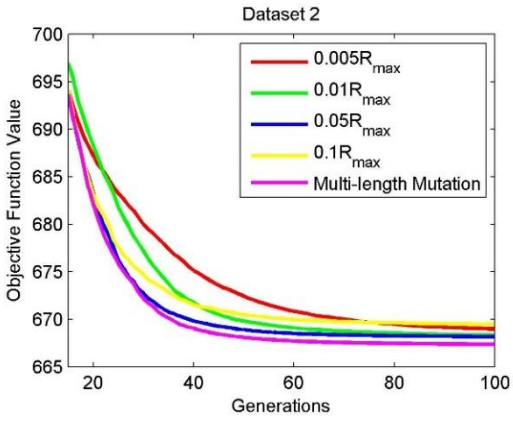
II. Find the maximal distance among $dist_i$ and denote it as R_{max}, where i is from 1 to number of dimensions of the data set.

III. At each generation, the i-th generated solution by crossover, uses a certain length step $\sigma_i = (i/\lambda) \times CR_{max}$ to mutate, for i is equal to 1 to λ, where λ is the number of offspring generated at each generation, and C is a constant value that is less than 1.

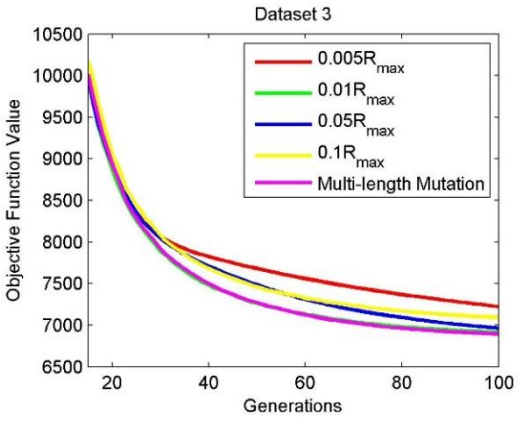
To evaluate how this multi-length mutation approach works with $C = 0.1$, we compared with the previous method described in Section 4.3.1, with certain mutation steps: $0.005R_{max}$, $0.01R_{max}$, $0.05R_{max}$, and $0.1R_{max}$, respectively. The results below are averaged among 100 independent trials for each case. Figure 5.2 shows the average value of the objective function at each generation, where each curve represents a case as the legend says, for the four data sets, respectively. All other parameters are the same as first shown in Section 4.3.2 as follows: population size: $N = (\mu + \lambda) = (30 + 30)$; mutation rate: $P_{mutate} = 0.1$; waiting generation: t = 10, and maximum generation: $G_{max} = 500$.



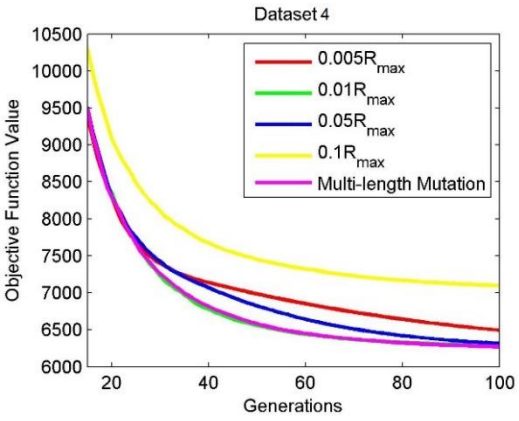
(a) Results used Data Set 1



(b) Results used Data Set 2



(c) Results used Data Set 3



(d) Results used Data Set 4

Figure 5.2 The average value of objective function at each generation for each case.

As expected again, the purple curves in all four figures, which represent the multi-length mutation proposed, have the fastest descending speed in Data Set 1 and Data Set 2, and almost the fastest in Data Set 3 and Data Set 4. Even more, this time the best solutions (with lowest objective function values) obtained for the four data sets were all found by multi-length mutation.

For the other four selected mutation lengths, they also give similar outcomes with the mutation rate. Either a large value or a small value will result in a very bad solution.. It

is $\sigma = 0.05R_{max}$ that makes evolving process faster on Data Set 1 and 2, but it turns to be $\sigma = 0.01R_{max}$ on Data Set 3 and 4. This increases the difficulty of setting the value of σ. Hence, applying multi-length mutation is also better than setting a single value for mutation length depending on experience or randomness.

5.4 Combination and Comparison

Two independent improvements, which focused on tuning the parameters of mutation, have been proposed in this thesis and both of them worked well as expected. However, only one parameter applied the multi-level strategy at one time in previous experiments, and the other parameter was set just by experience, i.e. not the fittest value. The challenge here was to find out if both parameters can use the strategy to form a multi-rate, multi-length mutation and provide an even better performance.

In this section, I ran a simple experiment to test the multi-rate, multi-length mutation approach, which can be described as follows. Assume the offspring size is set as equal before, say $\lambda = 30$; then, each new generated child solution will apply a different set of parameters. That is, at any Generation g, there will be $\lambda = 30$ children solutions generated by crossover. Label all these solutions from 1 to 30, and each solution will apply its corresponding parameters as shown in Table 5.1.

Table 5.1 Parameters used for every new generated child solution at each generation

P_{mutate} \ σ	$0.005R_{max}$	$0.010R_{max}$	$0.020R_{max}$	$0.050R_{max}$	$0.075R_{max}$	$0.100R_{max}$
0.050	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
0.075	Set 7	Set 8	Set 9	Set 10	Set 11	Set 12
0.100	Set 13	Set 14	Set 15	Set 16	Set 17	Set 18
0.125	Set 19	Set 20	Set 21	Set 22	Set 23	Set 24
0.150	Set 25	Set 26	Set 27	Set 28	Set 29	Set 30

For example, the 10^{th} child solution at any generation will use the 10^{th} parameter set, i.e., Set 10 in Table 5.1, which gives a mutation rate of $P_{mutate} = 0.075$ and mutation step length of $\sigma = 0.050R_{max}$.

Consider all four mutation approaches mentioned: (a) a normal mutation with a single mutation rate and a single mutation step length in Section 4.3.2, (b) the multi-rate mutation (MRM) in Section 5.2, (c) the multi-length mutation (MLM) in Section 5.3, and (d) the multi-rate multi-length mutation (MMM) described above. One hundred independent experiments for each case were executed and the averaged results in terms of Rand Index and runtime are shown in Table 5.2. All other parameters are the same as presented in Section 4.3.2 and include population size: N = (μ+ λ) = (30 +30); waiting generation: t = 10, and maximum generation: G_{max}= 500.

Generally, applying the multi-level strategy on either mutation rate or mutation step length is getting a better performance but uses less time compared with the basic EFCM approach. For Data Set 1 and Data Set 2, the Rand Index gets only a small increase

from the single mutation rate when compared to the MRM; but there is a bigger increase from single mutation length when compared to MLM. However, for Data Set 4 the opposite happens as the MRM get a bigger increase. For Data Set 3, applying the multi-level strategy on either parameter seems to get a similar increase. In addition, either MRM or MLM can be considered faster than the normal basic EFCM, since they all need less runtime and do not lose their clustering quality (Rand Index), but instead perform even better.

Table 5.2 Comparison of the four mutation methods

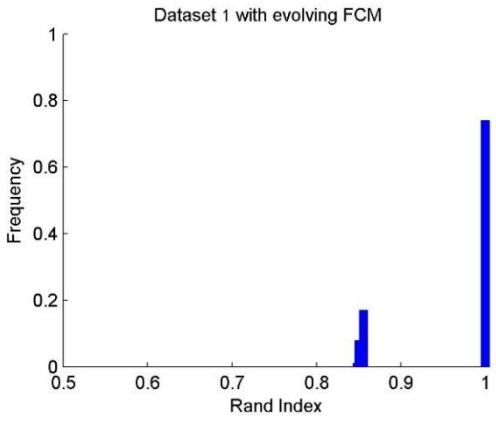
		Basic EFCM	EFCM with MLM	EFCM with MRM	EFCM with MMM
Data Set 1	Rand Index	0.963	0.964	0.970	0.972
	Runtime(ms)	576	414	380	338
Data Set 2	Rand Index	0.941	0.945	0.961	0.967
	Runtime(ms)	1831	1390	1410	1290
Data Set 3	Rand Index	0.821	0.854	0.855	0.866
	Runtime(ms)	1460	1137	1400	1510
Data Set 4	Rand Index	0.941	0.944	0.942	0.945
	Runtime(ms)	1680	1342	1430	1250

Another major achievement of this research was the finding that the new MMM approach has the best clustering results in term of Rand Index compared with all three of the other mutation approaches for all four datasets. However, there is one tiny flaw

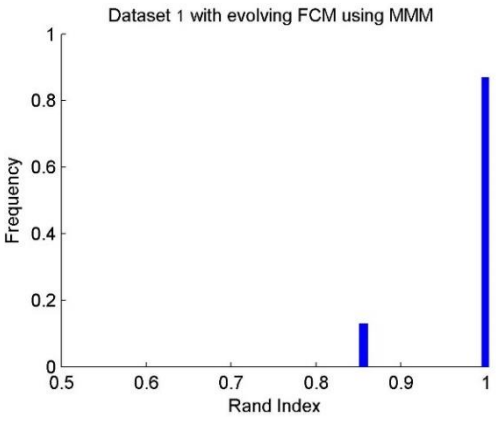
in Data Set 3 concerning the runtime. The runtime needed for all three of the other data sets are the least, but it is the largest in Data Set 3. In other words, the evolving speed is slower than even basic EFCM.

A comparable analogy allows us to view this from another angle: There is a person who needs 10 seconds to run 50 meters, and another person who ran 100 meters in 11 seconds. It is obvious that the second person's speed is faster, since he ran much longer. Hence, I think the unexpected length in time is both minimal and acceptable, and the newly developed MMM is the best choice for the multi-level strategy.

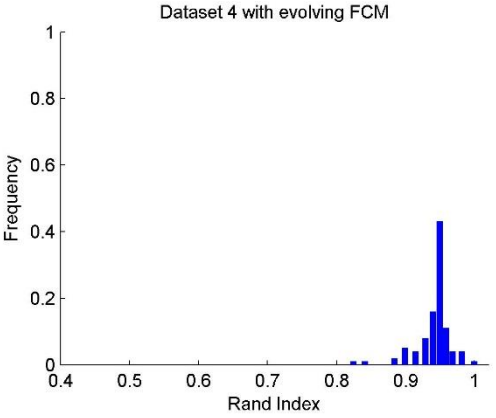
To gain a more straightforward view of how much better the MMM method is for FCM development, another statistical comparison between the earliest basic EFCM and current MMM was completed. Figure 5.3 shows the frequency of every Rand Index value obtained. Each figure shows a result of a different case that represents statistics from 100 independent trials, where a1 – a4 yielded the same results, obtained by the basic EFCM, as shown in Figure 4.5 for Data Sets 1 to 4, respectively, and b1–b4 are the results obtained by MMM for each data set, respectively. The x-axis represents the value of Rand Index obtained and the y-axis represents the frequency of that Rand Index value.



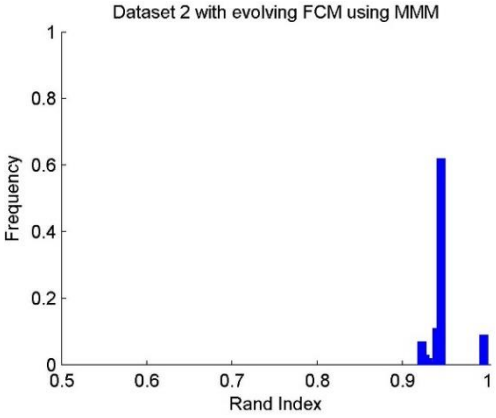
(a1) Basic evolving FCM with Data Set 1



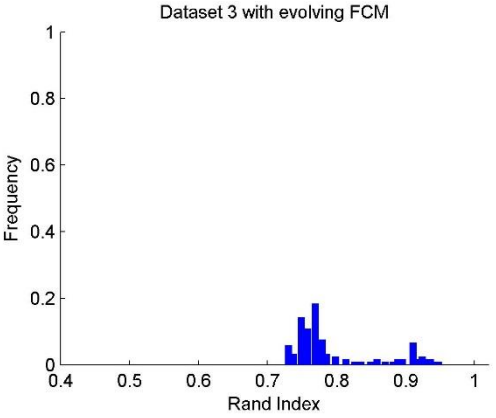
(b1) Evolving FCM using MMM with Data Set 1



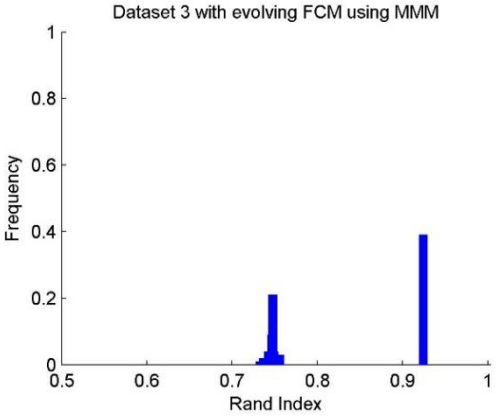
(a2) Basic evolving FCM with Data Set 2



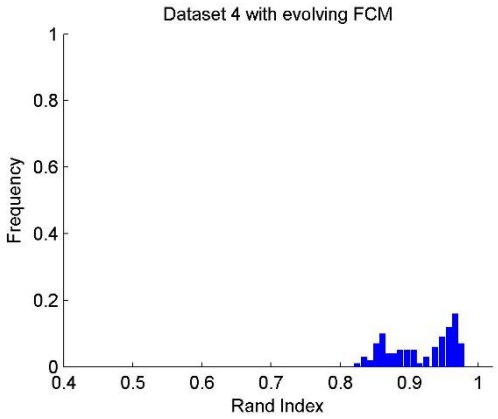
(b2) Evolving FCM using MMM with Data Set 2



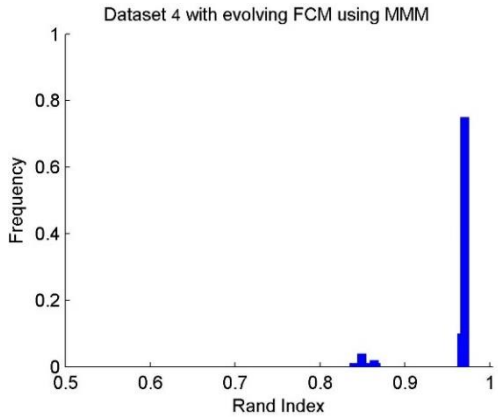
(a3) Basic evolving FCM with Data Set 3



(b3) Evolving FCM using MMM with Data Set 3



(a4) Basic evolving FCM with Data Set 4



(b4) Evolving FCM using MMM with Data Set 4

Figure 5.3 The Rand Index distribution for each experiment.

From the figures above for all four data sets, the MMM based EFCM has three main improvements compared with the basic EFCM. First, many scattered distributed solutions are converged together, which means similar solutions were obtained by the basic evolving FCM, which were considered as the ones that were not well optimized above. They became the same, which is very obvious in Data Sets 2 – 4. Second, the ratio of better solutions, in term of Rand Index, was further increased, which mainly caused the increase of the average performance (Rand Index) in Table 5.2. Third, it is very interesting that in Data Set 2, neither normal FCM nor the earliest evolving FCM found a solution, but their clustered centers achieved a Rand Index value equal to 1, which was found by the evolving process as it applied this multi-rate multi-length mutation.

In summary, the multi-level strategy can overcome the dilemma of setting the values of the mutation rate and mutation step length for the EFCM. The EFCM applied this strategy on only the mutation rate, mutation step length, or both and still achieved a

much better performance and runtime than when using some single valued parameters, especially since we did not know which values were the best in advance. However, there is still a problem in that sometimes, the solutions obtained by EFCM with MMM are still not well optimized, even though the situation is much better now. If these solutions are also well optimized and accepted as the real optimal solutions, the overall performance should be better.

CHAPTER 6. HYBRID IMPROVEMENT

The multi-level strategy proposed in Chapter 5 works well for improving evolving performance in terms of both results and runtime. However, the obtained solutions are still sometimes not well optimized, which are the scattered solutions distributed around the real optimal position. Considering the powerful local searching feature of the traditional FCM, this chapter continues to present the process of improving the evolving FCM performance.

6.1 Improved EFCM

Due to the problem we now have, where sometimes the final obtained solutions may not be well optimized to the optimal solutions, no matter what the local optimum or global optimum is, a simple consideration is to apply the traditional FCM to finally optimize the solution. This approach can be called improved EFCM. Recalling the evolving FCM described in Section 4.3.1 and the traditional FCM in Section 4.1.1, this new approach can be described as follows:

Step 1. (Initialize EFCM) Randomly generate μ initial solutions, using the structure shown in (4.11), $\text{solution}_i^{(g)} = \{\mathbf{v_1}, \mathbf{v_2}, \ldots \mathbf{v_C}\}$ where $\mathbf{v}_j$ is a vector that describes the j-th cluster's center, and C is the number of clusters. Then μ initial solutions are uniformly distributed at your discretion within the search space, and put in the current population.

Step 2. (Crossover) Uniformly random select 2λ solutions from current population with replacement, so that some solutions may be chosen more than once. Pair each of the two chosen solutions in order and apply uniform crossover to generate λ solutions as offspring.

Step 3. (MM mutation) For every new generated solution, each dimension has a mutation rate, P_{mutate} to mutate by adding a Gaussian random variable with a zero mean and standard deviation σ, i.e., N(0,σ). The parameter set applied the multi-rate multi-length mutation strategy, as shown in Table 5.1. These λ solutions are then added into the current population.

Step 4. (Selection) Compute the fitness of every new generated solution, i.e., the value of objective function J_{FCM}. Select only the best μ solutions, in terms of fitness, from all current population (μ+ λ solutions), and generate a new population.

Step 5. (Stop checking for Evolving FCM) Repeat Steps 2 to 4, until the maximum generation, G_{max}, is achieved or the fitness of the best solution in the population has not changed for t generations, where t is the number of waiting generations.

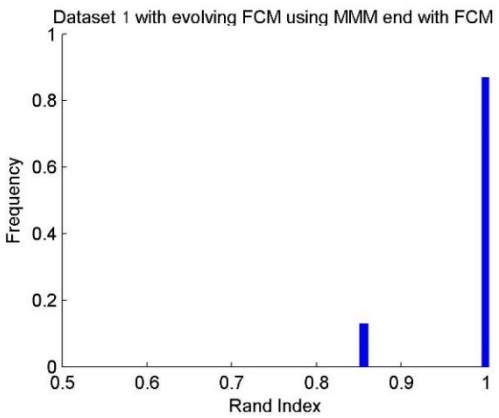
Step 6. (Decode) the best solution obtained by evolving FCM, and decode it into C cluster centers.

Step 7. (Update partition matrix U) Update every membership u_{ji} using cluster centers by applying Eq. (4.7).

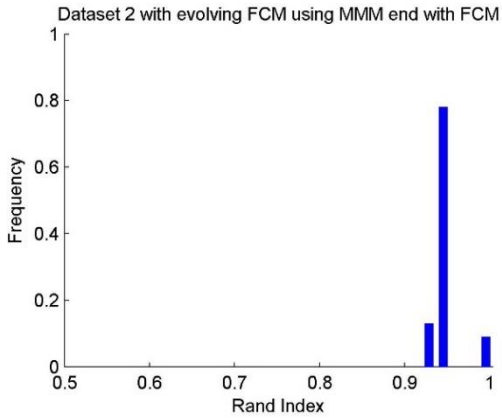
Step 8. (Update cluster centers V) Update every cluster center $\mathbf{v}_j$ using partition matrix by applying Eq. (4.8).

Step 9. (Finally stop checking) Repeat Step 7 and Step 8, until the change of J_{FCM}, ΔJ_{FCM}, is less than a preset convergence threshold, ε, where ε is set at 10^{-8} for every experiment in this thesis.

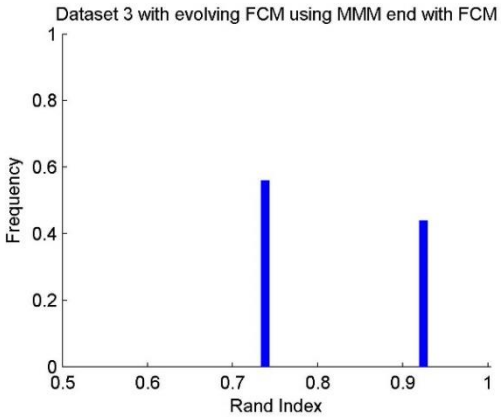
Reapplying this approach on all four datasets and Figure 6.1 shows another new Rand Index frequency distribution, extracted from 100 independent experiments for each data set. Obviously, all four figures are very clear now. There is no longer any scattered solution. By comparing these figures with the previous ones in Figure 5.3, which applied MMM evolving FCM only, the ratio of each optimal solution obtained for each data set is almost unchanged. In other words, it is mainly the traditional FCM that drags those scattered solutions toward their real optimal positions at the end of the whole clustering process as expected.



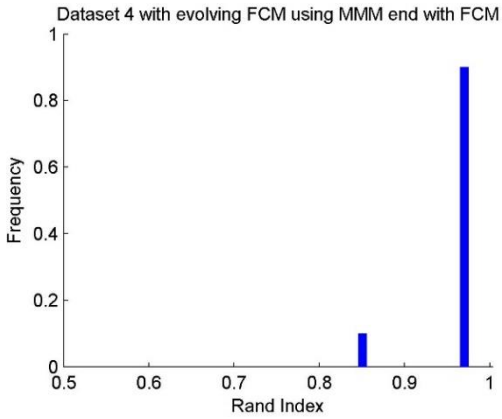
(a) Results used Data Set 1



(b) Results used Data Set 2



(c) Results used Data Set 3



(d) Results used Data Set 4

Figure 6.1 The Rand Index distribution for each experiment.

6.2 Improved GFCM

Previous researchers have proposed other similar ways to combine the traditional FCM and evolving FCM, referring to them as improved genetic FCM (improved GFCM) [21–24]. They used FCM to optimize every new child solution generated by crossover and mutation at each generation. Because of the powerful local searching ability, these newly generated solutions are all optimal solutions, whether locally or globally designated. This greatly simplifies the evolving process of EAs, since they only need to overcome local optimal problems. Meanwhile, due to the fast optimization speed, the whole clustering process will speed up by reducing the time consumption of local search processes. The process of this improved GFCM can be described as below:

Step 1. Randomly initialize μ initial solutions, using use the structure mentioned in (4.11), and uniformly distribute within the search space, putting them in current population.

Step 2. Apply the crossover by using the solutions in the current population to generate λ solutions.

Step 3. Apply the mutation on each new solution (totally λ solutions) generated from above.

Step 4. Optimize the λ mutated solutions above by applying FCM on each new solution, and put all optimized λ solutions into the current population.

Step 5. Compute the fitness of these λ new optimized solutions, i.e., value of objective function J_{FCM}. Select only the best μ solutions, in terms of fitness, from all current population ($\mu + \lambda$ solutions), and generate a new population.

Step 6. Repeat Steps 2–5 until the maximum generation, G_{max}, is achieved or the fitness of the best solution in the population has not changed for t generations, where t is the number of waiting generations.

Apply this approach 100 times independently for each data set, where the mutation process uses the multi-rate, multi-length mutation strategy, and the other parameters as shown below. The averaged results, in terms of Rand Index value and runtime are shown in Table 6.1 and are compared with the results obtained by improved EFCM in Section 6.1. The parameters which accompany the multi-rate, multi-length mutation strategy are:

Population size: $N = (\mu + \lambda) = (30 + 30)$

Waiting generation: $t = 10$

Maximum generation: $G_{max} = 500$

Stop threshold of FCM $\varepsilon = 10^{-8}$.

Table 6.1 Comparison of Mean Rand Index and runtime obtained by two hybrid approaches

		Data Set 1	Data Set 2	Data Set 3	Data Set 4
Mean Rand Index	Improved GFCM	1.000	0.943	0.925	0.970
	Improved EFCM	0.982	0.970	0.863	0.951
Mean Runtime(s)	Improved GFCM	3.85	22.13	12.60	11.24
	Improved EFCM	0.35	1.23	1.52	1.27

From the first look, this improved GFCM promised better results, in terms of Rand Index. What's more, the results obtained by this improved GFCM for each data set are the same. That is, every experiment of improved GFCM on Data Set 1 obtains the result with a Rand Index which is equal to 1. On Data Set 2, the Rand Index is always equal to 0.943, and so on for Data Set 3 and Data Set 4 respectively. Despite these good and stable results, other problems must be addressed here.

First, recall the results obtained by traditional FCM in Table 4.4. This improved GFCM obtained solutions, which are only as good as the best ones that the traditional FCM can obtain. In other words, if we run traditional FCM many times, say 100, and choose the

best solution among all 100 solutions as the final outcome, then there is no difference between this multiple FCM and that improved GFCM in terms of the outcome. Even after running 100 times, FCM needs much less time than the improved GFCM: 0.83 second is needed (average is 0.0083 second as shown in Table 4.4) by 100 times' FCM and 3.85 seconds are needed by improved GFCM for Data Set 1 as an example. In addition, the previous improved EFCM can even find a new optimal solution, which is actually the global one with a Rand Index equal to 1, but the improved GFCM did not show this advantage, which we are expected all evolving approaches to have. Hence, the conclusion is, this improved GFCM is only doing the same work like choosing a best result from abundant times' FCM, but without any obvious new advantage.

6.3 Modified GFCM

Taking another closer look at the improved GFCM, a big conflict was identified, which was responsible for the bad performance in the previously improved GFCM. In the previously improved GFCM, each new generated solution is optimized by FCM. If the population size is $(\mu + \lambda) = (30 + 30)$, there are 30 child solutions, which will be generated at each generation, and the FCM must be performed 30 times, once on each solution. Even if the evolving process stops at Generation 20, the FCM will perform $30 \times 20 = 600$ times. Considering the time consuming burden on FCM, as shown in Table 4.4, the slowly improved GFCM can be explained. Even worse, all newly

generated solutions are dragged right to the optimal positions. It is obvious that a solution which is located at the center of a basin is harder to get out of the basin, compared to a solution which is located at the edge. In other words, the FCM takes much more time to drag the new generated solutions towards the optimal position, which makes the solutions harder to jump out in search of other optimal positions. Hence, it takes a much longer time, but does not get a better performance.

The thought of introducing FCM to speed up the local search process did not encounter opposition. Only the FCM work may have been overdone. Hence, I proposed a new modified approach, which reduces the optimization pressure of FCM and is used to optimize the newly generated solutions. The idea is to set a very small maximum iteration count on FCM, so that the FCM can optimize the solution but only optimize it a little. This modification on FCM can be described as below:

Step 1. Randomly generate an initial set of cluster centers $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_C\}$. To keep the consistency with other approaches, I decided to initialize all initial cluster centers $\mathbf{v}_j$ uniformly at random within the domain of a data set.

Step 2. Update every membership u_{ji} by applying Eq. (4.7).

Step 3. Update every cluster center $\mathbf{v}_j$ by applying Eq. (4.8).

Step 4. Repeat Steps 2 and 3 **at most T_{max} times**, or at the change of J_{FCM}, ΔJ_{FCM}, is less than a preset convergence threshold, ε, where T_{max} is a very small number.

Replace the normal FCM with this modified FCM in previous improved GFCM to form the new modified GFCM. Perform 100 independent experiments on each data set respectively, with the parameters used as below. Tables 6.2 and 6.3 show the averaged results for each case in terms of the Rand Index and runtime, respectively, and compared with results obtained from previous improved GFCM (in Table 6.1).

Population size: $N = (\mu + \lambda) = (30 + 30)$

Waiting generation: $t = 10$

Maximum generation: $G_{max} = 500$

Maximum Iteration of FCM $T_{max} = \{1, 10, 20\}$

Stop threshold of FCM $\varepsilon = 10^{-8}$

Similar to the previous improved GFCM, the solutions obtained for each data set by this modified GFCM are all the same and as good as the best result that can be found by normal FCM. The reduction of maximum update iteration of FCM, which is applied on every newly generated solution, does not even reduce the quality of results obtained. The maximum iteration can be set at 1 or 10 or 20—no matter what setting it is at, this modified genetic FCM always gets the consistently good solutions. However, the time consuming factor of the modified GFCM is reduced to less than 10% of the improved GFCM when T_{max} is set to only 1, which means, only one iteration by FCM is needed to optimize the new solutions. The additional updating iterations had no any positive effects.

Table 6.2 Averaged Rand Index of the modified GFCM for the four data sets

Modified GFCM				Improved GFCM
T_{max}	1	10	20	
Data Set 1	1.000	1.000	1.000	1.000
Data Set 2	0.943	0.943	0.943	0.943
Data Set 3	0.925	0.925	0.925	0.925
Data Set 4	0.970	0.970	0.970	0.970

Table 6.3 Averaged runtime (ms) of the modified genetic FCM for the four data sets

Modified GFCM				Improved GFCM
T_{max}	1	10	20	
Data Set 1	263	423	557	3850
Data Set 2	1710	3340	5610	22130
Data Set 3	1120	1650	2270	12600
Data Set 4	960	2010	3440	11240

The time factor of running the modified GFCM one time is equal to running a normal FCM for Data Set 1 approximately 38 times. Table 4.4 shows the ratio for Data Set 2 and Data Set 3 to be about 1:28 times for the modified GFCM to the normal FCM, and for Data Set 4 1:50 times. Even if the normal FCM gets the same results by running multiple times, no one knows exactly how many times is needed to run the normal FCM so that it can guarantee securing the global optimal solutions. Considering Data Sets 1 and 3, see also Figure 4.5, where the number of best solutions found by FCM compared

to the modified GFCM is very low. Hence, this modified genetic FCM can be considered as a better evolving clustering approach. One small failure was noticed when the modified GFCM did not find the best solution in Data Set 2, which was found by the improved EFCM approach as shown in Figure 6.1. So, each of these two approaches has its own merits.

CHAPTER 7. CONCLUSIONS

This thesis applied EAs to improve the performance of FCM, since FCM may sometimes only get a local optimal solution. In Chapter 4, a basic EFCM approach was proposed. It had a tiny improvement in terms of Rand Index, compared with the normal FCM approach. In other words, the probability of getting better solutions was enlarged by EFCM. However, it takes a long time to find the solutions and the solutions obtained were not well optimized.

To solve the problems presented by the basic EFCM, I came out with a multi-level strategy aimed at having a better way to set parameters. A multi-rate mutation approach was proposed by applying the strategy to mutation rate, and a multi-length mutation was also proposed by applying the strategy on mutation step length. Both approaches had a better performance in terms of both Rand Index results and achieving a more efficient, faster performance of tasks.

Finally, a multi-rate, multi-length mutation was proposed, which got even better performances than the previous two approaches. What's more, this approach even found a new and better solution for Data Set 2, which could not be found by running FCM many times. However, some solutions are obtained at times that are not well optimized.

For further improving of the EFCM approach. I built another plan, which uses the normal FCM to optimize the solution obtained by EFCM with the help of FCM's powerful local searching ability. This approach can also be called improved EFCM. The solutions show a little improvement when all obtained solutions become real optimal solutions.

Another approach, improved GFCM, which is very similar to the improved EFCM, was introduced. However, this approach did not work well for the four clustering data sets I used. GFCM needed hundreds of times more computation than that required by the normal FCM, but could only attain the same results that the FCM attained. As the basis of this improved GFCM, I proposed a modified GFCM, which tried to solve the hidden conflict between normal FCM and the evolving process. This approach relaxes the FCM's work power to speed up the whole process, and it gets a large improvement in terms of saving time (with more efficiency and less run time). Modified GFCM needs only less than 10% time when compared to the original improved GFCM. In other words, the modified GFCM needs only a few dozen times than the normal FCM, and its outcomes are still very good and stable.

Both improved EFCM and modified GFCM have a similar time complexity, and they all have their own benefits and shortcomings. The results obtained by improved EFCM

still have some chance not to be the best. Modified GFCM can always obtain a same and good result, but it loses the ability of seeking the global optimal solutions, which are hard to find. Hence, the further considering is to build a new algorithms, which could have advantages of both algorithms.

REFERENCES

[1] De Jong, K. A. (2006). *Evolutionary computation: A unified approach*. Cambridge, Mass: MIT Press.

[2] Lin, C. D., Anderson‐Cook, C. M., Hamada, M. S., Moore, L. M., & Sitter, R. R. (2015). Using genetic algorithms to design experiments: A review. *Quality and Reliability Engineering International*, 31(2), 155-167.

[3] Corne, D., & Knowles, J. (2003). No free lunch and free leftovers theorems for multiobjective optimisation problems. *Evolutionary Multi-Criterion Optimization*. Berlin: Springer, 2362, 327-341.

[4] Sivaraj, R., & Ravichandran, T. (2011). A review of selection methods in genetic algorithm. *International journal of engineering science and technology*, 3, 5.

[5] Li, Y., Zhao, L., & Zhou, S. (2011). Review of Genetic Algorithm. *Advanced Materials Research*, 179, 365-367.

[6] Dunn, J. C. (1973). Fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *J Cybern*, 3(3), 32-57.

[7] Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms*. New York: Plenum Press.

[8] Fogel L. J., Owens A. J., & Walsh, M.J. (1966). Application of Evolutionary Programming. *IEEE Systems Science and Cybernetics Conference*, Washington, D.C.

[9] Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

[10] Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog, Stuttgart, 1973.

[11] Schwefel, H. (1975) Evolutionsstrategie und numerische Optimierung. Dr.-Ing. Thesis, *Technical University of Berlin, Department of Process Engineering*.

[12] Koza, J. (1992). *Genetic Programming*. Cambridge, MA: MIT Press.

[13] Jamshidi, M. (2003). Tools for intelligent control: Fuzzy controllers, neural networks and genetic algorithms. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1809), 1781-1808.

[14] Sivaraj, R., & Ravichandran, T. (2011). A review of selection methods in genetic algorithm. International Journal of Engineering Science and Technology, 3(5), 3792-3797.

[15] Keller, J., Liu, D., & Fogel, D. B., (2015). Basic Ideas and Fundamentals. *Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation*: unpublished book.

[16] Fogel, D. (2006). Evolutionary Computation: Toward a New philosophy of Machine Intelligence (3rd ed.). New York: IEEE Press/Wiley.

[17] Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer.

[18] Wikipedia. (31 January 2016, at 18:37). Cluster analysis. Retrieved from: https://en.wikipedia.org/wiki/Cluster_analysis.

[19] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336), 846-850.

[20] Wikipedia. (19 March 2016, at 08:20). Rand Index. Retrieved from: https://en.wikipedia.org/wiki/Rand_index.

[21] Peng, H., & Xu, L. (2007). Improved genetic FCM algorithm for color image segmentation. *Guangdian Gongcheng/Opto-Electronic Engineering*, 34(7), 126-134

[22] Singh, K. K., Mehrotra, A., Nigam, M. J., & Pal, K. (2013, April). Unsupervised change detection from remote sensing images using hybrid genetic FCM. In *Engineering and Systems (SCES), 2013 Students Conference on* (pp. 1-5). IEEE.

[23] Wei, C., Tingjin, L., Jizheng, W., & Yanqing, Z. (2010, May). An improved genetic FCM clustering algorithm. *In Future Computer and Communication (ICFCC), 2010 2nd International Conference on* (Vol. 1, pp. V1-45). IEEE.

[24] Garg, G., & Juneja, S. Extract Area of Tumor through MRI using Optimization Technique with Fuzzy C Means. *International Journal of Computer Applications (0975–8887) Volume.*