CLUSTERABILITY AND
CENTROID APPROXIMATION

By

Alina Romero

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF COMPUTER SCIENCE

Approved:

Dr. Mikkai Krishnamoorthy
Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

November 2002
(For Graduation December 2002)
TABLE OF CONTENTS

LIST OF FIGURES........................................................................................................ IV
ACKNOWLEDGMENTS................................................................................................. V
ABSTRACT.................................................................................................................. VI

1. INTRODUCTION....................................................................................................... 1
   1.1 Motivation........................................................................................................ 3
   1.2 Context of prior work...................................................................................... 3
   1.3 Contributions.................................................................................................. 4
   1.4 Overview of material...................................................................................... 4

2. CLUSTER ANALYSIS AND CLUSTERABILITY...................................................... 5
   2.1 Cluster analysis vs. Clusterability................................................................. 5
   2.2 Types of clustering......................................................................................... 5
       2.2.1 Hierarchical............................................................................................. 6
       2.2.2 Non-hierarchical................................................................................... 6
   2.3 K-means clustering........................................................................................ 6
   2.4 Clusterability.................................................................................................. 8
       2.4.1 Graph based methods vs. non-graphic methods................................. 8

3. RELATED WORK..................................................................................................... 10
   3.1 Recent trends in clustering algorithms....................................................... 10
   3.2 Work in clusterability................................................................................... 12

4. BACKGROUND AND THEORY............................................................................ 14
   4.1 Notations and definitions............................................................................. 14
   4.2 Types of clusters............................................................................................ 15
       4.2.1 Non-properly clusterable dataset......................................................... 15
       4.2.2 Perfectly clusterable dataset............................................................... 16
       4.2.3 Uniform dataset.................................................................................. 17

5. CLUSTERABILITY ALGORITHM........................................................................ 19
   5.1 Design goals.................................................................................................... 19
   5.2 Random sampling algorithms..................................................................... 20
   5.3 Edge frequency histogram.......................................................................... 22
5.4 Cut-offs.................................................................................. 23
5.5 Centroids.................................................................................. 24
5.6 Spies......................................................................................... 24
5.7 Time series............................................................................... 25
5.8 Level of confidence.................................................................. 26
5.9 Formal description of the algorithm........................................ 26
6. EXPERIMENTS.......................................................................... 29
   6.1 Two dimensional data........................................................... 29
       6.1.1 Samples......................................................................... 29
       6.1.2 Known number of clusters.......................................... 32
   6.2 Time series............................................................................ 33
7. RESULTS AND PERFORMANCE ANALYSIS............................ 41
   7.1 Time complexity.................................................................... 41
8. CONCLUSIONS AND FUTURE DIRECTIONS............................ 42
   8.1 Conclusions.......................................................................... 42
   8.2 Future directions................................................................... 42
9. SUMMARY.................................................................................... 44
LITERATURE CITED........................................................................ 46
APPENDICES
   Appendix A: Two-dimensional cluster experiments.................. 50
   Appendix B: Three-dimensional cluster experiments................ 54
   Appendix C: Sample run of Clusterability Algorithm: two dimensions.. 57
   Appendix D: Sample run of Clusterability Algorithm: three dimensions... 59
   Appendix E: Code of Clusterability Algorithm: two dimensions.......... 61
   Appendix F: Code of Clusterability Algorithm: three dimensions........... 89
LIST OF FIGURES

Figure 1. K-means clustering................................................................. 7
Figure 2. A subjectively clusterable dataset......................................... 8
Figure 3. Distance-based clustering.................................................... 9
Figure 4. Edge frequency histogram.................................................. 14
Figure 5. Non-properly clusterable dataset.......................................... 16
Figure 6. Perfectly clusterable dataset............................................... 17
Figure 7. Uniform dataset................................................................. 17
Figure 8. Minimums and maximums in the edge frequency histogram...... 26
Figure 9. Adult dataset.................................................................... 30
Figure 10. IPUMS Census dataset....................................................... 31
Figure 11. Iris dataset.................................................................... 31
Figure 12. Data with known number of clusters, two clusters.............. 32
Figure 13. Data with known number of clusters, three clusters.......... 32
Figure 14. IPUMS Census data Histogram.......................................... 34
Figure 15. IPUMS Census data, Clusters: 4........................................... 34
Figure 16. IPUMS Census data, Histogram.......................................... 35
Figure 17. IPUMS Census data, Clusters: 4........................................... 35
Figure 18. Entrée Chicago Recommendation Data, Histogram............... 36
Figure 19. Entrée Chicago Recommendation, Clusters: 5.................... 36
Figure 20. El Nino data1, Histogram.................................................. 37
Figure 21. El Nino data1, Clusters: 2.................................................... 37
Figure 22. El Nino data2, Histogram.................................................. 38
Figure 23. El Nino data2, Clusters: 2.................................................... 38
Figure 24. El Nino data2, Histogram.................................................. 39
Figure 25. El Nino data2, Clusters: 2.................................................... 39
Figure 26. El Nino data3, Histogram.................................................. 40
Figure 27. El Nino data3, Clusters: 2.................................................... 40
Figure 28. Performance evaluation..................................................... 41
ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Mukkai Krishnamoorthy, for his guidance and encouragement throughout my studies at RPI and the course of this thesis.

I would like to thank FUNDACYT my sponsor institution, for believing in me and giving me the opportunity to study in the USA.

Finally, I would like to thank my parents, my brothers and my sister for their encouragement through my life and career.
ABSTRACT

Identifying clusters in a dataset is valuable. Most existing data clustering algorithms need the number of clusters as an input. The present thesis introduces a graphical method that outputs the number of clusters. Once the number of clusters is calculated, other clustering algorithms may use it.

This method is cubic in the number of input data. Given that most databases are extremely big, it is convenient to choose a random sample to make a faster detection of the clusters. However, not only is speed important but the level of confidence in this method is as well. So, the level of accuracy is calculated for each run of the algorithm. Three different sampling algorithms have been used in order to choose the random sample. The efficiency of the three algorithms has been compared based on the number of clusters that they detect and the running time. In addition, depending on the number of points in each cluster, central points or spies that cover the clusters are assigned to each cluster. In order to have a complete vision of the evolution of data clustering and the future tendencies of the clusters, a time series analysis was conducted as well. All the tests have been conducted using datasets from the UCI Machine learning repository and artificially generated datasets. I present experimental results and show the effect of sampling algorithms and the number of clusters.
CHAPTER 1
INTRODUCTION

Several works rely on data clustering. The most widely used algorithm is the well-known k-means algorithm. The k-means algorithm needs as input the number of clusters. The efficiency of the k-means algorithm therefore depends on the initial input. The present thesis describes an algorithm that outputs the number of k clusters and the seed centroids needed in distance-based algorithms. The present algorithm should work in conjunction with distance-based algorithms that need as input the number k of clusters. Furthermore, the present algorithm is not a clustering algorithm but a measure of clusterability since it provides a measure of how successful a distance-based algorithm will be. For example, if the clusterability algorithm finds no clusters in the dataset there would be no need to use a distance-based algorithm, and the processing time would be shorter. On the other hand, if the clusterability algorithm finds more than one cluster, we can be sure that a distance based algorithm will be successful in assigning the points to each cluster.

One of the advantages of the clusterability algorithm is that it uses three different sampling algorithms in order to choose a sample of the data, and these solve the scalability problem. Consequently, from the sample it calculates the number of clusters and the centroids. Every sampling algorithm has been compared to each other in terms of number of clusters and in terms of running time. A detailed comparison of the sampling algorithms will be described later. This algorithm is
based on distance measures, and the two measures used most frequently are the Euclidean and Manhattan distances. In the present work the Manhattan distance is used for all the calculations.

The most innovative aspect of the present work is that the clusterability algorithm uses a graphical approach in order to find the number of clusters. In the past several researchers have used graphical tools to find the clusters; however, the present work differs greatly from those approaches. “The approaches that have been taken so far treat each point as a node in a graph and define adjacency relationship among nodes based on some distance metric” [5]. In the present algorithm, adjacency between two points is inferred through a histogram obtained using the edge lengths between every two points. Furthermore, the algorithm places central points or spies that cover the clusters. In order to prove that the present clusterability detection algorithm is efficient, the efficiency is calculated using maximums and minimums in the edge frequency histogram. In addition, the algorithm was tested with data with known number of clusters, and it proved to find most of the time the correct number of clusters. The output results of the algorithm are textual and graphical. The graphical output shows two graphics: 1) the edge frequency histogram and 2) the clusters found (in order to distinguish among clusters, different markers are used) along with the spies for every cluster. The textual output (see appendices C and D) presents: the number of total points, total edges, total buckets, cutoffs, number of points per cluster, number of clusters, centroids (average of the points in the cluster), centroid-points (closest point to a centroid, used as a spy), spies, level of confidence and running time.
1.1 Motivation

Currently, there is an explosion of information from various sources resulting in a huge amount of data stored in databases. This enormous amount of data contains information that needs to be analyzed. Therefore, research in the field of data mining has increased to keep up with the pace. Data mining is the technique of analyzing data to discover previously unknown information. Machine learning is the base tool used by data mining. Machine learning can be subdivided in supervised and unsupervised learning. This thesis deals primarily with data clustering that is an approach to unsupervised learning.

Clustering is a way of finding natural clusters or groupings of similar data items in large datasets. Then, given the large size of the dataset, the aim of data clustering is to make it easier for humans to understand the relationship between similar data. Applications of clustering are document categorization, pattern recognition, classification, spatial database applications, and web analysis, etc.

1.2 Context of prior work

Data clustering is not a new field. Many clustering algorithms were developed during the 1970s. The enormous amount of data to be clustered nowadays has led researchers to try to develop better clustering algorithms. "The clustering of a dataset really depends on the many different ways to express and formulate the clustering problem; as a consequence, the obtained results and its interpretations depend strongly on the way the clustering problem was originally formulated" [1].
1.3 Contributions

The primary contribution of my work is based on the paper “Detection and Cluster Initialization” by S. Epter, M. Krishnamoorthy, M.J.Zaki. That paper presents the idea of clusterability using an edge frequency histogram. Moreover, the basic idea of the clusterability algorithm comes from that paper. However, in the present thesis I present enhancements and additions to that algorithm.

1.4 Overview of material

In chapter two, I distinguish between clustering algorithms and clusterability detection. Moreover, I explain in detail cluster analysis and clusterability analysis, including a subdivision of clustering algorithms and clusterability algorithms. Then, in chapter three, I discuss related work in the area of clustering and clusterability. I discuss the theory behind my work in chapter four. Chapter five presents a detailed description of the clusterability algorithm. In chapter six, the types of experiments performed are presented. The next chapter presents the results of this research and time complexity. Chapter eight presents conclusions and future work. Finally, chapter nine presents a summary of the thesis.
CHAPTER 2
CLUSTER ANALYSIS AND CLUSTERABILITY

In this chapter a description of cluster analysis and clusterability is defined. The aim of the present clusterability algorithm is to find the k number of clusters needed as input for distance-based clustering algorithms. Therefore, the most used distance-based clustering algorithm, k-means, is described as well.

2.1 Cluster analysis vs. clusterability

Through clustering we group similar items together. By creating groups or clusters we reduce the amount of data needing to be analyzed. In other words, analyzing clusters is easier than analyzing the dataset as a whole. Furthermore, the ultimate goal of clustering is to group items with a high degree of similarity. Clustering has become important currently because datasets tend to be extremely large in size. Therefore, an automated program is needed to find the clusters and also to avoid human mistakes while clustering. On the other hand, clusterability provides a means of knowing whether a clustering algorithm will be successful or not. In other words, through clusterability we can infer whether we should cluster or not. In short, clusterability is a way of knowing if a dataset contains any clusters, if so, what type of clusters it contains.

2.2 Types of clustering

"The clustering of a dataset really depends on the many different ways to express
and formulate the clustering problem; as a consequence, the obtained results and its interpretations depend strongly on the way the clustering problem was originally formulated" [1]. There are several algorithms for data clustering and they can be subdivided in several categories [2]. The subdivision of hierarchical and non-hierarchical algorithms is one of the simplest categories [7].

2.2.1 Hierarchical algorithms

Hierarchical algorithms build a tree of clusters. They can be subdivided into agglomerative and divisive methods. The agglomerative technique consists of merging or agglomerating clusters, and functions as follows. First, a point is a cluster by itself. Then, the most similar pairs of clusters are merged together to form a single cluster. This process repeats until we get the number of clusters required. The divisive technique on the other hand starts with only one big cluster and divides the big cluster into dissimilar clusters. Hierarchical clustering algorithms have been used in many areas of data mining.

2.2.2 Non-hierarchical algorithms

In non-hierarchical clustering a centroid point is first chosen then all points that are closest to a centroid are included in that cluster. The very well known k-means algorithm falls into this category.

2.3 K-means clustering

The k-means algorithm is an iterative algorithm that finds the natural clusters using
a distance-based technique. In other words, 1) it takes each item in the dataset and assigns it to one of the clusters according to which one gives a minimum distance. See figure 1; 2) the new centroid is calculated taking an average of all the items in the cluster. These two steps are repeated until convergence. Therefore, each cluster is represented by the mean of its points, which is most commonly called its centroid.

![Figure 1. K-means clustering](image)

The k-means algorithm, as well as other non-hierarchical algorithms, needs as input the number k of clusters. Usually the user is expected to provide the value for k. This requirement is a drawback of the usage of non-hierarchical algorithms. Moreover, distance-based algorithms are based on the assumption that the input data is numerical. As a consequence, distance-based algorithms are being modified in order to cope with 1) different types of attributes (for example, images have color, shape, texture, etc.); 2) high dimensional data (or in other words, data items with hundreds of attributes); 3) outliers or noise.
2.4 Clusterability

2.4.1 Graph-based methods vs. non-graphic methods

Depending on the algorithm that we choose, the clusters found will be different. Figure 2 illustrates this point, taken from [5]. As we see in figure 2, a spatial method groups x, w and y, z in two different clusters. On the other hand, a distance-based algorithm will group points x, y and points w, z in two different clusters.

![A subjectively clusterable dataset](image)

*Figure 2. A subjectively clusterable dataset*

Distance-based algorithms are based on the assumption that data points in one cluster are more similar to one another and that data points in separate clusters are more dissimilar to each other. In other words, “the objects are clustered or grouped based on the principle of maximizing the inter-class similarity and minimizing the intra-class similarity” [4]. For an illustration of this idea see figure 3, taken from [33].
Figure 3. Distance-based clustering

As stated previously this thesis presents a clusterability algorithm, one that is not meant to replace current distance-based algorithms but to supplement them before clustering.
CHAPTER 3

RELATED WORK

This chapter presents recent trends in clustering algorithms and work in clusterability. Several algorithms have focused on clustering data that is, finding the natural clusters in a dataset. However, few have approached an analysis of clusterability. By clusterability, I mean an analysis of the possible outcomes of a clustering algorithm, and an analysis of how many and which k clusters are found in a dataset. In short, an analysis that gives as output the k clusters needed in distance-based clustering algorithms.

3.1 Recent trends in clustering algorithms

Several new clustering algorithms have been developed recently. Among the new algorithms that have emerged lately are [8]:

CLARANS (Clustering Large Applications Based on RANdomized Search) [9] is a spatial data-mining algorithm. Moreover, it is a k-medoid algorithm. A medoid is a centrally located data point in a group. The algorithm uses a randomized search to find the medoids that represent the clusters. However, one of the drawbacks of CLARANS is its inefficiency; it takes at least quadratic time to find the medoids.

DBSCAN (Density Based Spatial Clustering of Applications with Noise) is another recent algorithm [10]. It finds the clusters according to a density-based notion of cluster. In this scheme a cluster is represented as a group of k density-connected
points. One of the key points of this algorithm is that it finds clusters of arbitrary shape even in the presence of noise.

**DBCLASD** (Distribution Based Clustering of Large Spatial Databases) [11] is a locality-based algorithm. The difference with DBSCAN is that it assumes that the points in each cluster are randomly distributed. Moreover, it detects clusters of arbitrary shape and does not require input parameters.

**STING** (Statistical Information Grid-based method) [12] uses index structures. It uses a grid-based method for dividing the spatial area into rectangular cells. Then, it finds clusters as the density-connected components of the grid.

**BIRCH** (Balanced Iterative Reducing and Clustering using Hierarchies) [13] in order to achieve scalability uses a hierarchical data structure to cluster the data points. However, according to [2], hierarchical clustering of large databases can be very sub-optimal, even if data fits in memory. Moreover, BIRCH does not perform well if the clusters are not spherical.

**WaveCluster** [15] uses a grid in order to find clusters. “It considers the multidimensional spatial data as a multidimensional signal and it applies signal processing techniques to convert the spatial data into frequency domain.” [15].

**DENCLUE** [16] (Density-based clustering) merges partitioning, locality-based, and hierarchical or grid-based approaches. “It models the overall point density analytically as the sum of influence functions of the data points. Clusters can then be identified by determining density-attractors”.

**CLIQUE** [17] (Clustering in Quest) uses a density and grid-based approach to find subspace clusters in high dimensional data.
CURE (Clustering Using Representatives) [18] chooses a group of points to identify the distance between clusters. In general, it uses a bottom-up hierarchical clustering approach.

These are some of the most prominent algorithms developed lately, used in clustering. I next review related work in clusterability.

3.2 Work in clusterability

None of the algorithms discussed before, focuses on the problem of providing a solution to the input of k clusters necessary in distance-based algorithms. The research in this area is limited; among the works aimed to analyze cluster separation for different k are the following [2]:

Engleman and Hartigan [19] proposed: “The distance between any two centroids (medoids) normalized by corresponding cluster’s radii (standard deviations) and averaged (with cluster weights) is a reasonable choice of coefficient separation.”

Kaufmann and Rousseeuw [20] developed the Silhouette coefficient which “considers the average distance between the point x of cluster C and other points within C and compares it with averaged distance to the best fitting cluster G other than C.” The Silhouette coefficient ranks values close to +1 as good clustering choices and negative values as bad clustering choices.

Bezdek [21] proposed the Partition coefficient. “It is equal to the sum of squares of the weights. Each of the measures can be plotted as a function of k and the graph can be used to choose the best k”.
These three methods can be used to find or a approximate k; however, the method proposed in this thesis not only is used to find a good value for k, but also provides a measure of clusterability; in other words, a way of knowing a priory whether to cluster or not.

In [31], [32] a method to refine k is described. However, the initial input value k of clusters is expected from the user. In contrast, the clusterability algorithm presented in this thesis aims to produce a value for k. Moreover, this method indicates the clusterability of the data even before working with the clusters. One point that the clusterability algorithm presented in this thesis shares with [31], [32] is scalability. The clusterability algorithm uses three different sampling algorithms in order to work with a sample of the database. In [31], [32] they work with small samples of the dataset.

In light of the above descriptions of how to find a good value of k, the clusterability algorithm presented in this thesis so far is the only method that uses a graphical approach and also “determines the adjacency between two nodes not directly but indirectly by obtaining a histogram of all pair edge lengths” [5]. Through the edge frequency histogram, the cutoffs used to place each point in its respective cluster are determined.
4.1 Notations and definitions

I use the concepts and notation described in [5].

In order to find which points belong to which cluster, an edge $E$ or distance between two points $a, b$ is calculated. To measure the distance, I used the Manhattan distance. The idea is to form an edge frequency histogram that provides us with minimums that represent cutoffs for the clusters, or in other words, the points that lie between minimums form a cluster. In figure 4 the cutoffs in this case are: 1, 5, 7 and 9. Therefore the points that lie between 0-1, 1-5, 5-7, and 7-9 form four clusters.

![Figure 4. Edge frequency histogram](image)

In order to calculate which points belong to which cluster, I selected a reference point.
The reference point in this case is the first point in the dataset. Using this reference point, \( r \), I calculate the edges between \( r \) and every single point \( s \): \( \Delta_{r,s} \).

The size of the edge determines the cluster to which a point belongs. For example, in figure 4 there are four clusters: cluster 1: 0-1, cluster 2: 1-5, cluster 3: 5-7, cluster 4: 7-9. Then, let’s say that the edge between the reference point \( r \) and a point \( s \) falls between cutoffs 1-5 then, point \( s \) belongs to cluster 2. In this way all points inside of cluster 2 will have edges \( \Delta_{r,s} \) longer than 1 and smaller than 5. In other words, all points that belong to a cluster will have edges longer than the previous cutoff and smaller than the next cutoff.

\[
E_{r,si} = \text{cluster } i \text{ if } c_i \leq \Delta_{r,si} < c_{i+1}
\]

The centroid for the clusters is determined by averaging all points in a cluster. All edges inside of a cluster are considered as internal edges and in the same fashion all edges outside a cluster are external edges.

### 4.2 Types of clusters

Let’s illustrate the three classes of clusterability: Non-properly clusterable dataset, properly clusterable dataset and perfectly clusterable dataset. The three classes are mathematically precise. The edge frequency histogram gives us a good visual prediction of the three classes of clusterability. Let’s give some examples of datasets and histograms that support this claim.

#### 4.2.1 Non-properly clusterable dataset

Figure 5 shows a non-properly clusterable dataset. As it can be seen in the dataset
there is not really a way to separate the data into different clusters. This is supported by the edge frequency histogram that reveals no relevant maximums and minimums. Consequently, the dataset has low clusterability.

![Histogram](image1)

*Figure 5. Non-properly clusterable dataset*

**4.2.2 Perfectly clusterable dataset**

Figure 6 shows a perfectly clusterable dataset. From the edge frequency histogram we can visually confirm that the dataset contains 2 clusters that are defined by the minimums and maximums or in other words by the clear cut-offs in the graph.
4.2.3 Uniform dataset

If we compare the edge frequency histogram of figure 5 and figure 7 we can see the similarity between both of them because there are no clear clusters in the dataset.

From the figures we can infer that a k-means algorithm will be successful in finding the clusters if the edge histogram looks like figure 6 and unsuccessful if it looks like
figure 5. In short, the aim of this algorithm is to find or approximate good cutoff points using the edge frequency histogram.
CHAPTER 5

CLUSTERABILITY ALGORITHM

In chapter 3, I reviewed the algorithms developed for data clustering and clusterability. Now, I will present the clusterability algorithm that is the topic of this thesis. It is based on two main parts: edge frequency histogram and cutoffs. I will give a detailed description of the random-sampling algorithms, the edge frequency histogram, calculation of cut-offs, calculation of centroids, calculation of number of spies, time series, and calculation of level of confidence.

5.1 Design goals

The goal of my work has been to design an algorithm that is capable of coping with high dimensionality and at the same time be a fast algorithm. Moreover, this algorithm takes advantage of a graphical approach: edge frequency histogram. This is a complete new approach to clusterability. As stated before, distance-based algorithms expect as input the value for the k clusters, so the aim of this algorithm is to provide as output the number of k clusters. Moreover, using a sampling technique we can visually see the clusters found in a sample of the dataset. The clusterability algorithm does not handle outliers, it needs as an input a two- or three-dimensional vector representing the x-, y- and z-axis, it only accepts numerical data, and it does scale to n number of points. The datasets used to test the algorithm are datasets from the UCI Machine learning repository and artificially generated datasets.
5.2 Random sampling algorithms

The number of points that databases handle nowadays grows constantly as the information collected from users grows. Therefore, an efficient algorithm should keep this issue in mind. Consequently, I tested the clustering algorithm in a small sample of the database. This solves the scalability problem. In order to select the sample, I have implemented three different random sampling algorithms: Algorithm R [22], Algorithm S [22] and Random Algorithm. I slightly modified algorithms R and S in order to make them suitable for my needs. However, the main idea of both algorithms is preserved.

The three algorithms chosen are valid algorithms. In the case of Algorithm S, “the sample is completely unbiased. The probability that any given element is selected, such as the last element of the file is n/N (n desired number of points, N total points)” [22]. Algorithm R is very useful when the total size of the database is unknown. Random Algorithm is a simple algorithm that through testing has proved to be efficient enough to be used as part of this clusterability algorithm.

_Brief description of algorithm R:_

**Input:** n number of sample records

**Output:** a random subset with size equal to number of sample records

1 Copy the first n records in matrixX, and matrixY, \( t = n \)

2 While there are more records in the database
   
   Increase \( t \) by 1. Generate a random number \( M \) between 1 and \( t \). If \( M > n \) skip
   
   the next record

20
If $M < n$ replace the record in matrix $X$ and matrix $Y$ corresponding to position $M$

*Brief description of algorithm $S$:*

*Input:* N number of total points

$n$ number of sample records

*Output:* a random subset with size equal to number of sample records

1. Assign values to the variables $t$ and $m$.
   
   $t = 0$ (total number of input records)

   $m = 0$ (points selected so far)

2. Generate a random number $U$ between zero and one

3. If $(N-t) \geq n - m$ skip the next record and increase $t$ by 1. Otherwise select the next record and store it in matrix $X$ and matrix $Y$

*Brief description of Random Algorithm:*

*Input:* N number of total points

$n$ number of sample records

*Output:* a random subset with size equal to $n$

1. Calculate random maximum number: $N/n$

2. While total points chosen so far $m < n$
   
   2. 1. Generate a random number between 1 and random maximum number
2.2 Traverse the dataset sequentially choosing the next $x$, $y$ number according to the random number generated and store them in matrix X and matrix Y.

Using several datasets from the UCI Machine learning repository the three algorithms were tested twenty times per dataset. The data collected per algorithm was stored in tables. (See Appendices A and B)

I sampled 10% of the data. From these tests I can conclude: With respect to the running time Algorithm S always runs faster than the other two algorithms. Especially on very large databases the running time for algorithm S is almost half of the running time than the other two algorithms. Running times of algorithm R and Random Algorithm are almost the same. Each algorithm behaves in a different way depending on the database and the number of points in each database. However, I can say that in general Algorithm R and Random Algorithm find more clusters than Algorithm S. Consequently, even though the running time of Algorithm S is better than the other two running times I prefer to use Algorithm R or Random Algorithm in order to ensure a better sample in terms of number of clusters.

5.3 Edge frequency histogram

The backbone of the clusterability algorithm relies on the edge frequency histogram. As stated before, the histogram represents the distances or edges between the reference point and all the other points in the dataset. Moreover, from
the edge frequency histogram I can infer if the dataset shows a high, medium or low level of clusterability.

In order to construct the histogram, the user has to input the number of buckets. If the number of buckets is too large then the histogram shows empty buckets. However, ten seems to work fine with all datasets. Once I have the sample and the number of buckets, all the edges with respect to the reference point \( r \) are calculated. A dataset with size \( n \) has \( \frac{n^2 - n}{2} \) edges.

Then, each edge is normalized according to the maximum edge length in the set. The normalized value falls between a range from \([0...1]\). In order to find out to which bucket this value belongs to I additionally multiply this value by the total number of buckets \( b \) and finally round off this number. This number corresponding to an edge falls into a specific bucket increasing a counter \( l \) for bucket \( i \). Finally, the counter for each bucket gives the frequency of occurrence of edge lengths. The \( x \)-axis of the histogram corresponds to the numbers of buckets that range from \([0...b-1]\), and the \( y \)-axis corresponds to the counter of each bucket.

### 5.4 Cut-offs

Once I have the histogram I scan it in order to detect the type of clusterability: high clusterability, low clusterability and uniformity. The presence of large maximums and large minimums indicates high clusterability. The choice of cut-offs is easy: it corresponds to the minimums. Furthermore, if there are several small maximums and several small minimums I can say that the data presents low clusterability. Moreover, if the gap between maximums and minimums is small, then we are
dealing with a dataset that presents uniformity. In the last two cases, the cut-off has
to be approximated. “There are several means of approximating c in the absence of
a pronounced first spike. One method is to take c as some fraction of the bucket
that represents the highest maximum value. Another method is to fix c as some
small percentage of the total number of buckets” [5].

5.5 Centroids

A multidimensional array is needed to represent the points and the clusters. Every
dimension of the matrix corresponds to each cluster. So, the dimensions of the
matrix vary according to the clusters found. Moreover, each dimension of the
matrix stores the points that correspond to points that belong to the cluster. In other
words, points that share: c_i ≤ Δ_{r,s}i ≤ c_i+1. Using this matrix of adjacent points I can
average all the points corresponding to a cluster and therefore calculate the centroid
of each cluster.

5.6 Spies

Central points or spies that cover the clusters are assigned to each cluster. The
decision of how many spies to assign to each cluster is difficult to make. If the
shape of the cluster would be known, I could assign spies according to the
geometrical shape. For example, if the clusters have the shape of a triangle, the
spies would be located in each vertex of the triangle. However, the shape of the
cluster is unknown to us. Assuming a geometric shape for a cluster I assign a
maximum of five spies: four spies located at four “extreme” vertexes and one in the centroid of the shape.

The number of spies that I assign to each cluster depends on the number of points that belong to each cluster and the number of clusters. Of course, each cluster at least will have one spy and a maximum of 5 spies. Then, I use the following relation to calculate the number of spies:

\[ \text{Spies} = \text{round}\left[\frac{\text{Total points in cluster}}{\text{Total points in dataset/number of clusters}}\right] \]

In order to place the spies I choose for the vertexes of the shape, the highest leftmost, the highest rightmost, the lowest leftmost, the lowest rightmost points and the centroid of the cluster.

### 5.7 Time Series

Let’s first define time series. They are sequences of measurements taken at equally spaced time intervals.

In the clusterability algorithm two possible approaches are taken:

- a) Take two attributes that constitute the \(x\) and \(y\) coordinates

- b) Take two attributes that constitute the \(x\) and \(y\) coordinates and a third attribute, time (corresponding to the time series).

In case \(a\), attributes are plotted in two dimensions. Obviously, in case \(b\), the attributes are plotted in three dimensions.

The idea behind using time as the third attribute is to find the different clusters in time.
5.8 Level of confidence

It is important to define a certain measure of confidence, in other words, a measure of accuracy of the algorithm. In order to calculate the confidence I have chosen to compare the minimums with the maximum points in the histogram. Vertical lines in figure 8 represent the minimums and maximums.

From the same figure we can see that the minimums are 20444, 17844, 1851 and 3555, and the maximums are 45244, 35426, 21790 and 16539. I first divide each minimum by the correspondent maximum and so on. I then calculate the sum of these divisions and their average. I multiply the average by 100 to get a percentage of the level of confidence. The problem with this approach is that if the minimum is zero, I do not take into account that division, (We can not divide a number by zero) then this distorts the level of confidence.

![Figure 8. Minimums and maximums in the edge frequency histogram](image)

5.9 Formal description of the clusterability algorithm

The algorithm has three main parts: In step one based on the algorithm selection I create a sample of points. In step two I calculate and generate the edge frequency
histogram. In step three I examine the histogram and try to find or approximate the
cutoffs, then I create a high dimensional matrix with the adjacent points
corresponding to each cluster, and from the adjacency matrix I compute the average
of all the points in the cluster and generate their centroids. Finally, I calculate the
minimums and maximums and output the level of confidence along with a graph
that contains all clusters and spies.

**Clusterability Algorithm**

**Input:** number of buckets b

Dataset D

Total number of points in D

Total number of sample records n

Sampling algorithm (1, 2, 3)

**Output:** number of k clusters, centroids, spies, level of confidence, running time

1 Using the sampling algorithm chosen by the user select a sample of D

2 Find all edge lengths including Emax (largest edge in the sample)
   for each length: find normalized round index
   increment bucket counter for index

3 Plot a histogram using the total number of edge frequencies in the y-axis and the
   bucket number – 1 in the x-axis

4 Scan histogram
   find or approximate c

27
output clusterability

generate adjacency matrix

find connected components

for each connected component output centroid

for each cluster output spies

find maximums and minimums

output level of confidence

output running time

5 Plot the clusters with their respective spies
CHAPTER 6

EXPERIMENTS

I tested the accuracy of the algorithm using the entire dataset if the dataset was smaller than 3000 points. Otherwise, I used a sample of the dataset. The size of the sample depends on the desired size provided by the user.

All the figures in this chapter illustrate 1) the histogram and 2) the clusters (different markers were used to differentiate among clusters).

Figures 9-11 correspond to datasets from the UCI Machine learning repository.

Figures 12 and 13 correspond to artificially generated datasets with known number of clusters. From the datasets used in figures 9-13 two attributes were used as x- and y-axis giving a two-dimensional plot of the clusters.

Figures 14-27 correspond to datasets from the UCI Machine learning repository. From the datasets used in figures 14-27 two attributes correspond to the x- and y-axis and time corresponds to the z coordinate.

6.1 Two dimensional data

Two attributes are taken into account to represent the x and y coordinates of the points to cluster.

6.1.1 Samples

Samples were used if the dataset was bigger than 3000 points. The use of samples aims to achieve scalability.
In figure 9, the cutoffs are 0, 3, 6, 9, so the clusterability algorithm finds three different clusters. Cluster one corresponds to the points that lie between cutoffs 0 and 3. Cluster two corresponds to the points that lie between cutoffs 3 and 6. Cluster three corresponds to the points that lie between cutoffs 6 and 9. In this histogram we can see that there are no pronounced spikes therefore, the clusters found are close to each other.

**Figure 9. Adult dataset**

Clusters: 3

In figure 10, the cutoffs are 1, 5, 7, 9, so the clusterability algorithm finds four different clusters. Cluster one corresponds to the points that lie between 0 and cutoff 1. Cluster two corresponds to the points that lie between cutoffs 1 and 5. Cluster three corresponds to the points that lie between cutoffs 5 and 7. Cluster four corresponds to the points that lie between cutoffs 7 and 9. In this histogram we can see that there are no pronounced spikes therefore, the clusters found are close to each other.
In figure 11, the cutoffs are 0, 4, 9, so the clusterability algorithm finds two different clusters. Cluster one corresponds to the points that lie between cutoffs 0 and 4. Cluster two corresponds to the points that lie between cutoffs 4 and 9. In this histogram we can see that there is a pronounced spike, therefore the clusters found are separate from each other.
6.1.2 Known number of clusters

I used artificially generated datasets with known number of clusters in order to find out if the algorithm is able to find the accurate number of clusters. From the tests performed it has been shown that it does find the correct number of clusters.

In figures 12 and 13 there are pronounced spikes, therefore there is a clear separation between clusters as we can reconfirm by scanning the figures.

**Figure 12. Data with known number of clusters**

**Figure 13. Data with known number of clusters**
6.2 Time Series

Three-dimensional plotting of the clusters according to two attributes, $x$, $y$ coordinates, and time, $z$ coordinate. Three datasets were used: IPUMS Census data, Entrée Chicago Recommendation and El Nino dataset (by using different pairs of attributes, $x$, $y$, from El Nino dataset, several samples of the dataset were constructed).

Figures 14-19 do not show very pronounced spikes. Therefore, the plot shows clusters that are very close to each other. On the other hand, in figures 20-27 the spikes of the histograms are very pronounced giving as a result clusters that are easy to differentiate because they are separate from each other. Then, figures 20-27 below show perfectly clusterable datasets.
Figure 14. IPUMS Census data, Histogram

Figure 15. IPUMS Census data, Clusters: 4

34
Figure 16. IPUMS Census data, Histogram

Figure 17. IPUMS Census data, Clusters: 4
Figure 18. Entrée Chicago Recommendation Data, Histogram

Figure 19. Entrée Chicago Recommendation Data, Clusters: 5
Figure 20. El Nino data1, Histogram

Figure 21. El Nino data1, Clusters: 2
Figure 22. El Nino data2, Histogram

Figure 23 El Nino data2, Clusters: 2
Figure 24. El Nino data2, Histogram

Figure 25. El Nino data2, Clusters: 2
Figure 26. El Nino data3, Histogram

Figure 27 El Nino data3, Clusters: 2
CHAPTER 7
RESULTS AND PERFORMANCE ANALYSIS

I tested the algorithm using the UCI Data Mining repository and artificially generated datasets. The metric used was the Manhattan distance. In order to achieve scalability, the maximum number of points that form the sample are 3000. All tests were performed at a Sun Microsystems computer running Unix. I tested the data using several datasets. The total number of points in each dataset ranges from 140 to 46169.

7.1 Time complexity
In terms of running time, if I choose the same number of points for the sample, then the clusterability algorithm runs in approximately the same amount of time. Then, to evaluate the performance I chose the “Internet Usage Data” dataset and ran it with total number of points going from [50-3000]. As we can see in figure 28 the running time increases linearly.

![Graph](image)

*Figure 28. Performance evaluation*
CHAPTER 8
CONCLUSIONS AND FUTURE DIRECCIONS

8.1 Conclusions

Clusterability detection is aimed towards discovering how successful a distance-based algorithm will be when trying to cluster the points in a dataset. Moreover, this graphical method outputs the number of k clusters that constitutes the input of distance-based algorithms. The aim of this algorithm is not to replace clustering algorithms but to work in conjunction with them in order to improve their efficiency by providing them with a good seed of centroids. Furthermore, it outputs the level of confidence and plots spies that cover each cluster. Moreover, time series has been taken into consideration. In other words, in addition to choosing two attributes, time is also considered as a third attribute. Using this three-dimensional graph approach it was shown that the clustering algorithm was able to find the clusters in the dataset. The algorithm is scalable since it uses a sample of the data. The algorithm runs in linear time. One drawback is that it does not handle outliers or noise.

8.2 Future directions

Future improvements to the algorithm will be making it less sensitive to noise. In addition, right now, it only handles numerical data. It would be a great improvement if it would be able to handle any kind of data such as alphanumerical data, graphical data, etc. The algorithm is not suited for high dimensionality data.
So, making this algorithm handle high dimensionality will also improve its usage. Furthermore, the efficiency in terms of cluster detection depends on how well the sample represents the data.

The algorithm right now does take into account three-dimensional clusters by using time series. Time series was included in the clusterability algorithm because today, many of the data stored in datasets corresponds to time series. However, in a more general way, the data represents a stream. "A stream is an ordered sequence of records. Analysis of this data requires stream-processing techniques, which differ in significant ways from what current database query languages and statistical analysis tools support today. There is a real need for better stream data analysis systems" [14]. Therefore, using the clusterability algorithm to in some way analyze data streams would be another future improvement to the algorithm.
CHAPTER 9

SUMMARY

The present and future tendencies in technology demand a fast, reliable and easy way to analyze data. Moreover, the amount of data to be analyzed is growing exponentially; therefore research has turned its focus on trying to solve this problem. One way to cope with this issue is to cluster the data in order to get some information out of it. In this thesis, not a clustering algorithm but a clusterability algorithm was presented. The difference between both of them relies in that a clusterability algorithm focuses on finding if a dataset has clusters. Moreover, a clusterability algorithm determines if a clustering algorithm will succeed in clustering the data. In short, even before trying to cluster using a clustering algorithm a clusterability algorithm should be used. In other words, both of them should work in conjunction instead of replacing each other.

The clusterability algorithm developed uses distance-based and graphic techniques. Therefore, its basis relies in creating a histogram of distances between all points. Using this histogram, cutoffs can be calculated. Once a cutoff is discovered all points that lie between cutoffs are considered to be in the same cluster.

Moreover, in order to achieve scalability the algorithm chooses a sample of the data if the database exceeds 3000 points. In order to obtain the sample, three different sampling algorithms have been implemented. Algorithm S and Random Algorithm need as input the total number of points in the database. Among these three Algorithm S is the fastest one, but Random Algorithm and Algorithm R are more
efficient in terms of number of clusters found than Algorithm S. Finally, algorithm R is the slowest one, but it has the advantage that it can be used when the number of points in the database is unknown.

Two clustering approaches were developed. In the first one, two attributes of the database form a two-dimensional cluster graph using the attributes as the $x$ and $y$ coordinates for the points. In the same fashion, two attributes of the dataset and the time that the measure was taken were used to produce a three-dimensional cluster graph. Using time as the $z$ coordinate we see how different clusters are found in time.

For both, the two-dimensional and three-dimensional approaches spies are designated for each cluster. According to the number of points in each cluster, the number of clusters and a comparison between each cluster’s size 1, 2, 3, 4 or 5 spies are assigned for each cluster. The spies in a cluster are placed using a geometric shape in order to be able to cover each point in the cluster.

The clusterability algorithm has as output two graphs: edge frequency histogram and clusters. The output also includes the cut-offs, centroids, number of points in each cluster, spies, level of confidence and running time. The level of confidence is a measure of the precision of our calculations and the accuracy of the clusters. Using several experiments with different datasets we found out that the running time increases in a linear way. Therefore, the algorithm achieves scalability. Moreover, by using datasets with known number of clusters, we proved that the clustering algorithm is able to identify them accurately.
LITERATURE CITED


[33] V. Kumar, M. Joshi, High Performance Data Mining, University of Minnesota, 1999.
APPENDIX A

Two-dimensional cluster experiments

In the following tables experiments using Algorithm R, Algorithm S and Random Algorithm for two-dimensional cluster analysis are presented. The purpose of the experiments was to compare the running time and the number of clusters that each sampling algorithm finds. Each algorithm was tested 20 times.

Database used: 6cluster.dat(Artificially generated dataset)

Total points in the database 404

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>80.85</td>
<td>67.4</td>
<td>78.55</td>
</tr>
<tr>
<td>(milliseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total points</td>
<td>39</td>
<td>20.4 (Average)</td>
<td>39</td>
</tr>
<tr>
<td>1 cluster</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 clusters</td>
<td>8</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>3 clusters</td>
<td>8</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>4 clusters</td>
<td>2</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>5 clusters</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Database used: IPUMS Census data (UCI database repository)**

Total points in the database 6553

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>2678.35</td>
<td>1724.5</td>
<td>2409.45</td>
</tr>
<tr>
<td>(milliseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total points</td>
<td>650</td>
<td>331.1 (Average)</td>
<td>650</td>
</tr>
<tr>
<td>1 cluster</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 clusters</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3 clusters</td>
<td>10</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4 clusters</td>
<td>10</td>
<td>17</td>
<td>14</td>
</tr>
</tbody>
</table>

**Database used: Internet Usage Data (UCI database repository)**

Total points in the database 10104

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>4239.5</td>
<td>2062.8</td>
<td>4216</td>
</tr>
<tr>
<td>(milliseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total points</td>
<td>1000</td>
<td>486.8 (Average)</td>
<td>1000</td>
</tr>
<tr>
<td>1 cluster</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 clusters</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3 clusters</td>
<td>16</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>4 clusters</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Database used: COIL DATA (UCI database repository)

Total points in the database 40

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>202.1</td>
<td>166.9</td>
<td>188</td>
</tr>
<tr>
<td>(milliseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total points</td>
<td>40</td>
<td>20.85 (Average)</td>
<td>40</td>
</tr>
<tr>
<td>1 cluster</td>
<td>3</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>2 clusters</td>
<td>15</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>3 clusters</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>4 clusters</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Database used: Results data (Artificially generated data)

Total points in the database 40

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>178.65</td>
<td>150.65</td>
<td>169.45</td>
</tr>
<tr>
<td>(milliseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total points</td>
<td>40</td>
<td>21 (Average)</td>
<td>34.7 (Average)</td>
</tr>
<tr>
<td>1 cluster</td>
<td>7</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2 clusters</td>
<td>10</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>3 clusters</td>
<td>3</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>4 clusters</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Database used: 140eval.dat (Artificially generated data)

Total points in the database 40

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time (milliseconds)</td>
<td>144.45</td>
<td>124.25</td>
<td>146.75</td>
</tr>
<tr>
<td>Total points</td>
<td>40</td>
<td>21.4 (Average)</td>
<td>34.5 (Average)</td>
</tr>
<tr>
<td>1 cluster</td>
<td>6</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2 clusters</td>
<td>12</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>3 clusters</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>4 clusters</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

From the above tables it can be inferred that Algorithm S is definitely the fastest algorithm between the three algorithms presented. Which algorithm is more efficient in terms of number of clusters found really depends in each unique dataset.
APPENDIX B

Three-dimensional cluster experiments

In the following tables experiments using Algorithm R, Algorithm S and Random Algorithm for three-dimensional cluster analysis are presented. The purpose of the experiments was to compare the running time and the number of clusters that each sampling algorithm finds. Each algorithm was tested 20 times.

*Database used: IPUMS Census data (UCI database repository)*

Total points in the database 22882

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>7185.5</td>
<td>3096</td>
<td>7230.25</td>
</tr>
<tr>
<td>(milliseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total points</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>1 cluster</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 clusters</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3 clusters</td>
<td>8</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>4 clusters</td>
<td>12</td>
<td>9</td>
<td>11</td>
</tr>
</tbody>
</table>
**Database used: Entrée Chicago Recommendation Data (UCI database repository)**

Total points in the database 50672

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>6897.25</td>
<td>3178.25</td>
<td>6867</td>
</tr>
<tr>
<td>(milliseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total points</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>1 cluster</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 clusters</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3 clusters</td>
<td>8</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4 clusters</td>
<td>10</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>5 clusters</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

**Database used: El Nino Data (UCI database repository)**

Total points in the database 46169

<table>
<thead>
<tr>
<th></th>
<th>Algorithm R</th>
<th>Algorithm S</th>
<th>Random Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>7084.67</td>
<td>3110</td>
<td>7001.66</td>
</tr>
<tr>
<td>(milliseconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total points</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>1 cluster</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 clusters</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3 clusters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 clusters</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

55
IPUMS Census Data

In this case all the algorithms find more or least the same number of clusters. However, the running time of Algorithm S is much better than the other running times.

Entrée Chicago Recommendation Data

Algorithm S finds the largest number of clusters. Moreover, it has the best running time.

El Nino Data

In this case all the algorithms find the same number of clusters and the best running time belongs to Algorithm S.
APPENDIX C
Sample run of Clusterability Algorithm: two dimensions

******************************************************************************
*****CLUSTERABILITY ALGORITHM*******

Sampling Algorithm: Algorithm R

Reference point: X: 71.0 Y: 999.0
Total points 2000
Total edges 1999000
Buckets: 10

Cutoff 1: 1.0
Cutoff 2: 5.0
Cutoff 3: 7.0
Total cutoffs: 3

Total points in cluster: 490
Total points in cluster: 869
Total points in cluster: 402
Total points in cluster: 238

Centroid X: 43.634693 Y: 998.17145 (Centroid: Average of the points in the cluster)
Centroid X: 23.10587 Y: 833.7008
Centroid X: 36.415424 Y: 361.86566
Centroid X: 41.27731 Y: 77.22269

Centroidpoint X: 43.0 Y: 999.0 (Centroid point: closest point to the centroid)
Centroidpoint X: 17.0 Y: 820.0
Centroidpoint X: 36.0 Y: 350.0
Centroidpoint X: 43.0 Y: 78.0

57
Spies: 1.0
Spies: 2.0
Spies: 1.0
Spies: 1.0

Spies in cluster:
X: 43.634693 Y: 998.17145

Spies in cluster:
X: 0.0 Y: 999.0 X: 1.0 Y: 999.0

Spies in cluster:
X: 36.415424 Y: 361.86566

Spies in cluster:
X: 41.27731 Y: 77.22269

Total clusters: 4

Level of confidence: 75.05170387426973%

Running time: 7320 milliseconds
APPENDIX D
Sample run of Clusterability Algorithm: three dimensions

*******************************************************************************
******CLUSTERABILITY ALGORITHM******

Sampling Algorithm: Algorithm R

Reference point: X: 53.0 Y: 470.0 Z: 99.0
Total points 2000
Total edges 1999000
Buckets: 10

Cutoff 1: 1.0
Cutoff 2: 5.0
Cutoff 3: 7.0
Total cutoffs: 3

Total points in cluster 107
Total points in cluster 1098
Total points in cluster 794
Total points in cluster 0.

(Centroid: Average of the points in the cluster)
Centroid X: 41.757008 Y: 487.80374 Z: 97.90654
Centroid X: 38.726776 Y: 479.9754 Z: 97.96722
Centroid X: 23.27078 Y: 998.7078 Z: 97.96222

(Centroid point: closest point to the centroid)
Centroidpoint X: 42.0 Y: 490.0 Z: 97.0
Centroidpoint X: 32.0 Y: 514.0 Z: 97.0
Centroidpoint X: 23.0 Y: 999.0 Z: 98.0
Spies:1.0
Spies:2.0
Spies:1.0

Spies in cluster:
X: 41.757008 Y: 487.80374 Z: 97.90654

Spies in cluster:
X: 15.0 Y: 690.0 Z: 97.0
X: 15.0 Y: 930.0 Z: 97.0

Spies in cluster:
X: 23.27078 Y: 998.7078 Z: 97.96222

Total clusters: 3

Level of confidence: 86.63287113784918%

Running time: 8205 milliseconds
APPENDIX E

Code of Clusterability Algorithm: two dimensions

import java.io.*;
import java.lang.*;
import java.util.*;

public class RData {
    public static double [] bucketcounter;
    public static void main(String[] args) throws IOException {
        String nameoffile = (args[1]);
        int sizeoffile = Integer.parseInt(args[2]);
        int numberofpoints = Integer.parseInt(args[3]);
        int choice = Integer.parseInt(args[4]); //Sampling algorithm 1, 2, 3

        char typeX [];
        char typeY [];
        double [] Dnormbucket; //normalized bucket counter
        double [] min;
        double [] max;
        double temp5;
        double temp6;
        double temp7;
        double [] levelconfidence;
        int t = 0;
        int j = 0;
        int i;
        int totalclusters = 0;
int maxclusters = 20;
int pointsperc = 0;
int totalpoints = 0;
int buckets = Integer.parseInt(args[0]);
int bucketsaux;
int totaledges;
int counter = 0;
int aux;
int aux2;
int aux3 = 0;
int upperbound;
int [] allspiescounter;
int [] adjacencycounter;
int [] I; //algorithm R
int b = 0;
int flagsign = 0;
int flag = 0;
int sizeflag = 0;
int sizeflagaux;
int countmin;
int countmax;
int countcutoffs = 0;
int countcutoffs2 = 0;
int tmpaux2;
int tmpaux3;
int tmpaux1;
int newtmp;
int newtmp2;
float Emax = 0; //maximum Edge used for normalizing the bucket counter
float Xmax = 0;
float Ymin = 10000;
float centroiddistance = 1000000;
float Ymax = 0;
float X1 = 0;
float Y1 = 0;
float [] x;
float [] y;
float [] D;       //edge matrix
float [] cutoff;
float [] edgecentroids;
float [] centroidsX;       //Centroid: average of all points in a cluster
float [] centroidsY;
float [] centroidpointX;   //Point closest to a centroid used in spies
float [] centroidpointY;
float [] centroidpointZ;
float [] adjacencyXsum;
float [] adjacencyYsum;
float [][] adjacencyX;
float [][] adjacencyY;
float [] spies;
float [][] allspiesX;
float [][] allspiesY;
float [][] plotspiesX;
float [][] plotspiesY;
long time1 = System.currentTimeMillis();
long time2;
long time3;
long time4;
long time5;
long time6;
long runningtime;       //does not include the time to plot the graphs

int mynumber2 =0;       //Algorithm R
float mynumber1 = 0;     //Algorithm S
File fp;
String buf, str;
StringTokenizer st;

x = new float[numberofpoints];
y = new float[numberofpoints];
typeX = new char[numberofpoints];
typeY = new char[numberofpoints];
cutoff = new float[maxclusters];  //maximum number of cutoffs
centroidsX = new float[maxclusters];  //maximum number of centroids
centroidsY = new float[maxclusters];
centroidpointX = new float[maxclusters];
centroidpointY = new float[maxclusters];
centroidpointZ = new float[maxclusters];
edgecentroids = new float[maxclusters];
min = new double[maxclusters];
max = new double[maxclusters];
levelconfidence = new double[1];
spies = new float[maxclusters];
allspiesX = new float[maxclusters][5];  //maximum number of spies in each cluster 5
allspiesY = new float[maxclusters][5];
allspiescount = new int[maxclusters];
plottspiesX = new float[maxclusters][5];
plottspiesY = new float[maxclusters][5];
adjacencyXsum = new float[maxclusters];
adjacencyYsum = new float[maxclusters];
adjacencyX = new float[maxclusters][numberofpoints];
adjacencyY = new float[maxclusters][numberofpoints];
adjacencycounter = new int[maxclusters];
bucketcounter = new double[buckets];  //counts the number of Distances D
                                           //for each bucket
int numberofpointaux = (numberofpoints * --numberofpoints)/2;
                                           //Total Distances D
Dnormbucket = new double[numberofpointaux];  //normalization of D matrix
D = new float[numberofpointaux];
I = new int[++numberofpoints];

System.out.println(" ");
System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
System.out.println("**********CLUSTERABILITY ALGORITHM**********");
System.out.println(" ");

/*-----------------------------------------------*/

//Initializing the matrixes

for (aux = 0; aux < buckets; aux++) //bucket matrix
    bucketcounter[aux] = 0;

for (aux = 0; aux < maxclusters; aux++)
    edgecentroids[aux] = 0; //edge matrix

for (aux = 0; aux < maxclusters; aux++)
{
    adjacencyXsum[aux] = 0; //adjacency matrixes
    adjacencyYsum[aux] = 0;
    adjacencycounter[aux] = 0;
}

for (aux = 0; aux < maxclusters; aux++)
    for (aux2 = 0; aux2 < numberofpoints; aux2++)
    {
        adjacencyX[aux][aux2] = 0; //adjacency matrixes
        adjacencyY[aux][aux2] = 0;
    }

for (aux = 0; aux < maxclusters; aux++)
{
    centroidsX[aux] = 0;
}
centroidsY[aux] = 0;
}
fp = new File(nameoffile);
BufferedReader my_reader;
my_reader = new BufferedReader(new FileReader(fp));

Random mynumber = new Random(); //used by the sampling algorithms

/////////////////////////////////////////////
//SAMPLING ALGORITHMS

if (choice == 1) {

/////////////////////////////////////////////
//Algorithm R

System.out.println("Sampling Algorithm: Algorithm R");
System.out.println(" ");

//Initialize
//Input the first n records and copy them to the "reservoir"

aux = 0;
while ((buf=my_reader.readLine())!=null & & aux<numberOfpoints)
{
    st = new StringTokenizer(buf);
    str = st.nextToken();
    float temp1 = Float.parseFloat(str);
    str = st.nextToken();
    float temp2 = Float.parseFloat(str);
    x[aux] = temp1;
    y[aux] = temp2;
    aux++;
} //number of elements

66
t = numberofpoints;
totalpoints = numberofpoints;

while ((buf = my_reader.readLine()) != null)
{
    t++;
    mynumber2 = Math.abs(mynumber.nextInt()) % t;
    if (mynumber2 > numberofpoints)
    {
        buf = my_reader.readLine(); //skip next record
        continue;
    }
    if (mynumber2 < numberofpoints)
    {
        st = new StringTokenizer(buf);
        str = st.nextToken();
        float temp1 = Float.parseFloat(str);
        str = st.nextToken();
        float temp2 = Float.parseFloat(str);
        x[mynumber2] = temp1;
        y[mynumber2] = temp2;
        continue;
    }
}
for (aux = 0; aux < totalpoints; aux++) //Find the biggest number
{
    if (x[aux] > Xmax)
        Xmax = x[aux];
    if (y[aux] > Ymax)
        Ymax = y[aux];
}

67
if (choice == 2) {

    // ALGORITHM S

    System.out.println("Sampling Algorithm: Algorithm S");
    while (totalpoints < numberofpoints &&
            (buf = my_reader.readLine()) != null)
    {
        mynumber1 = mynumber.nextFloat() % 1;
        if (((float)sizeoffile - (float)t) * mynumber1 >=
            (float)numberofpoints - (float)totalpoints)
        {
            st = new StringTokenizer(buf);
            str = st.nextToken(); // skip next record
            t++;
            buf = my_reader.readLine();
            continue;
        }
        if (buf != null)
        {
            st = new StringTokenizer(buf);
            str = st.nextToken();
            float temp1 = Float.parseFloat(str);
            str = st.nextToken();
            float temp2 = Float.parseFloat(str);
            x[totalpoints] = temp1;
            y[totalpoints] = temp2;
            if (x[totalpoints] > Xmax)
                Xmax = x[totalpoints];
            if (y[totalpoints] > Ymax)
                Ymax = y[totalpoints];
            totalpoints++;
        }
    }
t++;

if (choice == 3)
{

    // MY RANDOM

    System.out.println("Sampling Algorithm: Random Algorithm");

    if (sizeoffile > numberofpoints) //otherwise choose all points
    sizeflag = 1;

    upperbound = sizeoffile/numberofpoints;
    while (totalpoints<numberofpoints &&
    (buf=my_reader.readLine())!=null)
    {
        mynumber1 = 1 + Math.abs(mynumber.nextInt()) %upperbound;
        if (sizeflag != 0) //skip records according to mynumber1
        {
            for(aux=0; aux<mynumber1 &&
            (buf=my_reader.readLine())!=null; aux++)
            {
                st = new StringTokenizer(buf);
                str = st.nextToken();
            }
        }
        if (buf != null)
        {
            st = new StringTokenizer(buf);
            str = st.nextToken();
            float temp1 = Float.parseFloat(str);

            etc...
        }
    }
}
str = st.nextToken();
float temp2 = Float.parseFloat(str);
x[totalpoints] = temp1;
y[totalpoints] = temp2;
if (x[totalpoints] > Xmax)
    Xmax = x[totalpoints];
if (y[totalpoints] > Ymax)
    Ymax = y[totalpoints];
totalpoints++; //number of elements
}
}
}
System.out.println("Reference point: "+"X: "+x[0] + " Y: "+y[0]);

//Calculation of the distance between all points
//total number of distances = (|D|^2 - |D|)/2

totaledges = (totalpoints * --totalpoints)/2;
--totaledges;
for (aux = 0; aux< toaledges; aux++)
{
    aux2 = aux;
    while (aux2 <-totalpoints)
    {
        D[counter] = Math.round(Math.abs(x[aux]-x[++aux2])+Math.abs(y[aux]-y[aux2]));
        if (D[counter] > Emax)
            Emax = D[counter];
        counter++;
    }
}
++totaledges;
++totalpoints;

System.out.println("Total points " + totalpoints);
System.out.println("Total edges " + counter);
System.out.println("Buckets: " + buckets);
bucketsaux = buckets;

for (b = 0; b< counter; b++) //normalizing the edge matrix
{
    float temp3 = D[b];
    float temp4 = Emax;
    Dnormbucket[b] = (int) Math.round(bucketsaux * (temp3/temp4));
    bucketcounter[(int)Dnormbucket[b]] =
    bucketcounter[(int)Dnormbucket[b]] + 1;
}
buckets++;
time2 = System.currentTimeMillis();

/////////////////////////////////////////////////////////////////////

//PLOT: EDGE FREQUENCY HISTOGRAM
/////////////////////////////////////////////////////////////////////

float xray [] = new float [buckets];
float y1ray [] = new float [buckets];

for (int r = 0; r < buckets; r++) {
    xray[r] = (float)r;
    y1ray[r] = (float)bucketcounter[r];
}
Dislin.metafl ("cons");
Dislin.disini ();
Dislin.pagera ();
Dislin.complx();
Dislin.axspos (450, 1800);
Dislin.axslen (2200, 1200);
Dislin.name ("X-axis", "x");
Dislin.name ("Y-axis", "y");
Dislin.labdig (-1, "x");
Dislin.ticks (10, "xy");
Dislin.titlin ("Edge Histogram", 1);
Dislin.graf (0.f,(float)buckets, 0.f, 1.f,
0.f,(float)(totalpoints*(totalpoints - 1)/2), 0.f,
(float)(totalpoints*(totalpoints - 1)/(2*buckets)));
Dislin.title ();
Dislin.color ("red");
Dislin.curve (xray, y1ray, buckets);
Dislin.color ("fore");
Dislin.dot ();
Dislin.xaxgtx ();
Dislin.disfin ();

//Calculation of cutoffs: the minimums in the histogram

time3 = System.currentTimeMillis();
--buckets;
aux2 = 0;

for (aux = 0; aux < buckets;)
{
    temp5 = bucketcounter[aux];
    temp6 = bucketcounter[++aux];
    temp7 = temp5 - temp6;

    if (temp7 < 0) //found the first largest minimum: a cutoff
    72
{  
cutoff[aux2] = --aux;  
aux++;  
System.out.println("Cutoff: " + cutoff[aux2]);  
countcutoffs++;  
aux2++;  
}  
//skipping minimums between largest minimum and a maximum  
while (aux < buckets && temp7 < 0)  
{  
temp5 = bucketcounter[aux];  
temp6 = bucketcounter[++aux];  
temp7 = temp5 - temp6;  
}  
}  
System.out.println("Total cutoffs: " + countcutoffs);  
System.out.println(" ");

="/\"Calculating the adjacency matrixes for the sample

counter = 0;  
aux2 =0;  
aux = 0;  
tmpaux1 = countcutoffs;  
tmpaux1--;  
tmpaux2=0;  
tmpaux3=0;  
--totedges;  
--totalpoints;  

while (aux2 < totalpoints)  
{
countcutoffs2 = 0;
D[counter] = Math.abs(x[aux]-x[++aux2])
    + Math.abs(y[aux]-y[aux2]);
float temp3 = D[counter];
float temp4 = Emax;
Dnormbucket[counter] = (int)Math.round(bucketsaux*(temp3/temp4));
for (tmpaux2 = 0; tmpaux2 < countcutoffs; tmpaux2++)
{
    flag = 0;
    if (Dnormbucket[counter] < cutoff[tmpaux2] && flag == 0)
    {
        newtmp = adjacencycounter[countcutoffs2];
        adjacencyXsum[countcutoffs2] =
            adjacencyXsum[countcutoffs2] + x[aux2];
        adjacencyYsum[countcutoffs2] =
            adjacencyYsum[countcutoffs2] + y[aux2];
        adjacencyX[countcutoffs2][newtmp] = x[aux2];
        adjacencyY[countcutoffs2][newtmp] = y[aux2];
        adjacencycounter[countcutoffs2] =
            adjacencycounter[countcutoffs2] + 1;
        flag = 1;
    }
    if (flag == 1)
        break;
}

// adjacency points in the last cluster bigger than the last cutoff
if (tmpaux2 == tmpaux1)
{
    countcutoffs2++;
    if (Dnormbucket[counter] >= cutoff[tmpaux1])
    {
        newtmp = adjacencycounter[countcutoffs2];
        adjacencyXsum[countcutoffs2] =
            adjacencyXsum[countcutoffs2] + x[aux2];
        adjacencyYsum[countcutoffs2] =
            adjacencyYsum[countcutoffs2] + y[aux2];
        adjacencycounter[countcutoffs2] =
            adjacencycounter[countcutoffs2] + 1;
        flag = 1;
    }
    if (flag == 1)
        break;
}
adjacencyXsum[countcutoffs2] += x[aux2];
adjacencyYsum[countcutoffs2] =
    adjacencyYsum[countcutoffs2] + y[aux2];
adjacencyX[countcutoffs2] [newtmp] =
    x[aux2];
adjacencyY[countcutoffs2] [newtmp] =
    y[aux2];
adjacencycounter[countcutoffs2] =
    adjacencycounter[countcutoffs2] + 1;
    flag = 1;
    }
    if (flag == 1)
        break;

    countcutoffs2++;
}
counter++;
}
for (aux = 0; aux <= countcutoffs; aux++)
    System.out.println("Total points in cluster " + adjacencycounter[aux]);

DIRECTORY

Calculation of the centroids

totaledges++;
totalpoints++;

//Average of adjacency values to get the centroids
for (aux = 0; aux <= countcutoffs; aux++)
    {
        if (adjacencycounter[aux] != 0)
        {

75
centroidsX[aux] = adjacencyXsum[aux]/adjacencycounter[aux];
centroidsY[aux] = adjacencyYsum[aux]/adjacencycounter[aux];
}
}

for (aux = 0; aux<=countcutoffs; aux++)
    if (adjacencycounter[aux] != 0)
        System.out.println("Centroid " + "X: "+centroidsX[aux] + " Y: "
        +centroidsY[aux]);
System.out.println(" ");

//Calculation of the nearest point to the centroids used in spies
//When we plot the spies all the spies have to be part of the cluster
//If the number of spies is > 2 then one of the spies corresponds to a
//centroid. However, given that all the spies have to be real points
//we have to find a point in the cluster that is closest to the
//centroid of that cluster

for (aux = 0; aux<=countcutoffs; aux++)
{
    if (adjacencycounter[aux] != 0)
    {
        for (aux2 = 0; aux2 <= adjacencycounter[aux]; aux2++)
        {
            D[counter] = Math.round(Math.abs(centroidsX[aux]
            -adjecencyX[aux][aux2])
            +Math.abs(centroidsY[aux]
            -adjecencyY[aux][aux2]));

            if (D[counter] < centroiddistance)
            {

            }
centroiddistance = D[counter];
centroidpointX[aux] = adjacencyX[aux][aux2];
centroidpointY[aux] = adjacencyY[aux][aux2];

}
}
}
}
}
centroiddistance = 100000000;
}

for (aux = 0; aux<=countcutoffs; aux++)
    if (adjecencycounter[aux] != 0)
        System.out.println("Centroidpoint " + "X: "+
                        centroidpointX[aux] +
                        " Y: "+centroidpointY[aux]);
System.out.println(" ");

///////////////////////////////////////////////////////////////////////////////////
//Calculating the total number of spies
//points-perc: points per cluster = adjecencycounter[aux]
//total-points: total points = totalpoints
//total-clusters: total clusters = countcutoffs + 1
//spies = points-perc / (total-points/total-clusters)

countcutoffs++;
int realclusters = countcutoffs;

for (aux = 0; aux<countcutoffs; aux++)
    {
        if (adjacencycounter[aux] == 0) //take out the empty clusters
            realclusters--;
    }

for (aux = 0; aux<countcutoffs; aux++)

77
{
    spies[aux]=(Math.round((float)adjecencycounter[aux])/
            (float)(totalpoints/realclusters));
    if (spies [aux] == 0 & & adjacencycounter[aux]!=0)
        spies [aux] = 1;
    if (adjacencycounter[aux] != 0)
        System.out.println("Spies:" + spies[aux]);
}
countcutoffs--;

 переведено на русский

找出可能的点来表示一个间谍在的簇

// Maximum number of spies = 4 vertex + 1 centroid = 5

aux3 = 0;

for (aux = 0; aux<=countcutoffs; aux++)
    allspiescounter[aux] = 0;

for (aux = 0; aux <= countcutoffs; aux++)
{
    if (spies [aux] != 0)  // Non-Empty cluster
    {
        aux3 = 0;

        // CALCULATING (Xmin, Ymin)

        X1 = 100000000;  // X1 = the first point in the cluster
        Y1 = 100000000;  // Y1 = the first point in the cluster

        // Calculating (Xmin, )
        for (aux2 = 0; aux2<adjecencycounter[aux]; aux2++)
            if (adjecencyX[aux][aux2]< X1)


{ 
    X1 = adjacencyX[aux][aux2];
    Y1 = adjacencyY[aux][aux2];
}

//Finding (Xmin, Ymin)
for (aux2 = 0; aux2<adjacencycounter[aux]; aux2++)
    if (adjacencyX[aux][aux2]==X1 &&
        adjacencyY[aux][aux2]<Y1)
    {
        X1 = adjacencyX[aux][aux2];
        Y1 = adjacencyY[aux][aux2];
    }

allspiesX[aux][aux3] = X1;
allspiesY[aux][aux3] = Y1;
allspiescounter[aux] = allspiescounter[aux] + 1;
aux3++;

//CALCULATING (Xmin, Ymax)

X1 = 100000000;  //X1 = the first point in the cluster
Y1 = 100000000;  //Y1 = the first point in the cluster

//Calculating (Xmin, )
for (aux2 = 0; aux2<adjacencycounter[aux]; aux2++)
    if(adjacencyX[aux][aux2]<X1&&
        (adjacencyX[aux][aux2]!=allspiesX[aux][0] ||
        adjacencyY[aux][aux2]!=allspiesY[aux][0]))
    {
        X1 = adjacencyX[aux][aux2];
        Y1 = adjacencyY[aux][aux2];
    }

//Finding (Xmin, Ymax)
for (aux2 = 0; aux2<adjacencycounter[aux]; aux2++)

79
if (adjacencyX[aux][aux2]==X1 &&
adjacencyY[aux][aux2]>Y1)
{
    X1 = adjacencyX[aux][aux2];
    Y1 = adjacencyY[aux][aux2];
}

allspiesX[aux][aux3] = X1;
allspiesY[aux][aux3] = Y1;
allspiescounter[aux] = allspiescounter[aux] + 1;
aux3++;

//CALCULATING (Xmax, Ymax)

X1 = -1000000000; //X1 = the first point in the cluster
Y1 = -1000000000; //Y1 = the first point in the cluster

//Calculating (Xmax, )
for (aux2 = 0; aux2<adjacencycounter[aux]; aux2++)
    if (adjacencyX[aux][aux2] > X1)
    {
        X1 = adjacencyX[aux][aux2];
        Y1 = adjacencyY[aux][aux2];
    }

//Finding (Xmax, Ymax)
for (aux2 = 0; aux2<adjacencycounter[aux]; aux2++)
    if (adjacencyX[aux][aux2]==X1 &&
        adjacencyY[aux][aux2]>Y1)
    {
        X1 = adjacencyX[aux][aux2];
        Y1 = adjacencyY[aux][aux2];
    }

allspiesX[aux][aux3] = X1;
allspiesY[aux][aux3] = Y1;

80
allspiescounter[aux] = allspiescounter[aux] + 1;
aux3++;

//CALCULATING (Xmax, Ymin)

X1 = -10000000;  //X1 = the first point in the cluster
Y1 = -10000000;  //Y1 = the first point in the cluster

//Calculating (Xmax2, )
for (aux2 = 0; aux2<adjecencycounter[aux]; aux2++)
    if (adjecencyX[aux][aux2]>X1&&
        (adjecencyX[aux][aux2]!= allspiesX[aux] [2] ||
         adjecencyY[aux][aux2]!= allspiesY[aux] [2]))
    {
        X1 = adjecencyX[aux][aux2];
        Y1 = adjecencyY[aux][aux2];
    }

//Finding (Xmax, Ymin)
for (aux2 = 0; aux2<adjecencycounter[aux]; aux2++)
    if (adjecencyX[aux][aux2]==X1 &&
        adjecencyY[aux][aux2]<Y1)
    {
        X1 = adjecencyX[aux][aux2];
        Y1 = adjecencyY[aux][aux2];
    }

allspiesX[aux] [aux3] = X1;
allspiesY[aux] [aux3] = Y1;
aux3++;
allspiescounter[aux] = allspiescounter[aux] + 1;
}

///////////////////////////////////////////////////////////////////

//Assigning the spies to the clusters

81
for (aux = 0; aux <= countcutoffs; aux++)
  allspiescounter[aux] = 0;

for (aux = 0; aux <= countcutoffs; aux++)
{
  if (spies [aux] != 0) //Empty cluster
    {
    if (spies[aux] == 1)
      {
      plotspiesX[aux][0] = centroidsX[aux]; //1 spie
      plotspiesY[aux][0] = centroidsY[aux];
      allspiescounter[aux] = allspiescounter[aux] + 1;
      continue;
      }
  if (spies[aux] > 1)
    {
    plotspiesX[aux][0] = allspiesX[aux][0]; //1 spie
    plotspiesY[aux][0] = allspiesY[aux][0];
    allspiescounter[aux] = allspiescounter[aux] + 1;

    plotspiesX[aux][1] = allspiesX[aux][1]; //2 spies
    plotspiesY[aux][1] = allspiesY[aux][1];
    allspiescounter[aux] = allspiescounter[aux] + 1;
    if (spies[aux] == 2)
      continue;

    plotspiesX[aux][2]=centroidpointX[aux]; //3 spies
    plotspiesY[aux][2] = centroidpointY[aux];
    allspiescounter[aux] = allspiescounter[aux] + 1;
    if (spies[aux] == 3)
      continue;
    }
}
plotspiesX[aux][3] = allspiesX[aux][2]; //4 spies
plotspiesY[aux][3] = allspiesY[aux][2];
allspiescounter[aux] = allspiescounter[aux] + 1;

if (spies[aux] == 4)
    continue;

plotspiesX[aux][4] = allspiesX[aux][3]; //5 spies
plotspiesY[aux][4] = allspiesY[aux][3];
allspiescounter[aux] = allspiescounter[aux] + 1;

}
}

counter = 0;
//actual points that serve as a spie
for (aux = 0; aux <= countcutoffs; aux++)
{
    if (spies[aux] != 0)
    {
        System.out.println(" ");
        System.out.println("Spies in cluster: ");
        for (aux2 = 0; aux2<allspiescounter[aux]; aux2++)
            System.out.print("X: " + plotspiesX[aux][aux2] +
                " Y: " + plotspiesY[aux] [aux2] + " ");
        counter++;
    }
}
System.out.println(" ");
System.out.println(" ");
System.out.println("Total clusters: "+counter);

time4 = System.currentTimeMillis();
Dislin.metafl ("cons");
Dislin.disini ();
Dislin.pagera ();
Dislin.complx ();
Dislin.axspos (450, 1800);
Dislin.axslen (2200, 1200);
Dislin.name ("X-axis", "x");
Dislin.name ("Y-axis", "y");
Dislin.labdig (-1, "x");
Dislin.ticks (10, "xy");
Dislin.titlin ("Clusters", 1);

float maxXaxis = (float) Xmax + 10;
float maxYaxis = (float) Ymax + 10;
float stepX = (float) Xmax / 15;
float stepY = (float) Ymax / 15;

Dislin.graf (0.f, maxXaxis, 0.f, stepX, 0.f, maxYaxis, 0.f, stepY);
float tempX[], tempY[];
tempX = new float[numberofpoints];
tempY = new float[numberofpoints];
int color = 25;
Dislin.title ();

aux3 = 0;
for (aux = 0; aux <= countcutoffs; aux++)
{
    aux3 = adjacencycounter[aux];
    if (aux3 > 0) {

84
for (aux2 = 0; aux2 < aux3; aux2++)
{
    tempX[aux2] = adjacencyX[aux][aux2];
    tempY[aux2] = adjacencyY[aux][aux2];
    tempX[aux2] = Dislin.xposn(tempX[aux2]);
    tempY[aux2] = Dislin.yposn(tempY[aux2]);
    Dislin.point ((int)tempX[aux2], (int)tempY[aux2],
                 5, 5, color);
}
color += 100;
}

UCCEEDED

// PLOTTING THE SPIES

color = 25;
aux3 = 0;
for (aux = 0; aux <= countcutoffs; aux++)
{
    aux3 = allspiescounter[aux];
    if (aux3 > 0) {
        for (aux2 = 0; aux2 < aux3; aux2++)
        {
            tempX[aux2] = plotspiesX[aux][aux2];
            tempY[aux2] = plotspiesY[aux][aux2];
            tempX[aux2] = (int)Dislin.xposn(tempX[aux2]);
            tempY[aux2] = (int)Dislin.yposn(tempY[aux2]);
            Dislin.point ((int)tempX[aux2], (int)tempY[aux2],
                           13, 13, color);
        }
        color += 100;
    }
}

85
Dislin.color ("fore");
Dislin.dash (0);
Dislin.xaxgit (0);
Dislin.disfin (0);

    time5 = System.currentTimeMillis();

    //Level of confidence

    aux2 = 0;
    flagsign = 0;
    countmin = 0;

    //Calculating the minimuns

    for (aux = 0; aux < buckets;)
    {
        temp5 = bucketcounter[aux];
        temp6 = bucketcounter[++aux];
        temp7 = temp5 - temp6;
        if (temp7<0)
        {
            min[aux2] = temp5;
            countmin++;
            aux2++;
        }
        if (aux == buckets && temp7 >= 0)
        {
            min[aux2] = temp6;
            aux2++;
            countmin++;
        }
    }

86
while (aux<buckets && temp7<0)
{
    temp5 = bucketcounter[aux];
    temp6 = bucketcounter[++aux];
    temp7 = temp5 - temp6;
    if (temp7 > 0)
        aux--;;
}
}

aux2 = 0;
flagsign = 0;
countmax = 0;

//Calculating the maximums

for (aux =0; aux< buckets;)
{
    temp5 = bucketcounter[aux];
    temp6 = bucketcounter[aux];
    temp7 = temp5 - temp6;
    if (temp7 > 0)
    {
        max[aux2] = temp5;
        countmax++;
        aux2++;;
    }
    if (aux == buckets && temp7 < 0)
    {
        max[aux2] = temp6;
        System.out.println("max " + max[aux2]);
        countmax++;
        aux2++;;
    }
while (aux<buckets & temp7 > 0)
{
    temp5 = bucketcounter[aux];
    temp6 = bucketcounter[++aux];
    temp7 = temp5 - temp6;
    if (temp7 < 0)
        aux--;
}

System.out.println("\n");
levelconfidence [0] = 0;
temp7 =0; //division min/max or max/min
temp6 = 0; //counts the number of divisions
for (aux = 0; aux < countmax; aux++)
    for (aux2 = 0; aux2<countmin; aux2++)
    {
        if (max[aux] < min[aux2] & min [aux2] != 0)
            temp7 = max[aux] / min[aux2];
        if (max[aux] > min[aux2] & max[aux] != 0)
            temp7 = min[aux2]/max[aux];
        levelconfidence [0] = levelconfidence [0] + temp7;
        if (temp7 != 0)
            temp6++;
    }
levelconfidence [0] = (levelconfidence[0] / temp6) * 100;
System.out.println("Level of confidence: "+levelconfidence[0] + "/%\n");
time6 = System.currentTimeMillis();
runningtime = (time2 - time1) + (time4 - time3) + (time6 - time5);
System.out.println("running time: "+ runningtime + " milliseconds\n");

{
APPENDIX F
Code of Clusterability Algorithm: three dimensions

import java.io.*;
import java.lang.*;
import java.util.*;

public class RData3d {
    public static double [] bucketcounter;
    public static void main(String[] args) throws IOException {

        String nameoffile = (args[1]);
        int sizeoffile = Integer.parseInt(args[2]);
        int numberofpoints = Integer.parseInt(args[3]);
        int choice = Integer.parseInt(args[4]);      //Sampling algorithm 1, 2, 3
        int numberofpoints2 = numberofpoints;
        char typeX [];
        char typeY [];
        char typeZ [];
        double [] Dnormbucket;  //normalized bucket counter
        double [] min;
        double [] max;
        double temp5;
        double temp6;
        double temp7;
        double [] levelconfidence;
        int t = 0;
        int j = 0;

    }
int i;
int totalclusters = 0;
int maxclusters = 20;
int pointsperc = 0;
int totalpoints = 0;
int buckets = Integer.parseInt(args[0]);
int bucketsaux;
int totaledges;
int counter = 0;
int aux;
int aux2;
int aux3 = 0;
int upperbound;
int [] allspiescounter;
int [] adjacencycounter;
int [] I;  //variable used in algorithm R
int b = 0;
int flagsign = 0;
int flag = 0;
int sizeflag = 0;
int sizeflagaux;
int countmin;
int countmax;
int countcutoffs = 0;
int countcutoffs2 = 0;
int tmpaux2;
int tmpaux3;
int tmpaux1;
int newtmp;
int newtmp2;
float Emax = 0; //maximum Edge used for normalizing the bucket counter
float Xmax = -1000000;
float Xmin = 1000000;
float Ymax = -1000000;
float Ymin = 1000000;
float Zmax = -1000000;
float Zmin = 1000000;
float centroiddistance = 1000000;
float X1 = 0;;
float Y1 = 0;
float Z1 = 0;
float [] x;
float [] y;
float [] z;
float [] D;  //edge matrix
float [] cutoff;
float [] edgacentroids;
float [] centroidsX;  //Centroid: average of all points in a cluster
float [] centroidsY;
float [] centroidsZ;
float [] centroidpointX;  //Point closest to a centroid used in spies
float [] centroidpointY;
float [] centroidpointZ;
float [] adjacencyXsum;
float [] adjacencyYsum;
float [] adjacencyZsum;
float [] [] adjacencyX;
float [] [] adjacencyY;
float [] [] adjacencyZ;
float [] spies;
float [][] allspiesX;
float [][] allspiesY;
float [] [] allspiesZ;
float [][] plotspiesX;
float [][] plotspiesY;
float [][] plotspiesZ;
long time1 = System.currentTimeMillis();
long time2;
long time3;
long time4;
long time5;
long time6;
long runningtime; //does not include the time to plot the graphs
int mynumber2 =0; //Algorithm R
float mynumber1 = 0; //Algorithm S
File fp;
String buf, str;
StringTokenizer st;

x = new float[numberofpoints];
y = new float[numberofpoints];
z = new float[numberofpoints];
typeX = new char[numberofpoints];
typeY = new char[numberofpoints];
typeZ = new char[numberofpoints];
cutoff = new float[maxclusters]; //maximum number of cutoffs
centroidsX = new float[maxclusters]; //maximum number of centroids
centroidsY = new float[maxclusters];
centroidsZ = new float[maxclusters];
centroidpointX = new float[maxclusters];
centroidpointY = new float[maxclusters];
centroidpointZ = new float [maxclusters];
edgecentroids = new float[maxclusters];
min = new double [maxclusters];
max = new double [maxclusters];
levelconfidence = new double [1];
spies = new float [maxclusters];
allspiesX=new float [maxclusters][5]; //maximum number of spies in each cluster 5
allspiesY = new float [maxclusters][5];
allspiesZ = new float [maxclusters][5];
allspiescounter = new int [maxclusters];
plotspiesX = new float[maxclusters][5];
plotspiesY = new float[maxclusters][5];
plotspiesZ = new float[maxclusters][5];
adjecencyXsum = new float[maxclusters];
adjecencyYsum = new float[maxclusters];
adjecencyZsum = new float[maxclusters];
adjecencyX = new float [maxclusters][numberofpoints];
adjecencyY = new float [maxclusters] [numberofpoints];
adjecencyZ = new float [maxclusters] [numberofpoints];
adjecencycounter = new int [maxclusters];
bucketcounter = new double [buckets];  //counts the number of Distances D
                                           //for each bucket

int numberofpointaux = (numberofpoints * --numberofpoints)/2;
                                           //Total Distances D

Dnormbucket = new double[numberofpointaux];  //normalization of matrix D
D = new float[numberofpointaux];
I = new int[++numberofpoints];

System.out.println(" ");
System.out.println("********************************************************************");
System.out.println("********CLUSTERABILITY ALGORITHM********");
System.out.println(" ");

///////////////////////////////////////////////////////////////////////////
//Initializing the matrixes

numberofpoints = numberofpoints2;

for (aux = 0; aux < buckets;aux++)  //bucket matrix
    bucketcounter[aux] = 0;
for (aux = 0; aux < maxclusters; aux++)
edgecentroids[aux] = 0; // edge matrix

for (aux = 0; aux < maxclusters; aux++)
{
    adjacencyXsum[aux] = 0; // adjacency matrixes
    adjacencyYsum[aux] = 0;
    adjacencyZsum[aux] = 0;
    adjacencycounter[aux] = 0;
}

for (aux = 0; aux < maxclusters; aux++)
for (aux2 = 0; aux2 < numberofpoints; aux2++)
{
    adjacencyX[aux][aux2] = 0; // adjacency matrixes
    adjacencyY[aux][aux2] = 0;
    adjacencyZ[aux][aux2] = 0;
}

for (aux = 0; aux < maxclusters; aux++)
{
    centroidsX[aux] = 0;
    centroidsY[aux] = 0;
    centroidsZ[aux] = 0;
}

fp = new File(nameoffile);
BufferedReader my_reader;
my_reader = new BufferedReader(new FileReader(fp));

Random mynumber = new Random(); // used by the sampling algorithms
if (choice == 1) {

    //Algorithm R

    System.out.println("Sampling Algorithm: Algorithm R");
    System.out.println(" ");
    //Initialize
    //Input the first n records and copy them to the "reservoir"

    aux = 0;

    while ((buf= my_reader.readLine()) != null && aux < number_of_points)
    {
        st = new StringTokenizer(buf);
        str = st.nextToken();
        float temp1 = Float.parseFloat(str);
        str = st.nextToken();
        float temp2 = Float.parseFloat(str);
        str = st.nextToken();
        float temp3 = Float.parseFloat(str);
        x[aux] = temp1;
        y[aux] = temp2;
        z[aux] = temp3;
        aux++;
        //number of elements
    }

    t = number_of_points;
    totalpoints = number_of_points;

    while ((buf = my_reader.readLine()) != null)

{ 
    t++;
    mynumber2 = Math.abs(mynumber.nextInt()) %t;
    if (mynumber2 > numberofpoints)
    {
        buf = my_reader.readLine(); //skip next record
        continue;
    }
    if (mynumber2 < numberofpoints)
    {
        st = new StringTokenizer(buf);
        str = st.nextToken();
        float temp1 = Float.parseFloat(str);
        str = st.nextToken();
        float temp2 = Float.parseFloat(str);
        str = st.nextToken();
        float temp3 = Float.parseFloat(str);
        x[mynumber2] = temp1;
        y[mynumber2] = temp2;
        z[mynumber2] = temp3;
        continue;
    }
}

for (aux = 0; aux< totalpoints; aux++) //Find the biggest number
{
    if (x[aux] > Xmax)
        Xmax = x[aux];
    if (y[aux] > Ymax)
        Ymax = y[aux];
    if (z[aux] > Zmax)
        Zmax = z[aux];
    if (x[aux] < Xmin)
        Xmin = x[aux];

96
Xmin = x[aux];
if (y[aux] < Ymin)
    Ymin = y[aux];
if (z[aux] < Zmin)
    Zmin = z[aux];
}

if (choice == 2)
{
    //ALGORITHM S

    System.out.println("Sampling Algorithm: Algorithm S");
    while (totalpoints<numberofpoints &&
        (buf=my_reader.readLine())!=null)
    {
        mynumber1 = mynumber.nextFloat() % 1;
        if (((float)sizeoffile-(float)t)*mynumber1>=
            (float)numberofpoints-(float)totalpoints)
        {
            st = new StringTokenizer(buf);
            str = st.nextToken(); //skip next record
            t++;
            buf = my_reader.readLine();
            continue;
        }
        if (buf != null)
        {
            st = new StringTokenizer(buf);
            str = st.nextToken();
            float temp1 = Float.parseFloat(str);
            str = st.nextToken();
            float temp2 = Float.parseFloat(str);
str = st.nextToken();
float temp3 = Float.parseFloat(str);
x[totalpoints] = temp1;
y[totalpoints] = temp2;
z[totalpoints] = temp3;
if (x[totalpoints] > Xmax)
    Xmax = x[totalpoints];
if (y[totalpoints] > Ymax)
    Ymax = y[totalpoints];
if (z[totalpoints] > Zmax)
    Zmax = z[totalpoints];
if (x[aux] < Xmin)
    Xmin = x[aux]
if (y[aux] < Ymin)
    Ymin = y[aux];
if (z[aux] < Zmin)
    Zmin = z[aux];
totalpoints++;
t++;
}
}
}
}
if (choice == 3)
{

// MY RANDOM

System.out.println("Sampling Algorithm: Random Algorithm");

if (sizeoffile > numberofpoints) //otherwise choose all points
    sizeflag = 1;

upperbound = sizeoffile/numberofpoints;

98
while (totalpoints<numberofpoints &&
        (buf=my_reader.readLine())!=null)
{
    mynumber1 = 1 + Math.abs(mynumber.nextInt()) % upperbound;
    if (sizeflag != 0) // skip records according to mynumber1
    {
        for(aux=0; aux<mynumber1 &&
                (buf=my_reader.readLine())!=null; aux++)
        {
            st = new StringTokenizer(buf);
            str = st.nextToken();
        }
    }
    if (buf != null)
    {
        st = new StringTokenizer(buf);
        str = st.nextToken();
        float temp1 = Float.parseFloat(str);
        str = st.nextToken();
        float temp2 = Float.parseFloat(str);
        str = st.nextToken();
        float temp3 = Float.parseFloat(str);
        x[totalpoints] = temp1;
        y[totalpoints] = temp2;
        z[totalpoints] = temp3;
        if (x[totalpoints] > Xmax)
            Xmax = x[totalpoints];
        if (y[totalpoints] > Ymax)
            Ymax = y[totalpoints];
        if (z[totalpoints] > Zmax)
            Zmax = z[totalpoints];
        if (x[aux] < Xmin)
            Xmin = x[aux];
    
    
    99
if (y[aux] < Ymin)
    Ymin = y[aux];
if (z[aux] < Zmin)
    Zmin = z[aux];
totalpoints++; //number of elements

System.out.println("Reference point: "+"X: "+x[0] + "Y: "+y[0] + "Z: "+z[0]);
System.out.println("Xmin: "+Xmin +"Ymin "+Ymin +"ZMin: "+Zmin);


//Calculation of the distance between all points
//total number of distances = (|D|^2 - |D|)/2

totaledges = (totalpoints * --totalpoints)/2;
--totaledges;
for (aux = 0; aux< totaledges; aux++)
{
    aux2 = aux;
    while (aux2 <totalpoints)
    {
        counter = Math.round(Math.abs(x[aux]-x[aux2]))
        + Math.abs(y[aux]-y[aux2])
        + Math.abs(z[aux]-z[aux2]));
        if (D[counter] > Emax)
            Emax = D[counter];
        counter++;
    }
}
++totaledges;
++totalpoints;

System.out.println("Total points " + totalpoints);
System.out.println("Total edges " + counter);
System.out.println("Buckets: " + buckets);
bucketsaux =--buckets;

for (b = 0; b< counter; b++) //normalizing the edge matrix
{
    float temp3 = D[b];
    float temp4 = Emax;
    Dnormbucket[b] = (int) Math.round (bucketsaux * (temp3/temp4));
    bucketcounter[(int)Dnormbucket[b]] =
        bucketcounter[(int)Dnormbucket[b]] + 1;
}
buckets++;
time2 = System.currentTimeMillis();

for (b = 0; b < buckets; b++)

    System.out.println(bucketcounter[b]);


//PLOT: EDGE FREQUENCY HISTOGRAM

float xray[] = new float[buckets];
float y1ray[] = new float[buckets];

for (int r = 0; r < buckets; r++) {
    xray[r] = (float)(r);
y1ray[r] = (float) bucketcounter[r];
}
Dislin.metafl ("cons");
Dislin.disini ();
Dislin.pagera ();
Dislin.complx ();
Dislin.axspos (450, 1800);
Dislin.axslen (2200, 1200);
Dislin.name ("X-axis", "x");
Dislin.name ("Y-axis", "y");
Dislin.labdig (-1, "x");
Dislin.ticks (10, "xy");
Dislin.titlin ("Edge Histogram", 1);
Dislin.graf (0.f,(float)buckets, 0.f, 1.f,
          0.f,(float)(totalpoints*(totalpoints - 1)/2), 0.f,
          (float)(totalpoints*(totalpoints - 1)/(2*buckets)));
Dislin.title ();
Dislin.color ("red");
Dislin.curve (xray, y1ray, buckets);
Dislin.color ("fore");
Dislin.dot ();
Dislin.xaxgit ();
Dislin.disfin ();

//Calculation of cutoffs: the minimums in the histogram

time3 = System.currentTimeMillis();
--buckets;
aux2 = 0;

for (aux = 0; aux < buckets;)
{


102
temp5 = bucketcounter[aux];
temp6 = bucketcounter[++aux];
temp7 = temp5 - temp6;

if (temp7 < 0) //found the first largest minimum: a cutoff
{
    cutoff[aux2] = --aux;
    aux++;
    System.out.println("Cutoff: "+ cutoff[aux2]);
    countcutoffs++;
    aux2++;
}
//skipping minimums between largest minimum and a maximum
while (aux < buckets && temp7 < 0)
{
    temp5 = bucketcounter[aux];
    temp6 = bucketcounter[++aux];
    temp7 = temp5 - temp6;
}
}
System.out.println("Total cutoffs: "+ countcutoffs);
System.out.println(" ");

////////////////////////////////////////////////////////////////////////////////////
//Calculating the adjacency matrixes for the sample

counter = 0;
aux2 = 0;
aux = 0;
tmpaux1 = countcutoffs;
tmpaux1--;
tmpaux2=0;
tmpaux3=0;
while (aux2 < totalpoints)
{
    countcutoffs2 = 0;
    D[counter]=Math.abs(x[aux]-x[++aux2]) + Math.abs(y[aux]-y[aux2])
    + Math.abs(z[aux]-z[aux2]);
    float temp3 = D[counter];
    float temp4 = Emax;
    Dnormbucket[counter]=(int)Math.round(bucketsaux*(temp3/temp4));
    for (tmpaux2 = 0; tmpaux2 < countcutoffs; tmpaux2++)
    {
        flag = 0;
        if (Dnormbucket[counter] < cutoff[tmpaux2] && flag == 0)
        {
            newtmp = adjecencycounter[countcutoffs2];
            adjacencyXsum[countcutoffs2] =
            adjacencyXsum[countcutoffs2] + x[aux2];
            adjacencyYsum[countcutoffs2] =
            adjacencyYsum[countcutoffs2] + y[aux2];
            adjacencyZsum[countcutoffs2] =
            adjacencyZsum[countcutoffs2] + z[aux2];
            adjacencyX[countcutoffs2][newtmp] = x[aux2];
            adjacencyY[countcutoffs2][newtmp] = y[aux2];
            adjacencyZ[countcutoffs2][newtmp] = z[aux2];
            adjacencycounter[countcutoffs2] =
            adjacencycounter[countcutoffs2] + 1;
            flag = 1;
        }
        if (flag == 1)
            break;
    }
}
//adjacency points in the last cluster bigger than the last cutoff
if (tmpaux2 == tmpaux1)
{
    countcutoffs2++;
    if (Dnormbucket[counter] >= cutoff[tmpaux1])
    {
        newtmp=adjacencycounter[countcutoffs2];
        adjacencyXsum[countcutoffs2] =
            adjacencyXsum[countcutoffs2]+x[aux2];
        adjacencyYsum[countcutoffs2] =
            adjacencyYsum[countcutoffs2]+y[aux2];
        adjacencyZsum[countcutoffs2] =
            adjacencyZsum[countcutoffs2]+z[aux2];
        adjacencyX[countcutoffs2] [newtmp] =
            x[aux2];
        adjacencyY[countcutoffs2] [newtmp] =
            y[aux2];
        adjacencyZ[countcutoffs2] [newtmp] =
            z[aux2];
        adjacencycounter[countcutoffs2]=
            adjacencycounter[countcutoffs2] + 1;
        flag = 1;
    }
    if (flag == 1)
    {
        break;
    }
    countcutoffs2++;
}

counter++;
}
for (aux = 0; aux<= countcutoffs; aux++)
System.out.println("Total points in cluster "+ adjacencycounter[aux]);
aux3 = 0;

///////////////////
//Calculation of the centroids

totaledges++;
totalpoints++;

//Average of adjacency values to get the centroids

for (aux = 0; aux<=countcutoffs; aux++)
{
    if (adjecencycounter[aux] != 0)
    {
        centroidsX[aux] = adjacencyXsum[aux]/adjecencycounter[aux];
        centroidsY[aux] = adjacencyYsum[aux]/adjecencycounter[aux];
        centroidsZ[aux] = adjacencyZsum[aux]/adjecencycounter[aux];
    }
}

for (aux = 0; aux<=countcutoffs; aux++)
    if (adjecencycounter[aux] != 0)
        System.out.println("Centroid " + "X: "+centroidsX[aux] +
        " Y: "+centroidsY[aux]+ " Z: "+centroidsZ[aux]);
System.out.println(" ");

///////////
//Calculation of the nearest point to the centroids used in spies
//When we plot the spies all the spies have to be part of the cluster
//If the number of spies is > 2 then one of the spies corresponds to a
//centroid. However, given that all the spies have to be real points
//we have to find a point in the cluster that is closest to the
//centroid of that cluster
///////////

106
for (aux = 0; aux<=countcutoffs; aux++)
{
    if (adjecencycounter[aux] != 0)
    {
        for (aux2 = 0; aux2 <=adjecencycounter[aux]; aux2++)
        {
            D[counter] = Math.round(Math.abs(centroidsX[aux]
                -adjecencyX[aux][aux2])
                +Math.abs(centroidsY[aux]
                -adjecencyY[aux][aux2])
                +Math.abs(centroidsZ[aux]
                -adjecencyZ[aux][aux2]));
            if (D[counter] < centroiddistance)
            {
                centroiddistance = D[counter];
                centroidpointX[aux] = adjecencyX[aux][aux2];
                centroidpointY[aux] = adjecencyY[aux][aux2];
                centroidpointZ[aux] = adjecencyZ[aux][aux2];
            }
        }
    }
    centroiddistance = 100000000;
}

for (aux = 0; aux<=countcutoffs; aux++)
    if (adjecencycounter[aux] != 0)
        System.out.println("Centroidpoint " + "X: "+
            centroidpointX[aux] +
            " Y: "+centroidpointY[aux]+
            " Z: "+centroidpointZ[aux]);
System.out.println(" ");
Calculating the total number of spies

//points-perc: points per cluster = adjacencycounter[aux]
//total-points: total points = totalpoints
//total-clusters: total clusters = countcutoffs + 1
//spies = points-perc / (total-points/total-clusters)

countcutoffs++; 
int realclusters = countcutoffs;

for (aux = 0; aux<countcutoffs; aux++) 
    if (adjacencycounter[aux] == 0) //take out the empty clusters
        realclusters--; 

for (aux = 0; aux<countcutoffs; aux++)
{
    spies[aux]=(Math.round((float)adjacencycounter[aux]/
        (float)(totalpoints/realclusters)));
    if (spies [aux] == 0 && adjacencycounter[aux]!=0)
        spies [aux] = 1;
    if (adjacencycounter[aux] != 0)
        System.out.println("Spies:" + spies[aux]);
} 

countcutoffs--; 

Finding possible points that correspond to a spie in thecluster
Maximum number of spies = 4 vertex + 1 centroid = 5

aux3 = 0;
for (aux = 0; aux <= countcutoffs; aux++)
    allspiescounter[aux] = 0;

for (aux = 0; aux <= countcutoffs; aux++)
{
    if (spies[aux] != 0)  //Non-Empty cluster
    {
        aux3 = 0;

        //CALCULATING (Xmin, Ymin)
        X1 = 100000000;  //X1 = the first point in the cluster
        Y1 = 100000000;  //Y1 = the first point in the cluster

        //Calculating (Xmin, )
        for (aux2 = 0; aux2 < adjacencycounter[aux]; aux2++)
            if (adjacencyX[aux][aux2] < X1)
            {
                X1 = adjacencyX[aux][aux2];
                Y1 = adjacencyY[aux][aux2];
                Z1 = adjacencyZ[aux][aux2];
            }

        //Finding (Xmin, Ymin)
        for (aux2 = 0; aux2 < adjacencycounter[aux]; aux2++)
            if (adjacencyX[aux][aux2] == X1 &&
                adjacencyY[aux][aux2] < Y1)
            {
                X1 = adjacencyX[aux][aux2];
                Y1 = adjacencyY[aux][aux2];
                Z1 = adjacencyZ[aux][aux2];
            }

        allspiesX[aux][aux3] = X1;
        allspiesY[aux][aux3] = Y1;
        allspiesZ[aux][aux3] = Z1;
}

109
allspiescounter[aux] = allspiescounter[aux] + 1;
aux3++;

//CALCULATING (Xmin, Ymax)
X1 = 100000000;  //X1 = the first point in the cluster
Y1 = 100000000;  //Y1 = the first point in the cluster

//Calculating (Xmin, )
for (aux2 = 0; aux2<adjecencycounter[aux]; aux2++)
if(adjecencyX[aux][aux2]<X1&&
(adjecencyX[aux][aux2]!= allspiesX[aux][0] ||
adjecencyY[aux][aux2]!=allspiesY[aux] [0]))
{
    X1 = adjecencyX[aux][aux2];
    Y1 = adjecencyY[aux][aux2];
    Z1 = adjecencyZ[aux][aux2];
}
//Finding (Xmin, Ymax)
for (aux2 = 0; aux2<adjecencycounter[aux]; aux2++)
if (adjecencyX[aux][aux2]==X1 &&
adjecencyY[aux][aux2]>Y1)
{
    X1 = adjecencyX[aux][aux2];
    Y1 = adjecencyY[aux][aux2];
    Z1 = adjecencyZ[aux][aux2];
}
allspiesX[aux] [aux3] = X1;
allspiesY[aux] [aux3] = Y1;
allspiesZ[aux] [aux3] = Z1;
allspiescounter[aux] = allspiescounter[aux] + 1;
aux3++;
X1 = -100000000;  //X1 = the first point in the cluster
Y1 = -100000000;  //Y1 = the first point in the cluster

//Calculating (Xmax, )
for (aux2 = 0; aux2<adjecencycounter[aux]; aux2++)
    if (adjecencyX[aux][aux2] > X1)
        {
            X1 = adjecencyX[aux][aux2];
            Y1 = adjecencyY[aux][aux2];
            Z1 = adjecencyZ[aux][aux2];
        }

//Finding (Xmax, Ymax)
for (aux2 = 0; aux2<adjecencycounter[aux]; aux2++)
    if (adjecencyX[aux][aux2]==X1 &&
        adjecencyY[aux][aux2]>Y1)
        {
            X1 = adjecencyX[aux][aux2];
            Y1 = adjecencyY[aux][aux2];
            Z1 = adjecencyZ[aux][aux2];

            allspiesX[aux][aux3] = X1;
            allspiesY[aux][aux3] = Y1;
            allspiesZ[aux][aux3] = Z1;
            allspiescounter[aux] = allspiescounter[aux] + 1;
            aux3++;

        }

//CALCULATING (Xmax, Ymin)
X1 = -100000000;  //X1 = the first point in the cluster
Y1 = -100000000;  //Y1 = the first point in the cluster

//Calculating (Xmax, )
for (aux2 = 0; aux2<adjecencycounter[aux]; aux2++)
    if (adjecencyX[aux][aux2]>X1
&& (adjacencyX[aux][aux2] != allspiesX[aux][2] || adjacencyY[aux][aux2] != allspiesY[aux][2])
{
    X1 = adjacencyX[aux][aux2];
    Y1 = adjacencyY[aux][aux2];
    Z1 = adjacencyZ[aux][aux2];
}

// Finding (Xmax, Ymin)
for (aux2 = 0; aux2 < adjacencycounter[aux]; aux2++)
    if (adjacencyX[aux][aux2]==X1 && adjacencyY[aux][aux2]<Y1)
        {
            X1 = adjacencyX[aux][aux2];
            Y1 = adjacencyY[aux][aux2];
            Z1 = adjacencyZ[aux][aux2];
        }
    allspiesX[aux][aux3] = X1;
    allspiesY[aux][aux3] = Y1;
    allspiesZ[aux][aux3] = Z1;
    aux3++;
    allspiescounter[aux] = allspiescounter[aux] + 1;
}


 ///////////// Assigning the spies to the clusters //

for (aux = 0; aux <= countcutoffs; aux++)
    allspiescounter[aux] = 0;

for (aux = 0; aux <= countcutoffs; aux++)
{
    if (spies[aux] != 0) // Empty cluster
if (spies[aux] == 1)
{
    plotspiesX[aux][0] = centroidsX[aux]; // 1 spie
    plotspiesY[aux][0] = centroidsY[aux];
    plotspiesZ[aux][0] = centroidsZ[aux];
    allspiescounter[aux] = allspiescounter[aux] + 1;
    continue;
}
if (spies[aux] > 1)
{
    plotspiesX[aux][0] = allspiesX[aux][0]; // 1 spie
    plotspiesY[aux][0] = allspiesY[aux][0];
    plotspiesZ[aux][0] = allspiesZ[aux][0];
    allspiescounter[aux] = allspiescounter[aux] + 1;

    plotspiesX[aux][1] = allspiesX[aux][1]; // 2 spies
    plotspiesY[aux][1] = allspiesY[aux][1];
    plotspiesZ[aux][1] = allspiesZ[aux][1];
    allspiescounter[aux] = allspiescounter[aux] + 1;
    if (spies[aux] == 2)
        continue;

    plotspiesX[aux][2] = centroidpointX[aux]; // 3 spies
    plotspiesY[aux][2] = centroidpointY[aux];
    plotspiesZ[aux][2] = centroidpointZ[aux];
    allspiescounter[aux] = allspiescounter[aux] + 1;
    if (spies[aux] == 3)
        continue;

    plotspiesX[aux][3] = allspiesX[aux][2]; // 4 spies
    plotspiesY[aux][3] = allspiesY[aux][2];
    plotspiesZ[aux][3] = allspiesZ[aux][2];
}
allspiescounter[aux] = allspiescounter[aux] + 1;
if (spies[aux] == 4)
    continue;

plotspiesX[aux][4] = allspiesX[aux][3];  // 5 spies
plotspiesY[aux][4] = allspiesY[aux][3];
plotspiesZ[aux][4] = allspiesZ[aux][3];
allspiescounter[aux] = allspiescounter[aux] + 1;

}
}

counter = 0;

//actual points that serve as a spie
for (aux = 0; aux <= countcutoffs; aux++)
{
    if (spies[aux] != 0)
    {
        System.out.println(" ");
        System.out.println("Spies in cluster: ");
        for (aux2 = 0; aux2<allspiescounter[aux]; aux2++)
            System.out.print("X: " + plotspiesX[aux][aux2] +
                             " Y: " + plotspiesY[aux][aux2] +
                             " Z: " + plotspiesZ[aux][aux2] + " ");
        counter++;
    }
}
System.out.println(" ");
System.out.println(" ");
System.out.println("Total clusters: " + counter);

time4 = System.currentTimeMillis();
float maxXaxis = (float) Xmax + 10;
float maxYaxis = (float) Ymax + 10;
float stepX = (Xmax - Xmin)/10;
float stepY = (Ymax - Ymin)/10;
float stepZ = (Zmax - Zmin)/10;
int n = 10;

Dislin.metafl ("cons");
Dislin.setpag ("da4p");
Dislin.disini ();
Dislin.pagera ();
Dislin.complx ();

Dislin.axspos (200, 2600);
Dislin.axslen (2000, 2000);

Dislin.name ("X-axis", "x");
Dislin.name ("Y-axis", "y");
Dislin.name ("Z-axis", "z");
Dislin.titlin ("Clusters", 1);

Dislin.view3d (-5.f, -5.f, 4.f, "abs");

System.out.println("Xmin: " + Xmin + "Ymin: " + Ymin + "zmin: " + Zmin);

Dislin.graf3d (Xmin, Xmax, Xmin, stepX,
        Ymin, Ymax, Ymin, stepY,
        Zmin, Zmax, Zmin, stepZ);
Dislin.height (50);
Dislin.title ();
Dislin.nocheck ();
Dislin.marker (21);  //marker type 0 -21
Dislin.hsymbl(17);   //size of the symbol up to 35
Dislin.incmrk(-n); //plots symbols not lines

String color [] = {"green", "blue", "yellow", "white", "red"};

aux3 =0;
for (aux = 0; aux<= countcutoffs; aux++)
{
    aux3 = adjacencycounter[aux];
    if (aux3 > 0) {
        Dislin.color (color[aux]);
        Dislin.crv3d (adjacencyX[aux], adjacencyY[aux],
                      adjacencyZ[aux], aux3);
    }
}

///////////////////////////

//PLOTTING THE SPIES

Dislin.marker (17);  //marker type 0 -21
Dislin.hsymbl(35);   //size of the symbol up to 35

aux3 =0;
for (aux = 0; aux<= countcutoffs; aux++)
{
    aux3 = allspiescounter[aux];
    if (aux3 > 0) {
        Dislin.color (color[aux]);
        Dislin.crv3d (plotspiesX[aux], plotspiesY[aux],
                      plotspiesZ[aux], aux3);
    }
}
Dislin.disfin();
time5 = System.currentTimeMillis();

//Level of confidence

aux2 = 0;
flagsign = 0;
countmin = 0;

//Calculating the minimums

for (aux = 0; aux < buckets;)
{
    temp5 = bucketcounter[aux];
    temp6 = bucketcounter[++aux];
    temp7 = temp5 - temp6;
    if (temp7 < 0)
    {
        min[aux2] = temp5;
        countmin++;
        aux2++;
    }
    if (aux == buckets && temp7 >= 0)
    {
        min[aux2] = temp6;
        aux2++;
        countmin++;
    }
    while (aux < buckets && temp7 < 0)
    {

117
temp5 = bucketcounter[aux];
temp6 = bucketcounter[++aux];
temp7 = temp5 - temp6;
if (temp7 > 0)
    aux--;
}
}
aux2 = 0;
flagsign = 0;
countmax = 0;

//Calculating the maximums

for (aux = 0; aux < buckets; )
{
    temp5 = bucketcounter[aux];
temp6 = bucketcounter[++aux];
temp7 = temp5 - temp6;
if (temp7 > 0)
    {
        max[aux2] = temp5;
countmax++;
aux2++;
    }
if (aux == buckets && temp7 < 0)
    {
        max[aux2] = temp6;
        System.out.println("max " + max[aux2]);
countmax++;
aux2++;
    }
while (aux < buckets && temp7 > 0) 
    {


}
temp5 = bucketcounter[aux];
temp6 = bucketcounter[+aux];
temp7 = temp5 - temp6;
if (temp7 < 0)
    aux--;

System.out.println("\n");
levelconfidence [0] = 0;
temp7 = 0; //division min/max or max/min
temp6 = 0; //counts the number of divisions

for (aux = 0; aux < countmax; aux++)
    for (aux2 = 0; aux2< countmin; aux2++)
    {
        if (max[aux] < min[aux2] && min [aux2] != 0)
            temp7 = max[aux] / min[aux2];
        if (max[aux] > min[aux2] && max[aux] != 0)
            temp7 = min[aux2]/max[aux];
        levelconfidence [0] = levelconfidence [0] + temp7;
        if (temp7 != 0)
            temp6++;}

levelconfidence [0] = (levelconfidence[0] / temp6) * 100;
System.out.println("Level of confidence: "+levelconfidence[0] + "%")

time6 = System.currentTimeMillis();

runningtime = (time2 - time1) + (time4 - time3) + (time6 - time5);
System.out.println("running time: "+ runningtime + " milliseconds");