

A Local Search Method to K-clustering



Web Mining

Autumn 2006

<p>Author(s):</p> <p>Xiong Wen Wang Wenjuan Huang Jiaquan</p>	<p>Supervisor(s):</p> <p>Noureddine Bouhmala</p>
<p>Version: 1.0 Status: FINAL</p>	<p>Pages: 30 (including this page) Modified date: 2006-11-25</p>
<p>Key words: Randomized Local Search Method, SSE, K- Means</p> <p>Abstract: Clustering is a technique used in data mining with a wide range of applications, including business transaction and pattern recognition. Although there are a variety of approaches of clustering, each one has its own disadvantages and limitations. In order to find a new method of clustering with good quality, we introduce a local search method of k-clustering. This method is an improvement of classic method of clustering, and we suppose it to get better clustering with lower Sum of Square Error (SSE). By comparison with k-means method, the results show that lower SSE of our solution than that of the classic one comes out after implementation, which prove our hypothesis before we make the design.</p>	

Version Control

<i>Version</i> ¹	<i>Status</i> ²	<i>Date</i> ³	<i>Change</i> ⁴	<i>Author</i> ⁵
0.1	Draft	2006-10-24	Original creation of the report. problem description, background, solution and work plan	Xiong Wen, Wang Wenjuan Huang Jiaquan
0.2	Draft	2006-11-18	Problem description edited, requirement and discussion added	Xiongwen
0.3	Draft	2006-11-19	Design, implement, evaluation and testing	Huang Jiaquan
0.4	Draft	2006-11-20	Background edited	Wang Wenjuan
0.5	Draft	2006-11-20	Abstract and introduction	Xiongwen
0.6	Draft	2006-11-20	Appendix and Executive summary	Huang Jiaquan
0.7	Draft	2006-11-21	Conclusion	Wang Wenjuan
1.0	final	2006-11-25		

¹ **Version** indicates the version number starting at 0.1 for the first draft and 1.0 for the first review version.

² **Status** is DRAFT, REVIEW or FINAL

³ **Date** is given in ISO format: yyyy-mm-dd

⁴ **Change** describes the changes carried out since the previous version

⁵ **Author** is the one who did the change

Table of Contents

1. Executive summary	5
2. Introduction	6
3. Background	7
3.1 Data mining.....	7
3.2 Clustering.....	7
3.2.1 Previous work.....	7
3.2.2 k-means clustering	8
3.2.3 Local search method.....	9
3.2.4 Hill climbing	9
3.3 Python	10
4. Problem description.....	11
5. Requirements	12
5.1 Functional Requirements.....	12
5.2 Non-functional Requirements.....	12
6. Design of clustering	13
6.1 raw data processing.....	13
6.2 Randomized Local Search Method to clustering processing	13
6.3 K-Means algorithm to clustering processing.....	14
7. Implementation	16
7.1. Preprocessing.....	16
7.2 Implementation of the Randomized Local search method.....	16

7.2.1 Divide points	16
7.2.2 Calculate the SSE	17
7.2.3 main loop	18
7.3 Implementation of K-Means algorithm	18
8. Evaluation and testing	19
8.1 test results	19
8.2 result analysis.....	21
9. Discussion.....	23
10 Conclusion and further work	24
10.1 Conclusion	24
10.2 Further work	24
Appendix	25
A 1 Glossary & Abbreviations	25
A 2 Reference.....	25
A3 Code snippets	26
A3.1 randomly grouping the points	26
A3.2 calculate the SSE	26
A3.3 main loop	27
A3.4 K-Means (main loop)	28
A4 Work strategy	30

1. Executive summary

Data clustering is one of the common techniques in data mining. As an increasing amount of information and textual data, we are urgent to divide complex and difficult problems into small parts.

In this project, we introduce an approach - a new local search method for solving the k-clustering problem, we call it Randomized Local Search (in following report, we will use RLS for short). Local search provides a simple and effective approach to many combinatorial optimization problems. It starts from a candidate solution and then moves to a neighbor solution. The choice of the neighbor solution is done by using *hill climbing* [1]. In our project, the proposed approach starts by randomly separating the original data set into some smaller sets which is the candidate solution. Then we randomly select sets to exchange data instead of using hill climbing. This method is easy to implement and competitive with the best clustering methods. The ease of implement makes it possible to apply the algorithm to various clustering applications with different distance metrics or images.

In order to test the performance of this new approach, we introduce fifteen textual files which contain different number of points, ranging from 280 to 33810, which are various in different types such as positive integer, negative integer, and float number and so on. Each point has two attributes: coordinate x and coordinate y. The conclusion is based on the experimentation of these data points.

Finally, we compare the performance between the new approach and that of K-Means. The result would be illustrated in different conditions.

2. Introduction

Clustering is a technique used in data mining in which similarity between objects can be presented in such a way that object within a particular cluster would be similar or close while objects in different clusters should be quite different or unrelated. It has a wide range of applications, such as business transaction and pattern recognition [13].

Clustering techniques vary greatly in the approaches of dividing data objects. We can classify them into two general categories based upon types of clustering methods: partitional clustering and hierarchical clustering. The first group, on which the method we apply in the project is based, ensures each data object belongs to one subset and there are no overlapping subsets. The number of clusters formed is typically chosen before dividing the data objects and referred to K as a reference. K-means is a classic algorithm in partitional clustering, with which we want to compare our method.

This report focuses on testing the quality of RLS method of K-clustering, a new method of clustering. As an improvement of k-means method, we suppose it performs better in clustering data than k-means. We hope that we can find a new approach for clustering with good quality.

In contrast to the previous approaches, we will test and implement RLS method through 15 files containing different numbers of points with two attributes: coordinate x and coordinate y . Also, we implement k-means algorithm for comparison under the some inputs. The results expectedly show that our solution is much better than the classic one because of the lower SSE. Meanwhile, the output gives the best clustering by determining the number of clusters.

The report is organized in 10 sections. We introduce the related information about clustering in section 3 and in section 4; we present the problem description of our project. In section 5, we list the requirement specifications of this problem, followed by an overall design of the RLS method in section 6. Section 7 is about the implementation of the solution based on the two algorithms in last section. Then we test the methods and compare the quality of two methods mentioned above in the next section. Some difficulties met and adjustment made during the process of project is discussed in section 9. In the last section, the conclusion is drawn from the results of implementation.

3. Background

In this section, we will describe the background of our project and previous work done. We look at the basic ideas and arithmetic that we have used in our project in the following content.

3.1 Data mining

There is a growing enormously rate about data kept in computer files and databases. This data would be impracticable, but some of them would be waiting to be discovered by user. How can we select valuable and important information needed?

Informally speaking, data mining is the process of extracting information or knowledge from a data set for the purpose of decision making [2]. With the rapid increase of large data sets, it becomes an active field in recent years. The technology of data mining is applied to scientific research, business performance management and many other fields.

Data mining supplies different algorithms according to different tasks. All algorithms attempt to fit a model to the data. [3] Model can be either predictive or descriptive in nature. Clustering is considered as descriptive in nature.

3.2 Clustering

Clustering is the process of dividing data objects into clusters, so that the objects in a group will be similar to one another and different from the objects in other groups. And clustering is also considered unsupervised learning. It is used in many fields, including image processing, business transaction analysis, and pattern recognition.

Clustering algorithms contain hierarchical and partitional. Hierarchical clustering is not related to our project, so we will introduce another one in-depth.

Partitional clustering is to divide data objects into non-overlapping subsets, so that each data objects is in exactly one subset. However, hierarchical clustering has tremendous difference compared with former clustering. It forms a hierarchical tree that is composed of a set of nested clusters. K-means clustering is a main algorithm in partitional clustering. Through K-means, some other algorithms yield like QT Clust algorithm and Fuzzy c-means clustering.[4]

3.2.1 Previous work

QT (Quality Threshold) Clustering, put forward by Heyer et al in 1999, is an alternative method of partitioning data and is invented for gene clustering. It requires more computing power than K-means, but does not require specifying

the number of clusters, and always returns the same result when run several times.[4]

On the other hand, in fuzzy clustering, each point has a degree of belonging to clusters, as in fuzzy logic, rather than belonging completely to just one cluster. Thus, points on the edge of a cluster, may be in the cluster to a lesser degree than points in the center of cluster.

The fuzzy c-means algorithm is very similar to the K-means algorithm. This algorithm minimizes intra-cluster variance as well, but has the same problems as K-means, the minimum is a local minimum, and the results depend on the initial choice of weights. [4]

3.2.2 k-means clustering

K-means clustering, put forward by J. Mac Queen in 1967, is one of partitional clustering approaches. Each cluster has a centroid(center point), and center point is the center of all points in a cluster.

Example: the data set has two dimensions and the cluster has three points $A=(x_1, y_1)$, $B=(x_2, y_2)$ and $C=(x_3, y_3)$. Then center point $D=(x_4, y_4)$, where $x_4=(x_1+x_2+x_3)/3$; $y_4=(y_1+y_2+y_3)/3$. Each point is assigned to the cluster whose center is the nearest. The basic algorithm has four steps. The exact steps have been mentioned in the section of problem description.

SSE, Sum of Squared Error is the most common measure.

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist^2(m_i, x)$$

k is the number of clusters; x is a data point in cluster C_i and m_i is the representative point for cluster C_i . [4] This equation will be used in K-means algorithm and local search method. We can see that the easy way to reduce SSE is to increase k, the number of clusters, but a good clustering with smaller k can have a lower SSE than a poor clustering with higher K. Thus, we want to find a good clustering with small k, and have a lower SSE by this equation.

The main advantages of this algorithm are its simplicity and speed which allows it to run on large datasets. Its disadvantage is that it does not yield the same result with each run, since the resulting clusters depend on the initial random assignments. It maximizes inter-cluster(for minimizes intra-cluster) variance, but does not ensure that the result has a global minimum of variance.[4]

3.2.3 Local search method

Local search is a technique which solves computationally hard optimization problems. Local search is used for solving problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions until a solution deemed optimal is found or a time bound is elapsed. Some problems apply this technology to practice, for example the vertex cover problem, the traveling sales man problem and the Boolean satisfiability problem. Most problems can be informed in terms of search space and target in several different manners.[7]

A local search algorithm starts from a selected solution to a neighbor solution. If a neighborhood relation is defined on the search space, this is only possible. Now taking an example, the neighborhood of a cycle contain all nodes of the graph is another cycle only differing by one length of the cycle. For Boolean satisfiability, the neighbors of a truth assignments are usually the truth assignments only differing from it by the evaluation of a variable.[7]

Aim of local search can be based on a time bound. Another common choice is to reach when the best solution found by the algorithm has not been improved in a given number of steps. Local search algorithms are typically incomplete algorithms, as the search may stop even if the best solution found by the algorithm is not optimal. [7]

Local search algorithms are applied to some hard computational problem, including mathematics, engineering and computer science. Because it is a new algorithm, few people heard it and done searching in this aspect. However our supervisor combined local search and genetic algorithms with the multilevel paradigm to search the traveling salesman problem which would be mentioned in conclusion part. [8]

3.2.4 Hill climbing

Hill climbing is an optimization technique which belongs to the family of Local search. Hill climbing is to maximize (minimize) a function $f(x)$, in which x is a discrete states. These states are changing. Hill climbing will vary from vertex to vertex. It also operates on a continuous space. It is used widely in different fields, like artificial intelligence.

In simple hill climbing, the first closer node is chosen and the closest to the solution is chosen. If there are local maxima in the search space, it may happen. Random-restart hill climbing is built on the top of the hill climbing algorithm. It simply runs an outer loop over hill-climbing. It is a

effective algorithm in many cases. It turns out that it is often better to spend CPU time exploring the space, rather than optimizing from an initial condition. [1]

3.3 Python

The name Python is derived from Monty Python, not from snakes. It is “an interpreted, object-oriented, high-level programming language with dynamic semantics.” It enables you to implement the functionality you want without any hassle, and lets you write programs that are clearer and more readable than programs in most other popular programming languages. [6] So we choose it as my platform. Using Python, we program two different algorithms, and implement them. At last the results are recorded.

4. Problem description

We want to test whether the RLS method to K-Clustering, a new method we use in the project is better than the classical K-means method. K-means method [8] is a method “where each cluster is represented by the mean value of the objects in the cluster”. K-means is supposed to be a good method in clustering points to optimize a certain criterion. By applying this method, a point in a cluster is closer to the points in the cluster than any points in other clusters. Consequently, we need a measurement to compare the two methods. In this project, we use SSE (Sum of Square Error) to decide which method is better. The lower SSE we get in the final result, the better the method is.

The main objective of the project is to find best clusters with the lowest SSE by using the new method. We have a set of points that range from 280 to 33810, and we have to divide them into a certain number of clusters. The differences from K-means method lie in that the points are divided into clusters randomly. By exchanging two points in two different clusters with each point belonging to each cluster, we can get the value of SSE. After exchanging the points for 1000 times, SSE would be stable.

The RLS method is supposed to be better in clustering according to the algorithms that we mentioned in background. In this project we will verify this and attempt to make some improvements based on K-means method. But further testing is still necessary before we reach the final conclusion.

5. Requirements

Before we start the solution of the project, we want to present the requirements necessary for the project.

5.1 Functional Requirements

Testing data resource: Initially, we have got points that are to be clustered from our supervisor. The 15 files contain different number of points, ranging from 280 to 33810, which are various in different types such as positive integer, negative integer, and float number and so on. Each point has two attributes: coordinate x and coordinate y. The conclusion is based on the experimentation of these data points.

Input: We can decide which file we want to test and how many clusters we desired

Randomly grouping the points: we can divide all the points into k clusters randomly

Exchange points: it must be possible to exchange the points in two random clusters randomly.

Calculate SSE: we can calculate the SSE in order to find the best clusters.

Implement K-Means algorithm: we can compare the performance between RLS method and a classic algorithm which we choose is K-Means.

Output: we want to present the clusters that are best clustered in order. As a result, we can see which clustering is the best.

5.2 Non-functional Requirements

Time consuming: the run-time should be as short as possible

System requirement: the algorithm can be implemented under Windows and Linux

User friendly: the algorithm should be easy for user to debug and input data

6. Design of clustering

In this section, we will present the raw data processing before clustering then illustrate the RLS method of clustering (new algorithm) and K-Means algorithm we used in our project.

6.1 raw data processing

Before the clustering process can start, we have to read data from files and change their type into float coordinates. The concept can be described below, in figure 1.

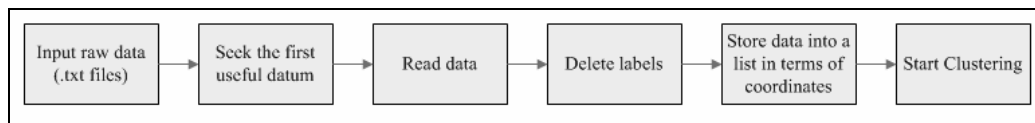


Fig 1: raw data processing

Initially, the data were stored in .txt files with some text such as file name and some explanations. Then, find the data which are our target and remove the label of each pair of data (for example: 1 2.0, 3.3 2 3.4, 5.6 in which 1 and 2 are labels we want to remove). Finally, store the data in a list in terms of coordinates which we can use to start clustering.

6.2 Randomized Local Search Method to clustering processing

The algorithm can be presented as follow and we present a flow chart below in fig 2:

1. Divide all the points into K clusters randomly
2. Calculate the initial centroid of each cluster
3. Repeat 1000 times:
 4. Calculate the distance between each point and the centroid in a cluster(SSE)
 5. Choose 2 clusters at random, exchange 1 point from each cluster and Calculate the new centroid of each.
6. Compare all the SSE and choose the lowest one as the best clustering

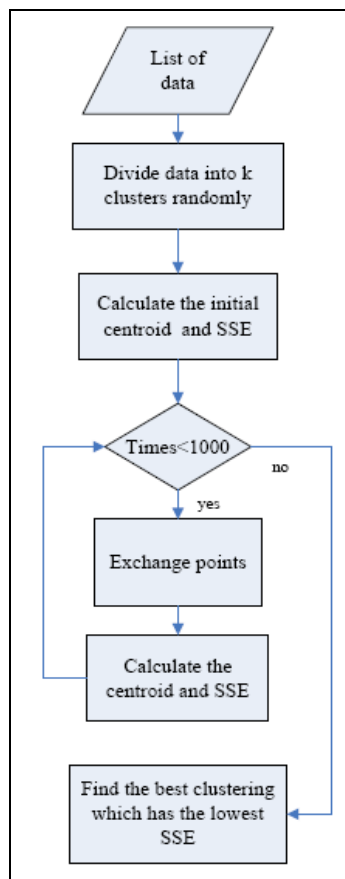


Fig 2 : clustering processing

We have to mention that the iterate times is supposed to be 1000 according to our tests. We choose 100, 1000, 2000 and 3000 as iterate times in the test, the result of which can be presented below in table 1

	Iterate times			
	100	1000	2000	3000
280(SSE)	2571042.87143	2533580.87143	2489022.64286	2496133.95714
1291(SSE)	1449057281.08	1446776023.63	1440619733.23	1443626055.58
14051(SSE)	81952521013.9	81942421247.8	81909806217.8	81946345219.8

Table 1 : test of iterate times based on 280 ,1291 and 14051 points

We can find that the lowest SSE changes slightly after 1000 times' iteration, which reaches the lowest value until 2000 times' iteration. Nevertheless, as the iterate time increasing, the speed would be slow down. In our test, the time spent on 2000 times' iteration is nearly as twice as that on 1000 times' iteration. Considering the quality and speed, we choose 1000 as our iterate time.

6.3 K-Means algorithm to clustering processing

K-Means is a classic algorithm which is still popular in clustering because of its extremely quick convergence. This algorithm can be presented below in fig 3:

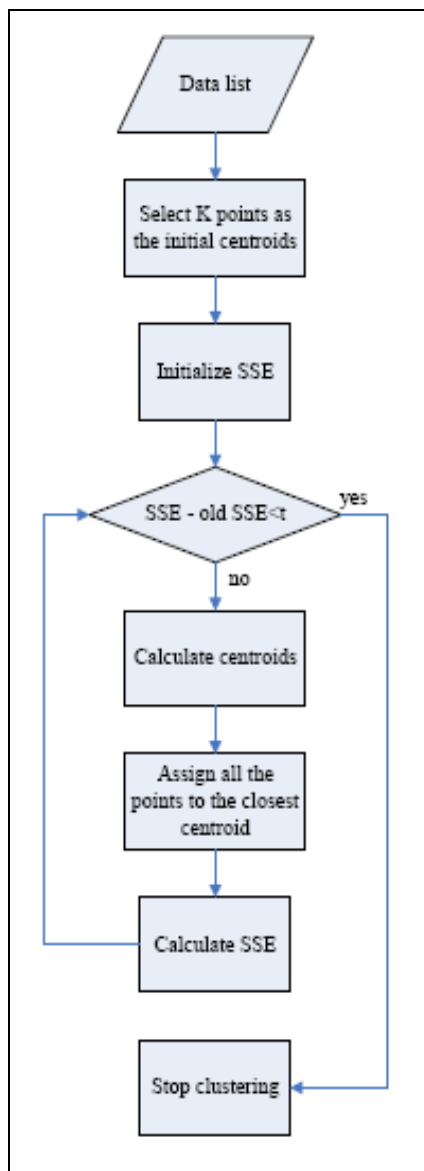


Fig 3: K-Means algorithm, t is error tolerance, used as the stopping criterion

The error tolerance is a judgment which depends on the user's requirement. In this project we define the error tolerance as 0.0001 that means that when the SSE of an iteration is within 0.0001 ranges from that of the previous iteration, the centroids of clusters are considered "stable".

We initialize SSE 1.797693e308 which presents infinity according to some references ([9],[10]). We use the infinity value in order to make sure our iteration can start.

7. Implementation

After setting up the clustering model, the implementation would be illustrated in detail in this section.

7.1. Preprocessing

1. Input your requirements

```
# input
f=open(input('enter the filename:\n'),'r')
num =input('the number of the points:\n')
number=input('the number of the clusters:\n')
```

Fig 4: the file you want to clustering and desired number of clusters

2. Data processing

```
# read data from files
file.seek(position of useful data)
for i in range(number of points):
    temp_testdata=file.readline()
    str=temp_testdata.split()
    del str[0]
    testdata.append(map(float,str))
file.close
```

Fig 5: deal with data (pseudo code)

It is difficult to delete the textual title automatically since it is combined with text, iterations and numbers which are hard to be partitioned from the data we intend to use. (see fig 6):

```
NAME : a280 COMMENT : drilling problem (Ludwig) TYPE : TSP DIMENSION: 280 EDGE_WEIGHT_TYPE :
149 2 288 129 3 270 133 4 256 141 5 256 157 6 246 157 7 236 169 8 228 169 9
204 169 13 196 169 14 188 169 15 196 161 16 188 145 17 172 145 18 164 145 19 156 145
23 164 169 24 172 169 25 156 169 26 140 169 27 132 169 28 124 169 29 116 161 30 104 1
```

Fig 6: raw data with text

In this case, we use method seek() to find the useful data directly.

7.2 Implementation of the Randomized Local search method

7.2.1 Divide points

```

def clustering(testdata, numberofclusters)
    numberofpoints=len(testdata)/numberofcluster
    cluster = []
    temp_cluster = []
    for i in length of testdata:
        temp_cluster = random.sample(testdata,numberofpoints)
        cluster.append(temp_cluster)
        for uesdpoint in temp_cluster:
            testdata.remove(usedpoint)
        cluster.append(temp_cluster)
    return clsuter

```

Fig 7: divide points into k clusters (pseudo code)

The method we that we use is random.sample() which can select points randomly.

7.2.2 Calculate the SSE

```

def center(blist):
    """center
    Examples:
    center([1,2],[4,5],[6,7])
    [3.333333,4.666666]
    """
    import math
    clist=[]
    for b in blist:
        clist.append(b[:])

    dif2=[]
    dis = []
    n = len(clist[0]) # dimension
    centers=[0.0]*n
    s=[0.0]*len(blist)
    sse=0

    for i in range(n):
        for j in range(len(blist)):
            s[i]=s[i]+blist[j][i]
            centers[i]=s[i]/len(clist)
        for i in range(n):
            dif=[]
            for j in range(len(clist)):
                dif.append(clist[j][i] - centers[i])
            dif2.append(dif)
        for i in range(len(clist)):
            distance=0
            for j in range(n):
                distance+=math.pow(dif2[j][i],2)
            dis.append(distance)
        for i in dis:
            sse = sse+i
    return sse

```

Fig 8: calculate the SSE

This snippet of code implements calculating the centroid and SSE, basing on Euclidean distance. Though the points are given in terms of two dimensions of coordinates, the dimensions can be expanded to any quantity you want.

7.2.3 main loop

This part contains two steps, one is exchange points after selecting two clusters randomly (see fig 9), and the other is calculating SSE which has been presented above (see fig 8). These two steps would be iterated for 1000 times as we mentioned in the previous section.

```
# randomly choose 2 clusters and exchange points
excluser=random.sample(clusters,2)
for i in excluser:
    if i in clusters:
        clusters.remove(i)

excluser[0].append(random.choice(excluser[1]))
for i in excluser[1]:
    for j in excluser[0]:
        if i==j:
            k=i
            excluser[1].remove(i)
excluser[1]=excluser[1]+random.sample([i for i in excluser[0] if i!=k],1)

for i in excluser[0]:
    for j in excluser[1]:
        if i==j:
            excluser[0].remove(i)
inclusters1=[]
clusters=clusters+excluser
for c in clusters:
    inclusters1.append(c[:])
sumclusters.append(inclusters1)
```

Fig 9: divide points into k clusters

7.3 Implementation of K-Means algorithm

The code was adapted from a C version by Roger Zhang[11]. The main part of loop is below (see fig 10):

```
for h in xrange(n):
# identify the closest cluster
    min_distance = DOUBLE_MAX
    for i in xrange(k):
        distance = 0
        for j in xrange(m):
            diff = data[h][j] - c[i][j]
            distance += diff * diff
        if distance < min_distance:
            labels[h] = i
            min_distance = distance

# update size and temp centroid of the destination cluster
    for j in xrange(m):
        c1[labels[h]][j] += data[h][j]
    counts[labels[h]] += 1
# update SSE
    SSE += min_distance
```

Fig 10: calculate SSE and assign points

8. Evaluation and testing

8.1 test results

The whole result including the SSE and best clusters can be found in Appendix 2. We list the SSE here to analyze the performance of these two algorithms.

test1. Assign all the points to 4 clusters

Number of points	280	493	532	535	657
RLS method	2.560e+6	2.875e+8	3.488e+9	2.062e+6	6.862e+8
K-Means	6.233e+6	6.504e+8	8.473e+9	5.168e+6	1.724e+9
Ratio of results	0.41	0.44	0.41	0.40	0.40

Number of points	1291	1655	3038	3795	4461
RLS method	1.446e+9	1.758e+9	5.920e+9	2.175e+9	9.793e+9
K-Means	3.724e+9	4.502e+8	1.475e+10	5.022e+9	2.233e+10
Ratio of results	0.39	0.39	0.40	0.43	0.44

Number of points	11849	14051	15112	18512	33810
RLS method	3.996e+11	8.194e+10	7.474e+11	1.193e+11	1.611e+15
K-Means	9.74e+11	1.75e+11	1.86e+12	2.82e+11	4.164e+15
Ratio of results	0.41	0.47	0.40	0.42	0.39

Table 2: comparison of the lowest SSE of the two algorithms when the points are divided into 4 clusters, Ratio=RLS method/K-Means

test2. Assign all the points to 10 clusters

Number of points	280	493	532	535	657
RLS method	2.500e+6	2.803e+8	3.388e+9	2.031e+6	6.674e+8
K-Means	8.480e+6	9.770e+8	9.2577e+9	5.893e+6	2.441e+9
Ratio of results	0.29	0.29	0.36	0.34	0.27

Number of points	1291	1655	3038	3795	4461
RLS method	1.435e+9	1.753e+9	5.90e+9	2.174e+9	9.793e+9
K-Means	5.477e+9	6.159e+8	2.073e+10	7.220e+9	3.476e+10
Ratio of results	0.26	0.28	0.29	0.30	0.28

Number of points	11849	14051	15112	18512	33810
RLS method	3.992e+11	8.190e+10	7.470e+11	1.193e+11	1.610e+15
K-Means	1.376e+12	2.492e+11	2.456e+12	4.122e+11	5.688e+15
Ratio of results	0.29	0.32	0.30	0.30	0.28

Table 3: comparison of the lowest SSE of the two algorithms when the points are divided into 10 clusters, Ratio=RLS method/K-Means

test3. Assign all the points to 16 clusters

Number of points	280	493	532	535	657
RLS method	2.353e+6	2.774e+8	3.377e+9	1.995e+6	6.651e+8
K-Means	8.799e+6	1.306e+9	1.105e+10	7.858e+6	2.899e+9
Ratio of results	0.27	0.21	0.30	0.25	0.23

Number of points	1291	1655	3038	3795	4461
RLS method	1.438e+9	1.748e+9	5.905e+9	2.166e+9	9.767e+9
K-Means	6.323e+9	6.841e+8	2.344e+10	1.038e+10	3.681e+10
Ratio of results	0.23	0.26	0.25	0.21	0.26

Number of points	11849	14051	15112	18512	33810
RLS method	3.991e+11	8.186e+10	7.464e+11	1.192e+11	1.611e+15
K-Means	1.584e+12	2.808e+11	2.823e+12	4.697e+11	6.862e+15
Ratio of results	0.25	0.25	0.26	0.30	0.24

Table 4: comparison of the lowest SSE of the two algorithms when the points are divided into 16 clusters, Ratio=RLS method/K-Means

Test4 Assign all the points to 20 clusters

Number of points	280	493	532	535	657
RLS method	2.403e+6	2.801e+8	3.332e+9	1.988e+6	6.597e+8
K-Means	9.512e+6	1.404e+9	1.188e+10	8.908e+6	3.057e+9
Ratio of results	0.25	0.20	0.28	0.22	0.22

Number of points	1291	1655	3038	3795	4461
RLS method	1.431e+9	1.739e+9	5.899e+9	2.166e+9	9.754e+9
K-Means	6.741e+9	7.579e+9	2.529e+10	8.374e+9	3.929e+10
Ratio of results	0.21	0.23	0.23	0.26	0.25

Number of points	11849	14051	15112	18512	33810
RLS method	3.991e+11	8.185e+10	7.463e+11	1.191e+11	1.610e+15
K-Means	1.714e+12	3.189e+11	3.055e+12	4.779e+11	7.531e+15
Ratio of results	0.23	0.26	0.24	0.25	0.21

Table 5: comparison of the lowest SSE of the two algorithms when the points are divided into 20 clusters, Ratio=RLS method/K-Means

Test5 Assign all the points to 24 clusters

Number of points	280	493	532	535	657
RLS method	2.323e+6	2.653e+8	3.413e+9	2.032e+6	6.562e+8
K-Means	9.770e+6	1.407e+9	1.256e+10	7.351e+6	3.209e+9
Ratio of results	0.24	0.19	0.27	0.28	0.20

Number of points	1291	1655	3038	3795	4461
RLS method	1.414e+9	1.732e+9	5.894e+9	2.170e+9	9.755e+9
K-Means	6.990e+9	8.568e+9	2.858e+10	8.541e+9	4.131e+10
Ratio of results	0.20	0.20	0.21	0.25	0.24

Number of points	11849	14051	15112	18512	33810
RLS method	3.991e+11	8.182e+10	7.466e+11	1.191e+11	1.610e+15
K-Means	1.877e+12	3.315e+11	3.213e+12	5.162e+11	7.538e+15
Ratio of results	0.21	0.25	0.23	0.23	0.21

Table 6: comparison of the lowest SSE of the two algorithms when the points are divided into 24 clusters, Ratio= RLS method/K-Means

Test6 Assign all the points to 32 clusters

Number of points	280	493	532	535	657
RLS method	2.250e+6	2.648e+8	3.306e+9	1.998e+6	6.592e+8
K-Means	1.017e+7	1.423e+9	1.296e+10	1.200e+7	3.588e+9
Ratio of results	0.22	0.19	0.26	0.17	0.18

Number of points	1291	1655	3038	3795	4461
RLS method	1.442e+9	1.723e+9	5.868e+9	2.162e+9	9.731e+9
K-Means	7.894e+9	8.621e+9	2.819e+10	9.419e+9	4.469e+10
Ratio of results	0.18	0.20	0.21	0.23	0.22

Number of points	11849	14051	15112	18512	33810
RLS method	3.987e+11	8.181e+10	7.462e+11	1.191e+11	1.610e+15
K-Means	1.917e+12	3.097e+11	3.361e+12	5.669e+11	8.103e+15
Ratio of results	0.21	0.26	0.22	0.21	0.20

Table 7: comparison of the lowest SSE of the two algorithms when the points are divided into 32 clusters, Ratio=RLS method/K-Means

8.2 result analysis

According to the tables above, we infer to some conclusions as follow:

1. Considering the result of clustering, the performance of RLS method is better than K-Means because the RLS method can reach much lower SSE

- than that of K-Means. The lower SSE the better convergence of clusters.
2. As the increasing number of clusters, the performance of RLS method becomes better which can be indicated in fig 11.

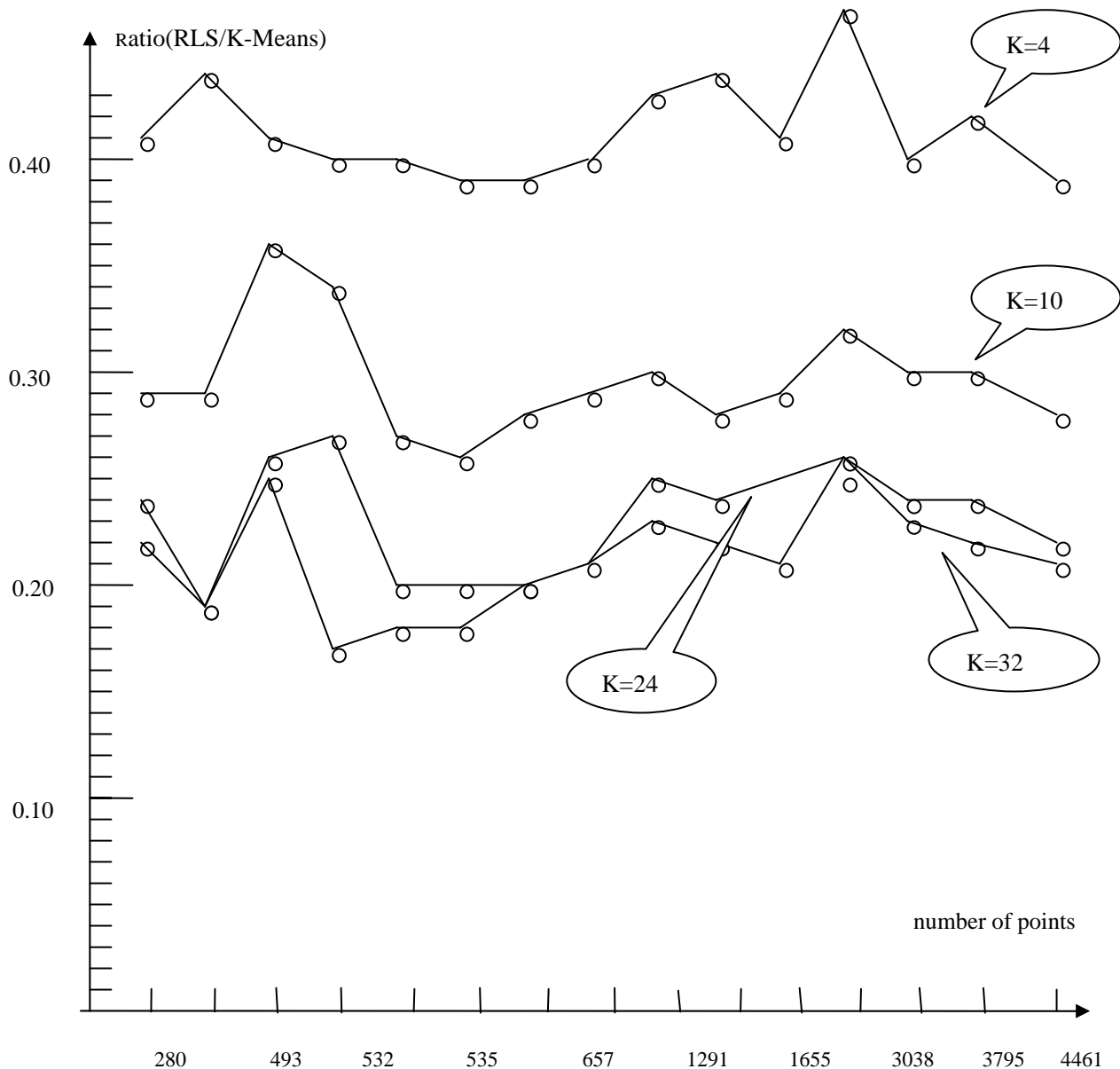


Fig 11. The ratio of RLS and K-Means (note: the ratio goes down when increase the number of clusters, but the ratio becomes stable after 20 clusters)

3. To RLS method, the performance changes along with the number of desired clusters. The more clusters divided the lower SSE would be. In the opposite, the lowest SSE increases along with the number of clusters in K-Means algorithm.

However, if we consider the runtime, the extremely quick convergence highlights the advantage of K-Means. During the test we found that even if the number of points is 33810 which is assigned to 50 clusters, the centroid would be stable after 366 times' iteration.

9. Discussion

First of all, we wrote the programs that can read all the points from the files and restore them in an array. In the beginning, we just planned to cluster all the points we get to find out the best clustering by applying the RLS method to K-Clustering. Actually, we have done it successfully and managed to get the lowest value of SSE.

However, later we found out that it was not enough to reach a conclusion on the performance of the new method. Consequently, we should compare this method with the method that is already known. The new method is an improvement of K-means algorithm. At last, we decided to select this classic method as a reference. In fact, the result that we get proved that the new algorithm performed better than K-means. As a result, the new method with better performance can be available in the application of clustering.

During the processing, time spent on implementation of all the points varies in the different numbers of points and the more points were, the longer time it cost. When it came to 33810 points that are the largest number, it could cost one hour or more to get the final result. Each time we ran the program, we may get different result because we divided and exchanged the points randomly. Different number of clusters also contributed the disparity of clustering. These have been illustrated in former sections.

Using RLS method to K-clustering will have the lowest sum of square errors in clustering large number of data sets. Furthermore, when the number of clusters desired is large enough, the advantages will be more obvious.

10. Conclusion and further work

10.1 Conclusion

In this report, we have presented the advantages of the RLS method to K-clustering, comparing with K-means clustering. Whatever we divide all the points into four, ten clusters or more, the results show that SSE of new method is lower than that of K-means method. In all, the more clusters divided the better the results reach our ideal. In addition, our program is not only suitable for two dimensions, but also for three dimensions or more.

However, comparing with rapid convergence of K-means algorithm, the new algorithm operates more slowly because of 1000 times iteration. When we tested them, the time consumed in new method was the 30 times than that in previous method. Furthermore, these two kinds of algorithm don't yield the same result with each run, since the resulting clusters depend on the initial random assignments. It maximizes inter-cluster (for minimizes intra-cluster) variance, but does not ensure that the result has a global minimum of variance.

Thus our conclusion is that local search method is superior to K-means clustering in the whole, before having new findings in this field.

10.2 Further work

Concerning the future work related to clustering algorithm, it is very important to find a new method that is less time-consuming and strong with convergence. Maybe others could consider a multilevel local search method for solving the k-clustering problems. Multilevel techniques refer to the process of diving large and difficult problem into small ones, which are expected to handle easily. Using a way to solve solutions is that a previous level is considered as a starting solution at the next level. The proposed approach starts by coarsening the original problem into a sequence of smallest problem using coarsening scheme. Thereafter considering the smallest problem is the key of problems. After solving this problem, go back to the original problems.[12]

Appendix

A 1 Glossary & Abbreviations

SSE: Sum of Square Error

RLS: Randomized Local Search

A 2 Reference

[1]Hill Climbing, http://en.wikipedia.org/wiki/Hill_climbing

[2] U.Fayyad and R.Uthurusamy. **Data mining and knowledge discovery in databases.** Communication of the ACM, 39(11): 24-26, 1996

[3]Margaret H.Dunham. **Data Mining introductory and advanced topics.** 4: 17-18.

[4]http://en.wikipedia.org/wiki/Data_clustering

[5]<http://eiao.net/webmining/projects>

[6]Magnus Lie Hetland, **Beginning Python: From novice to professional.** Apress,2005

[7]http://en.wikipedia.org/wiki/Local_search_%28optimization%29

[8] J.MacQueen. **Some methods for classification and analysis of multivariate observation.** In Proceedings of the 5th Berkeley Symposium on Math. Statist. Problems, pages 281-297, 1967

[9]support.sas.com/documentation/onlinedoc/91pdf/sasdoc_91/or_ug_math_7309.pdf

[10] <http://www.csc.fi/cshelp/sovellukset/stat/sas/sasdoc/sashtml/ormp/chap3/sect52.htm>

[11]C version Zhang http://cs.smu.ca/~r_zhang/code/kmeans.c

[12]<http://eiao.net/webmining/projects>

[13]George Chang, Marcus J.Healey, James A.M. McHugh, Jason T.L.Wang, **Mining eh World Wide Web: An Information Search Approach.** 78

A3 Code snippets

A3.1 randomly grouping the points

```
#!/usr/bin/python
# Filename: clustering.py

' generate 4 clusters'

def cluster(alist,number1):
    """ Clusting
    Examples:
    >>> cluster([1,2,3,4,5,6,7,8],4)
    [[1,2],[3,4],[5,6],[7,8]]
    """
    import random
    retlist = []
    blist = []
    for a in alist:
        blist.append(a[:])
    #used = []
    for i in range(number1-1):
        ran = random.sample(blist,len(alist)/number1)
        retlist.append(ran)
        for j in ran:
            blist.remove(j)
    retlist.append(blist)
    return retlist

version = '0.1'

# End of clustering.py
```

A3.2 calculate the SSE

```
#!/usr/bin/python
# Filename: ssecal.py

def center(blist):
    """center
    Examples:
    center([1,2],[4,5],[6,7])
```

```

[3.333333,4.666666]
"""
import math
clist=[]
for b in blist:
    clist.append(b[:])

dif2=[]
dis = []
n = len(clist[0]) # dimension
centers=[0.0]*n
s=[0.0]*len(blist)
sse=0

for i in range(n):
    for j in range(len(blist)):
        s[i]=s[i]+blist[j][i]
    centers[i]=s[i]/len(clist)
for i in range(n):
    dif=[]
    for j in range(len(clist)):
        dif.append(clist[j][i] - centers[i])
    dif2.append(dif)
for i in range(len(clist)):
    distance=0
    for j in range(n):
        distance+=math.pow(dif2[j][i],2)
    dis.append(distance)
for i in dis:
    sse = sse+i
return sse

version = '0.2'

# End of ssecal.py

```

A3.3 main loop

```

# main loop
times=0
while times <1000:
    # randomly choose 2 clusters and exchange points
    excluster=random.sample(clusters,2)

```

```

for i in excluster:
    if i in clusters:
        clusters.remove(i)

excluster[0].append(random.choice(excluster[1]))
for i in excluster[1]:
    for j in excluster[0]:
        if i==j:
            k=i
            excluster[1].remove(i)
excluster[1]=excluster[1]+random.sample([i for i in excluster[0] if i!=k],1)

for i in excluster[0]:
    for j in excluster[1]:
        if i==j:
            excluster[0].remove(i)
inclusters1=[]
clusters=clusters+excluster
for c in clusters:
    inclusters1.append(c[:])
sumclusters.append(inclusters1)

# calculate Squared Error
sselect=[]
for j in range(number):
    sselect.append(ssecal.center(clusters[j]))
sumlist.append(sselect)
del sselect
times+=1

```

A3.4 K-Means (main loop)

```

flag = 0

# main loop
while True:
    # save error from last step
    old_SSE = SSE
    SSE = 0

    # clear old counts and temp centroids
    for i in xrange(k):

```

```

        counts[i] = 0
        for j in xrange(m):
            c1[i][j] = 0.0

    for h in xrange(n):
        # identify the closest cluster
        min_distance = DOUBLE_MAX
        for i in xrange(k):
            distance = 0
            for j in xrange(m):
                diff = data[h][j] - c[i][j]
                distance += diff * diff
            if distance < min_distance:
                labels[h] = i
                min_distance = distance

        # update size and temp centroid of the destination cluster
        for j in xrange(m):
            c1[labels[h]][j] += data[h][j]
        counts[labels[h]] += 1
        # update SSE
        SSE += min_distance

    for i in xrange(k): # update all centroids
        for j in xrange(m):
            if counts[i]:
                c[i][j] = c1[i][j] / counts[i]
            else:
                c[i][j] = c1[i][j]

    flag += 1
    print "%d) SSE:" % flag, SSE, ", ", "error tolerance:", abs(SSE -
old_SSE)
    if (abs(SSE - old_SSE) < t) or (flag > maxiter):
        break

```

A4 Work strategy

Project progress		Time plan
Analysis project and divide tasks		12.sep.2006-28.sep.2006
Learning Python and Java		12.sep.2006-3.oct.2006
Decide the algorithm		28.sep.2006-3.oct.2006
Coding:	Open file to read data and store them into coordinate type	3.oct.2006-19.oct.2006
	Complete the algorithm	20.oct.2006-30.oct.2006
	Test and improve the algorithm	31.oct.2006-10.nov.2006
Complete the report		11.nov.2006-25.nov.2006