SP09015

Study, Evaluation and Implementation of Various Clustering Techniques

By

Khushboo – 091313

Department of Computer Science and Engineering And Information Technology

Jaypee University of Information Technology-Waknaghat

MAY-2013

Study, Evaluation and Implementation of Various Clustering Techniques

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Computer Science & Engineering

under the Supervision of

Mr. Ravindra Bhatt

By

Khushboo(091313)

to

Jaypee University of Information and Technology Waknaghat,
Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "**Study, Evaluation and Implementation of Various Clustering Techniques**", submitted by **Khushboo(091313)** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date: 29.05.2013

Mr. Ravindra Bhatt

(Project Guide)

Acknowledgement

I extend my warm and sincere thanks to my project supervisor Mr. Ravindra Bhatt who has been a staunch supporter and motivator of this project. Right from the inception of this project work, Mr. Ravindra Bhatt guided me till the very end in the true sense of the word. He always came with innovative ways and creative terms, thus, also helping us to instill and enhance the quality of creative thinking.

Sincere thanks to Brig (Retd.) S.P. Ghrera, HOD, CSE & IT Department, for being co-operative to the students of the department and providing relevant guidance in their endeavors.

Thanks to all the teaching and non-teaching staff of the CSE Department who have helped in every possible way throughout the 4 years of the B.Tech Academic Program.

I would also like to express my gratitude to this alma mater JUIT, Waknaghat for providing proper resources as and when required such as an all time internet facility and other resources.

Hence, without giving a warm thanks to all of them who made this project work a reality, my work would be incomplete.

Khushboo

Date: 28 MAY 2013

Khushboo
(091313)

Table of Contents

List of Figures

CHAPTER 1

1. Introduction

1.1 Objective

The objective is to study and evaluate various clustering techniques and implement them and also to carry out a comparison of the techniques.

1.2 General Description

In any network, conventionally flat topology had been used. But the overheads involved in a flat structure are much high and does not support much favourable routing conditions.
So, to remove the unnecessary overheads we need to form clusters in the network so that the cluster heads communicate with each other and hence remove the overheads. Implementing Routing on such a network where the data is to be sent only to cluster heads leads to a hassle free data transmission.

The process of grouping of objects into classes of similar objects is called clustering. Cluster analysis has been widely used in numerous applications, including market research, pattern recognition, data analysis and image processing. In business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of groups of houses in a city according to house type, value, and geographic location as well as the identification of groups of automobile insurance policy holders with a high average claim cost. It can also be used t o classify documents on the web for information discovery.

1.3 Modules And Timeline

MODULES:

Module 1: Literature Survey.

Module 2: Evaluation Of various clustering techniques.

Module 3: First attempt on implementation of a clustering technique.

Module 4: Final implementation of the clustering technique.

TIMELINE:

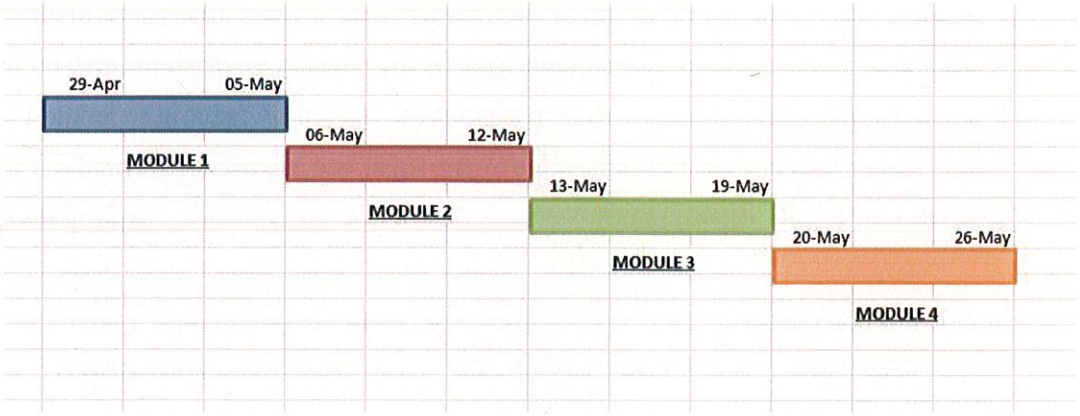


Fig 1: Timeline

CHAPTER – 2

2. Literature Survey:

2.1 Clustering:

The process of grouping of objects into classes of similar objects is called **clustering.** A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group and so may be considered as a form of data compression. Although classification is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labeling of a large set of training tuples or patterns, which the classifier uses to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity (e.g. using clustering), and then assign labels to the relatively small number of groups. Additional advantages of such a clustering based process are that it is adaptable to changes and helps single out useful features that distinguish different groups.

Cluster analysis is an important human activity. By automated clustering, we can identify dense and sparse regions in object space, and, therefore overall distribution patterns and interesting correlations among data attributes.

Cluster analysis has been widely used in numerous applications, including market research, pattern recognition, data analysis and image processing. In business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of groups of houses in a city according to house type, value, and geographic location as well as the

identification of groups of automobile insurance policy holders with a high average claim cost. It can also be used t o classify documents on the web for information discovery.

Clustering is also called **data segmentation** in some applications because clustering partitions large data sets into groups according to their similarity.
Clustering can also be used for outlier detection, where outlier (values that are "far away" from any cluster) may be more interesting that common cases. Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal cases in e commerce. For example, exceptional cases in credit card transactions, such as very expensive and frequent purchases, may be of interest as possible fraudulent activity. As a data mining function, cluster analysis can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristic of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a preprocessing step for other algorithms, such as characterization, attribute subset selection, and classification, which would then operate on selected clusters and the selected attributes or features.

Clustering is the **unsupervised classification** of patterns (observations, data items) into groups (clusters). Clustering is useful in several exploratory pattern-analysis, grouping, decision-making, and machine-learning situations, including data mining, document retrieval, image processing and pattern classification. However, in many such cases, there is a little prior information (e.g. statistical models) available about the data, and the decision-maker must make as few assumptions about the data as possible. It is under these restrictions that clustering methodology is particularly appropriate for the exploration of interrelationships among the data points to make an assessment (perhaps preliminary) of their structure. In the case of clustering, the problem is to group a given collection of unlabeled patterns into meaningful clusters. In a sense, labels are associated with clusters also, but these category labels are data driven, that is, they are obtained solely from the data.

A good clustering method will produce high quality clusters with

- High intra-class similarity
- Low inter-class similarity

The quality of a clustering result depends on both the similarity measure used by the method and its implementation. The quality of a clustering method is also measured by its ability to discover some or all of the hidden patterns.

2.2 Components of a Clustering Task

Typical pattern clustering activity involves the following:

1. Pattern representation
2. Definition of a pattern
3. Clustering or grouping
4. Data abstraction(if needed)
5. Assessment of output(if needed)

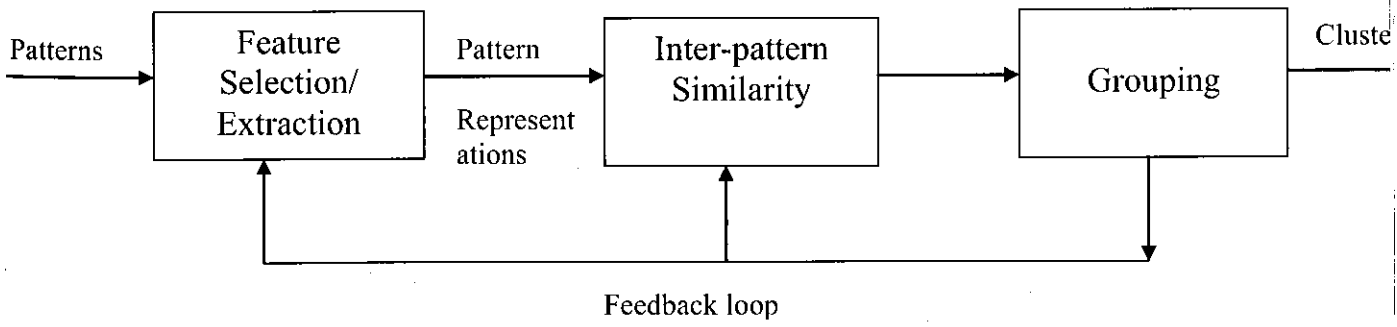


Fig 2 Stages in Clustering

Figure 2.1 depicts a typical sequencing of the first three steps, including a feedback path where the grouping process output could affect subsequent feature extraction and similarity computations.

Pattern representations refer to the number of classes, the number of available patterns, and the number, type, and scale of the features available to the clustering algorithm. Some of this information may not be controllable by the practitioner.

Feature selection is the process of identifying the most effective subset of the original features to use in clustering.

Feature Extraction is the use of one or more transformations of the input features to produce new salient features. Either both of these techniques can be used to obtain an appropriate set of features to use in clustering.

Pattern Proximity is usually measured by a distance function defined on pairs of patterns. A variety of distance measures are is use in various communities. A simple distance measure like Euclidean distance can often be used to reflect dissimilarity between two patterns, whereas other similarity functions can be used to characterize the conceptual similarity between patterns.

The *grouping* step can be performed in a number of ways. The output clustering can be hard (a partition of the data into groups) or fuzzy (where each pattern has a variable degree of membership in each of the output clusters).

Data abstraction is the process of extracting a simple and compact representation of a data set. Here, simplicity is either from the perspective of automatic analysis or it is human –oriented. In the clustering context, a typical data abstraction is a compact description of each cluster, usually in terms of cluster prototypes or representative patterns.

2.3 Requirements of Clustering:

- Scalability
- Ability to deal with different types of attributes
- Discovery of clusters with arbitrary shape
- Minimal requirement for domain knowledge to determine input parameters
- Able to deal with noise and outliers
- Insensitive to order of input records
- High dimensionality
- Incorporation of user-specified constraints
- Interpretability and usability

2.4 Typical Applications:

- As a Stand-alone tool to get insight into data distribution
- As a preprocessing step for other algorithms

2.5 General Applications of Clustering:

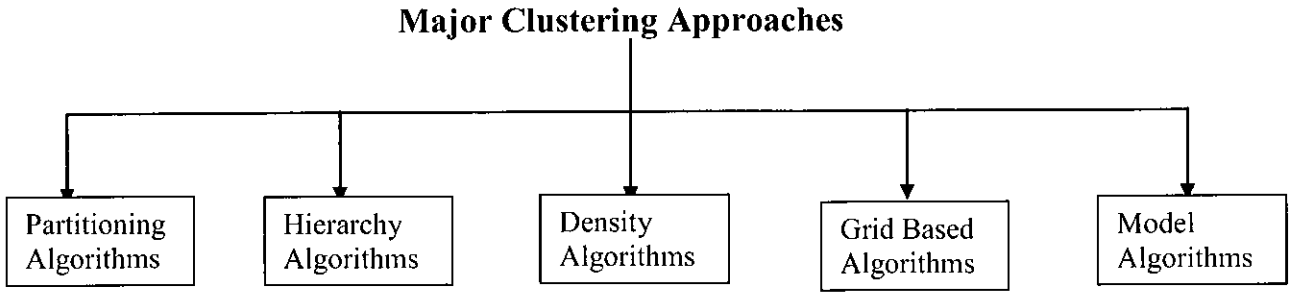
- Pattern Recognition
- Spatial Data Analysis

  -Detect spatial clusters and explain them in spatial data mining
- Image Processing
- Economic Science

  -Especially market research
- WWW

  -Document Classification

2.6 Classification



2.6.1 Partitioning Algorithms:

Construct various partitions and then evaluate them by some criterion. Construct a partition of a database **D** of **n** objects into a set of **k** clusters.

Given a *k*, find a partition of k clusters that optimizes the chosen partitioning criterion.

Global optimal: exhaustively enumerate all partitions.

Heuristic methods: k-means and k-medoids algorithms

k-means:

Each cluster is represented by the center of the cluster.

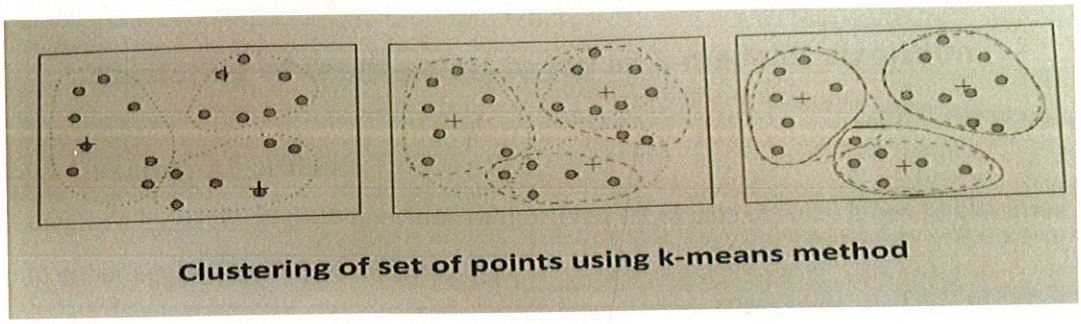


Fig 3

k-medoids or* PAM *(partition around medoids):

Basically, the algorithm finds the center of a cluster and takes the element closest to the center as the "Medoid" – means the most centrally located object in a cluster. Starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering. It is performed based on the principle of minimizing the sum of the dissimilarities between each object. PAM works effectively for small data sets, but does not scale well for large data sets.

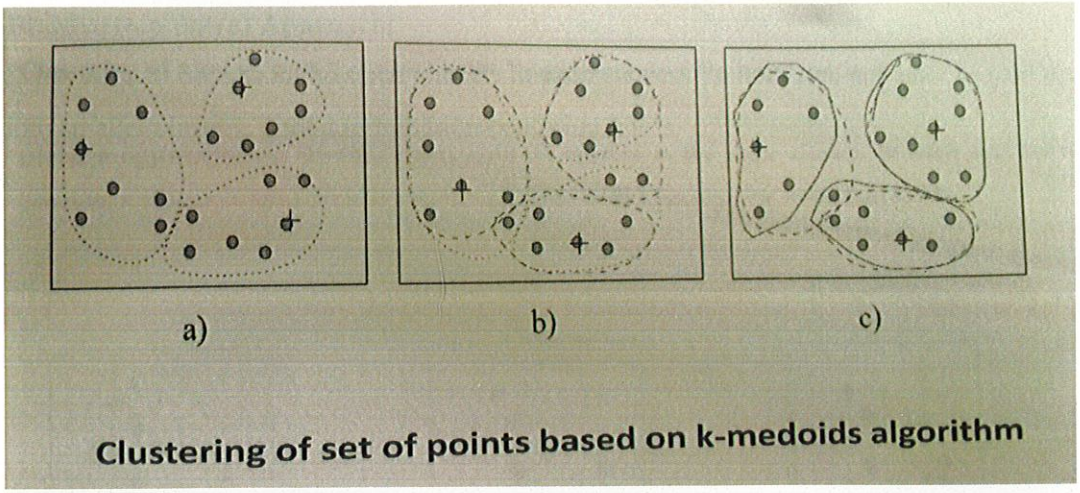


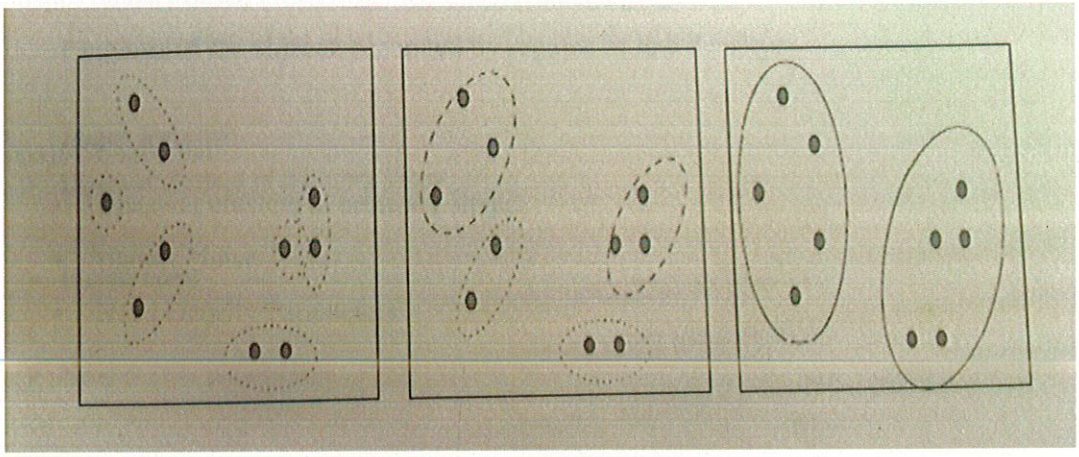
Fig 4

2.6.2 Hierarchy Algorithms:

It creates a hierarchical decomposition of the given set of data objects. It can be classified into either **agglomerative** or **divisive.**

Agglomerative (bottom-up) approach:

Set each object as an individual cluster or group and merges the objects or groups close to one another, until all of the groups are merged into one (the topmost level of the hierarchy).



Fig

Divisive (top down) Approach:

Starts with all objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until a termination condition holds.

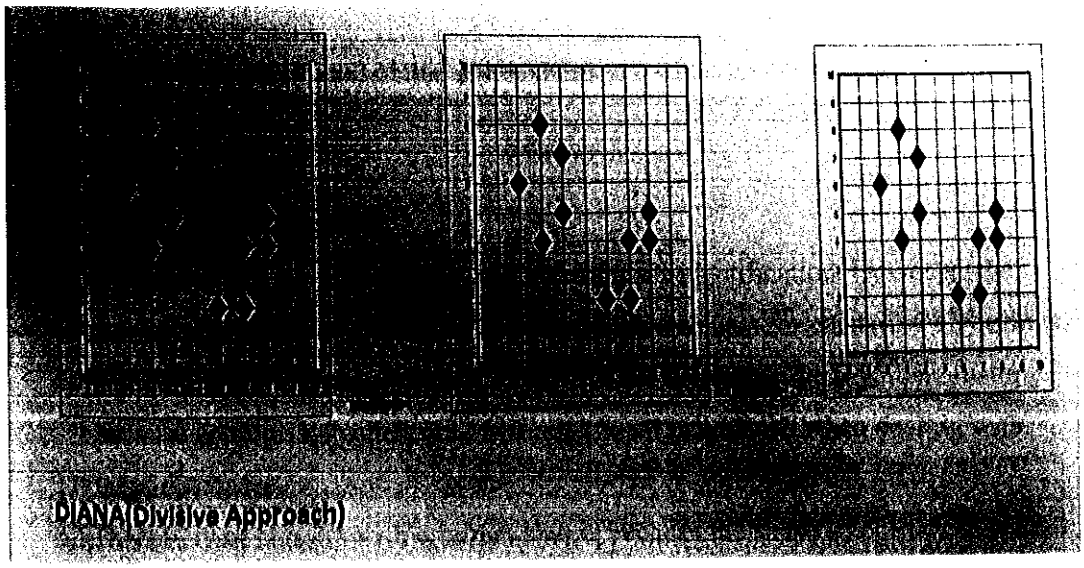


Fig 6

2.6.3 Density-Based Approach:

It is based on connectivity and Density functions. To discover the clusters with arbitrary shapes density based clustering methods have been proposed. Clusters are regarded as the dense regions of the objects, separated by regions of low density.

Major Features:

- Discover clusters of arbitrary shape.
- Handle Noise
- Need Density Parameters

Two Parameters:

- Maximum radius of the neighborhood.
- MinPts: Minimum number of points in a neighborhood of that point.

ε neighborhood of that point: Nε(p):{q ∈ D | dist(p,q) <= ε}

Example: **DBSCAN Algorithm**

Clustering Algorithms are attractive for the task of class identification. However, the application to large databases raises the following requirements for clustering algorithms:

1. Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.
2. Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.
3. Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects.

DBSCAN refers to **Density Based Spatial Clustering Of Applications with Noise.** It relies on a density-based notion of cluster: A *cluster* is defined as a maximal set of density-connected points. It was able to discover clusters of arbitrary shape in spatial databases with noise.

When looking at the sample set of points depicted in figure, we can easily and unambiguously detect clusters of points and noise points not belonging to any of those clusters.

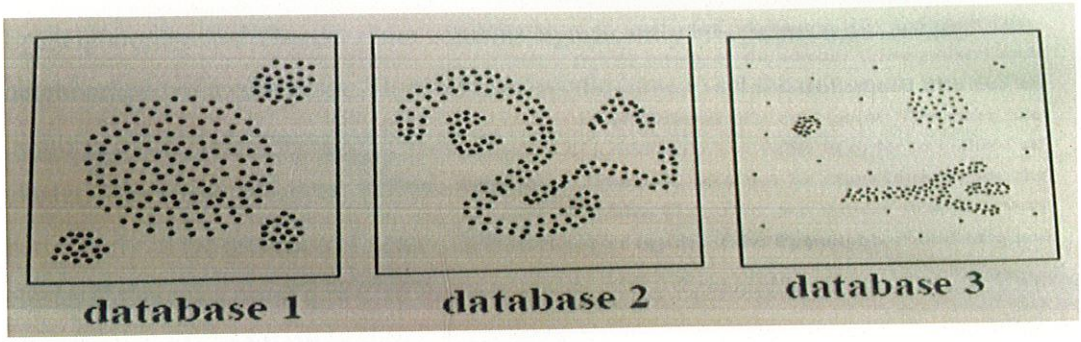


Fig 7

The main reason why we recognize the clusters is that within each cluster we have a typical density of points which is considerably higher than outside of the cluster. Furthermore, the density within the areas of noise is lower than the density in any of these clusters.

In the following, we try to formalize this intuitive notion of "clusters" and "noise" in a database D of points of some k-dimensional space S. Note that both, our notion of clusters and our algorithm DBSCAN, apply as well to 2D or 3D Euclidean space as to some high dimensional feature space. The key idea is that for each point of a cluster the neighborhood of a given radius has to contain at least a min no of points, i.e. the density in the neighborhood has to exceed some threshold. The shape of a neighborhood is determined by the choice of a distance function for tow points, p and q, denoted by dist (p, q). Note that our approach works with any distance function so that an appropriate function can be chosen for some given application. For the purpose of proper visualization, all examples will be in 2D space using the Euclidean distance.

Definition 1: (Eps neighborhood of a point) The Epsneighborhood of a point p, denoted by NEps(p), is defined by

NEps(p)={q belongs to D | dist(p,q) < Eps}

A naïve approach could require for each point in a cluster that there are at least a minimum number (MinPts) of points in an Eps-neighborhood of that point. However, this approach fails because there are two kinds of points in a cluster, points inside of a cluster(core points) and points on the border of the cluster(border points). In general, an

Epsneighborhood of a border point contains significantly less points than an Eps-neighborhood of a core point. Therefore, we would have to set the minimum number of points to a relatively low value in order to include all points belonging to the same cluster. This value, however, will not be characteristic for the respective cluster – particularly in the presence of noise. Therefore, we require that for every point p in a cluster C there is a point q in C so that p is inside of the Epsneighborhood of q and Eps (q) contains at least MinPts points.

Definition 2: (directly density-reachable) A point p is directly density-reachable from a point q w.r.t Eps, MinPts if

1. P belongs to NEps(q) and
2. |NEps(q)| >= MinPts (core point condition)

Obviously, directly density-reachable is symmetric for pairs of core points. In general, however, it is not symmetric if one core point and one border point are involved. Figure shows the asymmetric case.

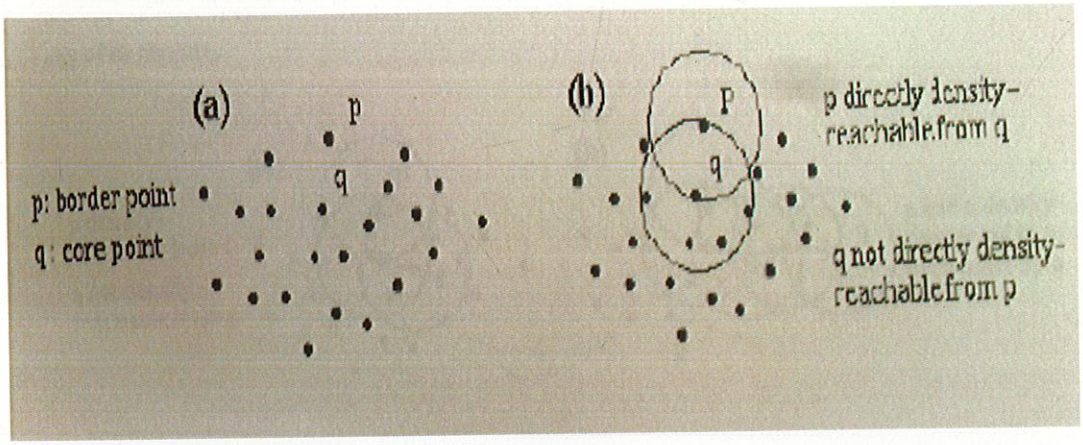


Fig 8

Definition 3: (density-reachable) A point p is density reachable from a point q w.r.t Eps and MinPts if there is a chain of points p1,….,pn ,p1=q , pn=p such that pi+1 is direct density-reachable. This relation is transitive, but it is not symmetric. Figure depicts the relations of some sample points , and in particular, the asymmetric case. Although not

symmetric in general, it is obvious that density-reachability is symmetric for core points. Two border points of the same cluster C are possibly not density reachable from each other because the core point condition might not hold for both of them. However, there must be a core point in C from which both border points of C are density-reachable . Therefore, we introduce the notion of density-connectivity which covers this relation of border points.

Definition 4: (density connected) A point p is density connected to a point q w.r.t Eps and MinPts if there is a point o such that both, p and q, are density reachable from o w.r.t Eps and MinPts. Density-connectivity is a symmetric relation. For density reachable points, the relation of density –connectivity is also reflexive.

Now, we are able to define our density based notion of a cluster. Intuitively, a cluster is defined to be a set of density connected points which is maximal w.r.t density-reachability.

Noise will be defined relative to a given set of clusters. Noise is simply the set of points in D not belonging to any of its clusters.

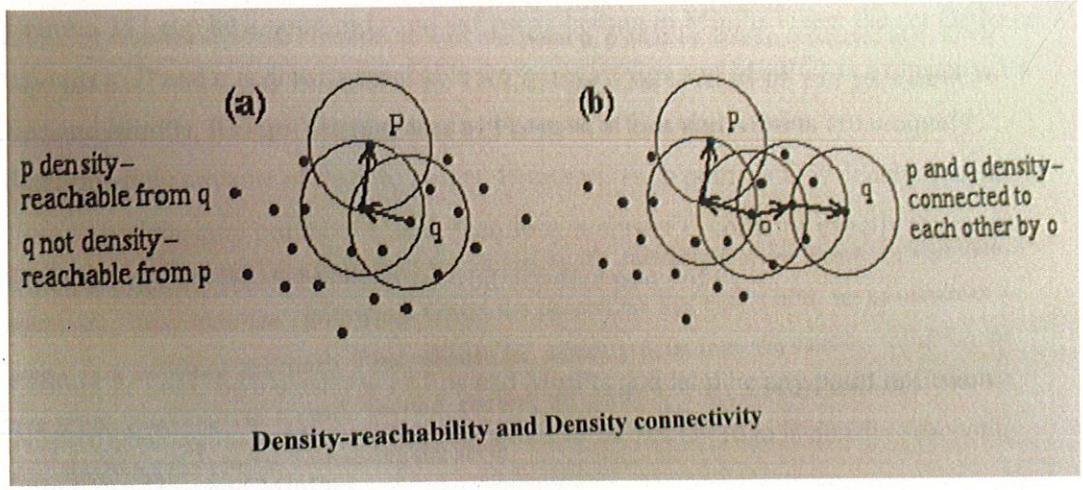


Fig 9

Definition 5: (cluster) Let D be a database of points. A cluster C w.r.t Eps and MinPts is a non empty subset of D satisfying the following conditions:

1. p,q :if p belongs to C and q is density reachable from p w.r.t Eps and MinPts , then q belongs to C.

2. For every p,q belonging to C :p is density connected to q w.r.t Eps and MinPts(Connectivity).

Definition 6: (noise) Let C1, ….. , Ck be the clusters of the database D w.r.t parameters Epsi and MinPtsi, i=1,…..,k. Then, we define the noise as the set of points in the database D not belonging to any cluster Ci ,i.e.

Noise ={p belongs to D | For Every i:p does not belong to Ci}.

Note that a cluster C w.r.t Eps and MinPts contains at least MinPts points because of the following reasons. Since C contains at least one point p, p must be density-connected to itself via some point o(which may be equal to p). Thus, at least o has to satisfy the core point condition and, consequently, the Eps-Neighborhood of o contains at least MinPts points.

Lemma 1: Let p be a point in D and |NEps(p)| belong to MinPts. Then the set O={o | o belongs to D and o is density reachable from p w.r.t Eps and MinPts} is a cluster w.r.t Eps and MinPts. It is not obvious that a cluster w.r.t Eps and MinPts are uniquely determined by any one of its core points. However, each point in C is density reachable from any of the core points of C and, therefore, a cluster C contains exactly the points which are density reachable from an arbitrary core point of C.

Lemma 2: Let C be a cluster w.r.t Eps and MinPts and let p be any point in C with |NEps(p)| belonging to MinPts. Then C equals to the set O={o |o is density reachable from p w.r.t Eps and MinPts}.

2.6.4 Grid-Based:

- Based on a multiple level granularity structure.
- No distance computations.
- Clustering is performed on summaries and not individual objects; complexity is usually O (#-populated-grid-cells) and not O (# objects).
- Easy to determine which clusters are neighboring
- Shapes are limited to union of grid-cells.

Basic Grid –Based Algorithm:

- Define a set of grid-cells.
- Assign objects to the appropriate grid cell and compute the density of each cell.
- Eliminate cells, whose density is below a certain threshold t.
- Form clusters from continuous (adjacent) groups of dense cells (usually minimizing a given objective function).

Example:

STING: A Statistical Information Grid Approach

- The spatial area is divided into rectangular cells.
- There are several levels of cells corresponding to different levels of resolution.
- Each cell at a high level is partitioned into a number of smaller cells in the next lower level.
- Statistical info of each cell is calculated and stored beforehand and is used to answer queries.
- Parameters of higher level cells can be easily calculated from parameters of lower level cell: count, mean ,sd, min, max
- Type of distribution- normal, uniform etc.
- Use a top-down approach to answer spatial data queries.

Advantages:

- Query –independent, easy to parallelize, incremental update.
- O (K), where K is the number of grid cells at the lowest level.

Disadvantages:

- All the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected.

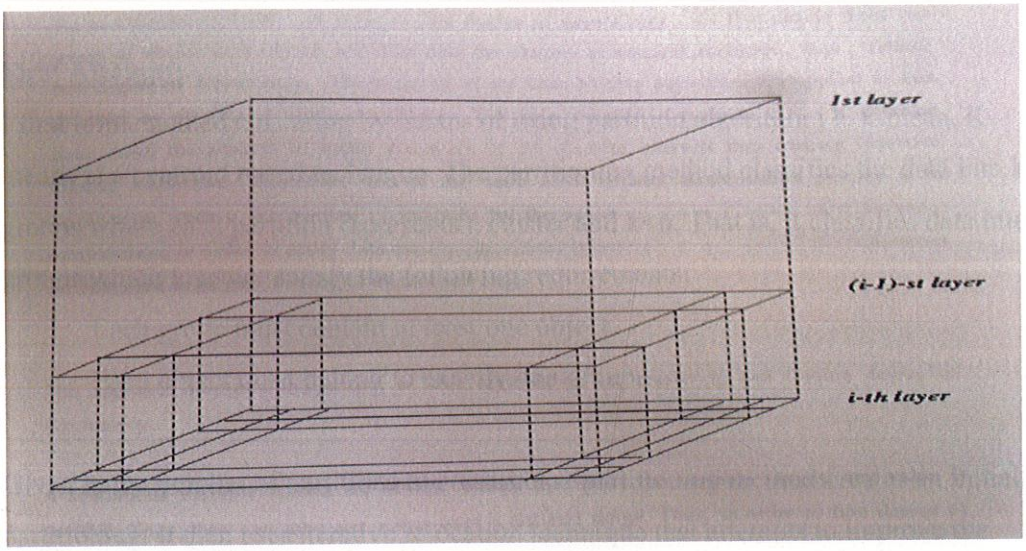


Figure 10: STING

2.6.5 Model- Based:

A model is hypothesized for each of the clusters and the idea is to find the best fit of that model to each other.

CHAPTER 3

3 Design and Implementation:

3.1 Modular Description

First Attempt:

I first implemented clustering by means of using partition algorithm i.e. k-mean. K-means is a centroid based technique. The partitioning method classifies the data into k-groups where each partition represents a cluster and k<n. That is, it classifies data into k-groups which together satisfy the following requirements:

1. Each group must contain at least one object.
2. Each object must belong to exactly one group.

Given k, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses iterative relocation technique that attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are "close" or related to each other, whereas objects of different clusters are "far apart" or very different.

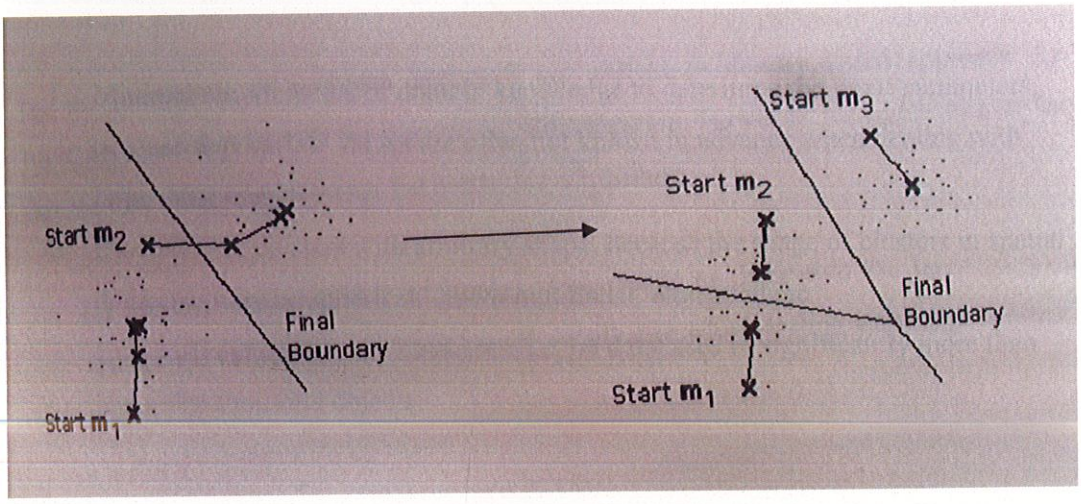


Fig 11

These partitioning clustering methods work well for finding spherical shaped clusters in small to medium partition sized databases.

Weakness:

- Applicable only when mean is defined.
- Need to specify k, the number of clusters, in advance.
- Unable to handle noisy data and outliers
- K mean sensitive to outliers, extremely large value may distort the data distribution.
- Not suitable to discover clusters with non-convex shapes.

Second Attempt:

Due to the limitations of k-means approach, we shifted to density based approach, i.e. we used **DBSCAN** algorithm in 1-D for clustering identification. However, the application to large databases raises the following requirements for clustering algorithms:

1. Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.
2. Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.
3. Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects.

Quality Threshold:

This algorithm requires the apriori specification of the threshold distance within the cluster and the minimum number of elements in each cluster. Now from each data point we find all its candidate data points. Candidate data points are those which are within the range of the threshold distance from the given data point. This way we find the candidate data points for all data point and choose the one with large number of candidate data points to form cluster. Now data points which belongs to this cluster is removed and the same procedure is repeated with the reduced set of data points until no more cluster can be formed satisfying the minimum size criteria. This algorithm sounds like a predecessor of **DBSCAN** and was hence implemented for comparison.

DBSCAN also "removes" points from further consideration, but simply by marking them as already assigned. That way you don't need to update the index; and it's sufficient to bulk-load it once in the beginning. You should be able to do this for QT, too. So unless , we can get the QT clustering efficiently by running DBSCAN with epsilon set to the QT clustering and MinPts=2 (although one would prefer higher values in proper DBSCAN).

3.2 Algorithm:

First Attempt:

The implementation of k-means required various steps:

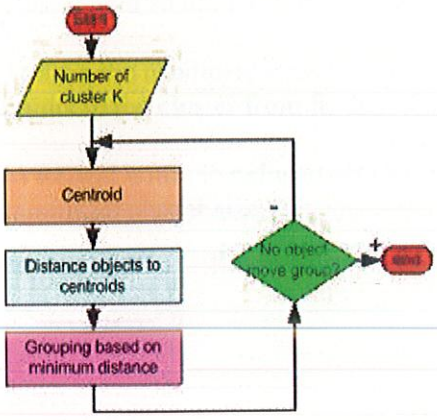


Fig 12

- First, we take n number of points.
- We allocate any 'k' no. of points as centroid.
- 'K' is less than or equal to 'n'.
- We calculate the distance of each 'n' points from each 'k' centroid.
- We save the distance data in a matrix.
- We compare the distance of a point to each centroid.
- And we allocate '1' if the distance is minimum , else '0' in another matrix.
- We again calculate the new values of centroid considering only points with value in matrix equal to '1' in respective rows.
- We repeat the process till the value of coordinates of centroid become constant.
- The final groups of points are the required clusters.

Second attempt:

Algorithmic steps for QT clustering

1) Initialize the threshold distance allowed for clusters and the minimum cluster size.

2) Build a candidate cluster for each data point by including the closest point, the next closest, and so on, until the distance of the cluster surpasses the threshold.

3) Save the candidate cluster with the most points as the first true cluster, and remove all points in the cluster from further consideration.

4) Repeat with the reduced set of points until no more cluster can be formed having the minimum cluster size.

In order to implement **DBSCAN,** the following algorithm was used:

Ideally, we would have to know the appropriate parameters Eps and MinPts of each cluster and at least one point from the respective cluster. Then we could retrieve all points that are density-reachable from the given point using the correct parameters. But there is no easy way to get this information in advance for all clusters of the database. However, there is simple and effective heuristic to determine the parameters Eps and MinPts of the "thinnest" i.e. the least dense, cluster in the database. Therefore, DBSCAN uses global values for Eps and MinPts, i.e. the same values for all clusters. The density parameters of the "thinnest" cluster are good candidates for these global parameter values specifying the lowest density which is not considered to be noise.

To find a cluster, DBSCAN starts with an arbitrary point p and retrieves all points density-reachable from p w.r.t Eps and MinPts. DBSCAN needs two parameters, Eps and MinPts. However, experiments indicate that the k-dist graphs for k>4 do not significantly differ from the 4-dist graph and, therefore , they need considerably more computation. Therefore, we eliminate the parameter MinPts by setting it to 4 for all databases(for 1-D data). If p is a core point, this procedure yields a cluster w.r.t Eps and MinPts(see Lemma 2).If p is a border point, no points are density reachable from p and DBSCAN visits the next point of the database. Since we use global values for Eps and MinPts, DBSCAN may merge two clusters according to definition 5 into one cluster, if two clusters of different density are "close" to each other. Let the distance between two sets of points be defined as

Dist(S1,S2)=min{dist(p,q) | p belongs to S1, q belongs to S2}

Then, two sets of points having at least the density of the thinnest cluster will be separated from each other only if the distance between the two sets is larger than Eps. Consequently, a recursive call of DBSCAN may be necessary for the detected clusters with a higher value for MinPts. This is, however , no disadvantage because the recursive application of DBSCAN yields an elegant and very efficient basic algorithm.

Furthermore, the recursive clustering of the points of cluster is only necessary under conditions that can be easily detected.

- Arbitrary select a point p.
- Retrieve all points' density reachable from p w.r.t Eps and MinPts.
- If p is a core point (points inside of the cluster), a cluster is formed.
- If p is not a core point, no points are density reachable from p and DBSCAN visits the next point of the database.
- Continue the process until all of the points have been processed.

3.3 CODE

Quality Threshold:

```
using System;
using System.Collections.Generic;

struct Point
{
   public int X, Y;

   public Point(int x, int y)
   {
      this.X = x;
      this.Y = y;
   }

   public override string ToString()
   {
      return String.Format("({0}, {1})", X, Y);
   }

   public static int DistanceSquared(Point p1, Point p2)
   {
      int diffX = p2.X - p1.X;
      int diffY = p2.Y - p1.Y;
      return diffX * diffX + diffY * diffY;
```

```
    }
}

static class QT
{
    static void Main()
    {
        List<Point> points = new List<Point>();
        // sample data
        points.Add(new Point(0, 100));
        points.Add(new Point(0, 200));
        points.Add(new Point(0, 275));
        points.Add(new Point(100, 150));
        points.Add(new Point(200, 100));
        points.Add(new Point(250, 200));
        points.Add(new Point(0, 300));
        points.Add(new Point(100, 200));
        points.Add(new Point(600, 700));
        points.Add(new Point(650, 700));
        points.Add(new Point(675, 700));
        points.Add(new Point(675, 710));
        points.Add(new Point(675, 720));
        points.Add(new Point(50, 300));
        points.Add(new Point(76, 324));
        points.Add(new Point(50, 325));
        points.Add(new Point(47, 350));
        points.Add(new Point(50, 313));
        points.Add(new Point(50, 300));
        points.Add(new Point(76, 1124));
        points.Add(new Point(50,1125));
        points.Add(new Point(47, 1150));
        points.Add(new Point(50, 1113));
        points.Add(new Point(800, 400));
        points.Add(new Point(820, 413));
        points.Add(new Point(805, 467));
        points.Add(new Point(767, 410));
        points.Add(new Point(559, 400));
        points.Add(new Point(50, 800));
        points.Add(new Point(3000, 99999));

        double maxDiameter = 150.0;

        List<List<Point>> clusters = GetClusters(points, maxDiameter);
        Console.Clear();
```

```
        // print points to console
        Console.WriteLine("The {0} points are :\n", points.Count);
        foreach(Point p in points) Console.Write(" {0} ", p);
        Console.WriteLine();

        // print clusters to console
        for(int i = 0; i < clusters.Count; i++)
        {
          int count = clusters[i].Count;
          string plural = (count != 1) ? "s" : "";
          Console.WriteLine("\nCluster {0} consists of the following {1} point{2} :\n", i +
1, count, plural);
          foreach(Point p in clusters[i]) Console.Write(" {0} ", p);
          Console.WriteLine();
        }

        Console.ReadKey();
    }

    static List<List<Point>> GetClusters(List<Point> points, double maxDiameter)
    {
        if (points == null) return null;
        points = new List<Point>(points); // leave original List unaltered
        List<List<Point>> clusters = new List<List<Point>>();

        while (points.Count > 0)
        {
            List<Point> bestCandidate = GetBestCandidate(points, maxDiameter);
            clusters.Add(bestCandidate);
        }

        return clusters;
    }

    /*
        GetBestCandidate() returns first candidate cluster encountered if there is more than
one
        with the maximum number of points.
    */
    static List<Point> GetBestCandidate(List<Point> points, double maxDiameter)
    {
        double maxDiameterSquared = maxDiameter * maxDiameter; // square maximum
diameter
        List<List<Point>> candidates = new List<List<Point>>(); // stores all candidate
clusters
        List<Point> currentCandidate = null; // stores current candidate cluster
```

```
        int[] candidateNumbers = new int[points.Count]; // keeps track of candidate
numbers to which points have been allocated
        int totalPointsAllocated = 0; // total number of points already allocated to candidates
        int currentCandidateNumber = 0; // current candidate number

        for(int i = 0; i < points.Count; i++)
        {
          if (totalPointsAllocated == points.Count) break; // no need to continue further
          if (candidateNumbers[i] > 0) continue; // point already allocated to a candidate
          currentCandidateNumber++;
          currentCandidate = new List<Point>(); // create a new candidate cluster
          currentCandidate.Add(points[i]); // add the current point to it
          candidateNumbers[i] = currentCandidateNumber;
          totalPointsAllocated++;

          Point latestPoint = points[i]; // latest point added to current cluster
          int[] diametersSquared = new int[points.Count]; // diameters squared of each
point when added to current cluster

          // iterate through any remaining points
          // successively selecting the point closest to the group until the threshold is
exceeded
          while (true)
          {
            if (totalPointsAllocated == points.Count) break; // no need to continue
further
            int closest = -1; // index of closest point to current candidate cluster
            int minDiameterSquared = Int32.MaxValue; // minimum diameter squared,
initialized to impossible value

            for (int j = i + 1; j < points.Count; j++)
            {
              if(candidateNumbers[j] > 0) continue; // point already allocated to a
candidate

              // update diameters squared to allow for latest point added to current cluster
              int distSquared  = Point.DistanceSquared(latestPoint, points[j]);
              if (distSquared > diametersSquared[j]) diametersSquared[j] = distSquared;

              // check if closer than previous closest point
              if (diametersSquared[j] < minDiameterSquared)
              {
                minDiameterSquared = diametersSquared[j];
                closest = j;
              }
            }
```

```
            // if closest point is within maxDiameter, add it to the current candidate and
mark it accordingly
            if ((double)minDiameterSquared <= maxDiameterSquared)
            {
              currentCandidate.Add(points[closest]);
              candidateNumbers[closest] = currentCandidateNumber;
              totalPointsAllocated++;
            }
            else // otherwise finished with current candidate
            {
              break;
            }
          }

          // add current candidate to candidates list
          candidates.Add(currentCandidate);
        }

        // now find the candidate cluster with the largest number of points
        int maxPoints = -1; // impossibly small value
        int bestCandidateNumber = 0; // ditto
        for(int i = 0; i < candidates.Count; i++)
        {
          if (candidates[i].Count > maxPoints)
          {
            maxPoints = candidates[i].Count;
            bestCandidateNumber = i + 1; // counting from 1 rather than 0
          }
        }

        // iterating backwards to avoid indexing problems, remove points in best candidate
from points list
        // this will automatically be persisted to caller as List<Point> is a reference type
        for(int i = candidateNumbers.Length - 1; i >= 0; i--)
        {
          if (candidateNumbers[i] == bestCandidateNumber) points.RemoveAt(i);
        }

        // return best candidate to caller
        return candidates[bestCandidateNumber - 1];
      }
    }
```

DBSCAN:

```
using System;
using System.Collections.Generic;
using System.Linq;


class Point
{
   public const int NOISE = -1;
   public const int UNCLASSIFIED = 0;
   public int A, B, Cluster_Id;
   public Point(int x, int y)
   {
      this.A = x;
      this.B = y;
   }
   public override string ToString()
   {
      return String.Format("({0}, {1})", A, B);
   }
   public static int DistanceSquared(Point p1, Point p2)
   {
      int diffA = p2.A - p1.A;
      int diffB = p2.B - p1.B;
      return diffA * diffA + diffB * diffB;
   }
}
static class DBSCAN
{
   static void Main()
   {
      List<Point> points = new List<Point>();
      // sample data
      points.Add(new Point(0, 100));
      points.Add(new Point(0, 200));
      points.Add(new Point(0, 275));
      points.Add(new Point(100, 150));
      points.Add(new Point(200, 100));
      points.Add(new Point(250, 200));
      points.Add(new Point(0, 300));
      points.Add(new Point(100, 200));
      points.Add(new Point(600, 700));
      points.Add(new Point(650, 700));
      points.Add(new Point(675, 700));
```

```
        points.Add(new Point(675, 710));
        points.Add(new Point(675, 720));
        points.Add(new Point(50, 300));
        points.Add(new Point(76, 324));
        points.Add(new Point(50, 325));
        points.Add(new Point(47, 350));
        points.Add(new Point(50, 313));
        points.Add(new Point(50, 300));
        points.Add(new Point(76, 1124));
        points.Add(new Point(50,1125));
        points.Add(new Point(47, 1150));
        points.Add(new Point(50, 1113));
        points.Add(new Point(800, 400));
        points.Add(new Point(820, 413));
        points.Add(new Point(805, 467));
        points.Add(new Point(767, 410));
        points.Add(new Point(559, 400));
        points.Add(new Point(50, 800));
        points.Add(new Point(3000, 99999));

        double eps = 100.0;
        int minPts = 3;
        List<List<Point>> clusters = GetClusters(points, eps, minPts);
        Console.Clear();
        // print points to console
        Console.WriteLine("The {0} points are :\n", points.Count);
        foreach (Point p in points) Console.Write(" {0} ", p);
        Console.WriteLine();
        // print clusters to console
        int total = 0;
        for (int i = 0; i < clusters.Count; i++)
        {
            int count = clusters[i].Count;
            total += count;
            string plural = (count != 1) ? "s" : "";
            Console.WriteLine("\nCluster {0} consists of the following {1} point{2} :\n", i +
1, count, plural);
            foreach (Point p in clusters[i]) Console.Write(" {0} ", p);
            Console.WriteLine();
        }
        // print any points which are NOISE
        total = points.Count - total;
        if (total > 0)
        {
            string plural = (total != 1) ? "s" : "";
            string verb = (total != 1) ? "are" : "is";
```

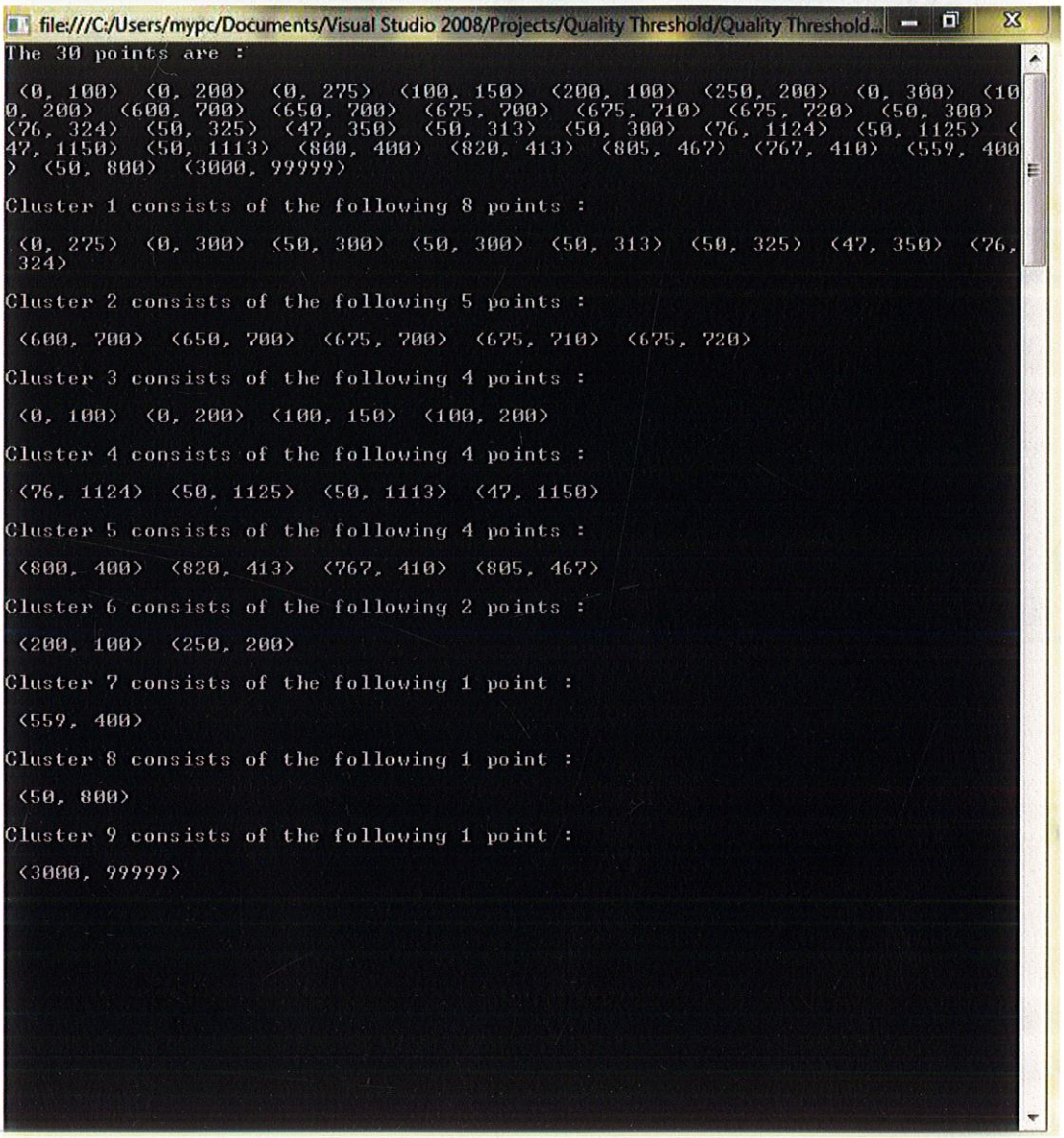
```
            Console.WriteLine("\nThe following {0} point{1} {2} NOISE :\n", total, plural,
verb);
            foreach (Point p in points)
            {
               if (p.Cluster_Id == Point.NOISE) Console.Write(" {0} ", p);
            }
            Console.WriteLine();
         }
         else
         {
            Console.WriteLine("\nNo points are NOISE");
         }
         Console.ReadKey();
      }
      static List<List<Point>> GetClusters(List<Point> points, double eps, int minPts)
      {
         if (points == null) return null;
         List<List<Point>> clusters = new List<List<Point>>();
         eps *= eps; // square eps
         int clusterId = 1;
         for (int i = 0; i < points.Count; i++)
         {
            Point p = points[i];
            if (p.Cluster_Id == Point.UNCLASSIFIED)
            {
               if (ExpandCluster(points, p, clusterId, eps, minPts)) clusterId++;
            }
         }
         // sort out points into their clusters, if any
         int maxClusterId = points.OrderBy(p => p.Cluster_Id).Last().Cluster_Id;
         if (maxClusterId < 1) return clusters; // no clusters, so list is empty
         for (int i = 0; i < maxClusterId; i++) clusters.Add(new List<Point>());
         foreach (Point p in points)
         {
            if (p.Cluster_Id > 0) clusters[p.Cluster_Id - 1].Add(p);
         }
         return clusters;
      }
      static List<Point> GetRegion(List<Point> points, Point p, double eps)
      {
         List<Point> region = new List<Point>();
         for (int i = 0; i < points.Count; i++)
         {
            int distSquared = Point.DistanceSquared(p, points[i]);
            if (distSquared <= eps) region.Add(points[i]);
         }
```

```
        return region;
    }
    static bool ExpandCluster(List<Point> points, Point p, int clusterId, double eps, int
minPts)
    {
        List<Point> seeds = GetRegion(points, p, eps);
        if (seeds.Count < minPts) // no core point
        {
            p.Cluster_Id = Point.NOISE;
            return false;
        }
        else // all points in seeds are density reachable from point 'p'
        {
            for (int i = 0; i < seeds.Count; i++) seeds[i].Cluster_Id = clusterId;
            seeds.Remove(p);
            while (seeds.Count > 0)
            {
                Point currentP = seeds[0];
                List<Point> result = GetRegion(points, currentP, eps);
                if (result.Count >= minPts)
                {
                    for (int i = 0; i < result.Count; i++)
                    {
                        Point resultP = result[i];
                        if (resultP.Cluster_Id == Point.UNCLASSIFIED || resultP.Cluster_Id ==
Point.NOISE)
                        {
                            if (resultP.Cluster_Id == Point.UNCLASSIFIED) seeds.Add(resultP);
                            resultP.Cluster_Id = clusterId;
                        }
                    }
                }
                seeds.Remove(currentP);
            }
            return true;
        }
    }
}
```

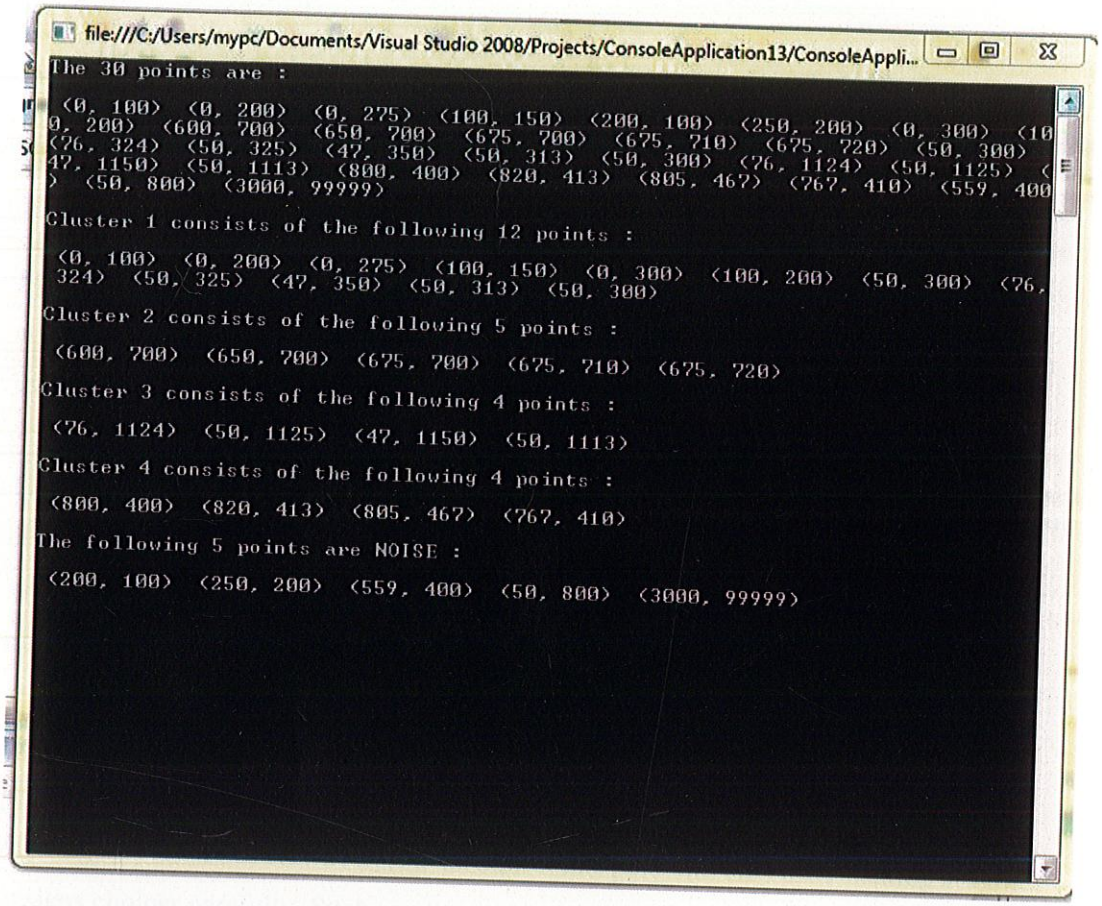
CHAPTER 4

4 Results

4.1 Quality Threshold:



4.2 DBSCAN:


file:///C:/Users/mypc/Documents/Visual Studio 2008/Projects/ConsoleApplication13/ConsoleAppli...
The 30 points are :
(0, 100) (0, 200) (0, 275) (100, 150) (200, 100) (250, 200) (0, 300) (10
0, 200) (600, 700) (650, 700) (675, 700) (675, 710) (675, 720) (50, 300)
(76, 324) (50, 325) (47, 350) (50, 313) (50, 300) (76, 1124) (50, 1125) (
47, 1150) (50, 1113) (800, 400) (820, 413) (805, 467) (767, 410) (559, 400
) (50, 800) (3000, 99999)
Cluster 1 consists of the following 12 points :
(0, 100) (0, 200) (0, 275) (100, 150) (0, 300) (100, 200) (50, 300) (76,
324) (50, 325) (47, 350) (50, 313) (50, 300)
Cluster 2 consists of the following 5 points :
(600, 700) (650, 700) (675, 700) (675, 710) (675, 720)
Cluster 3 consists of the following 4 points :
(76, 1124) (50, 1125) (47, 1150) (50, 1113)
Cluster 4 consists of the following 4 points :
(800, 400) (820, 413) (805, 467) (767, 410)
The following 5 points are NOISE :
(200, 100) (250, 200) (559, 400) (50, 800) (3000, 99999)

CHAPTER -5

Future Work and Conclusion

For the process of routing on any network, it was assessed that an efficient clustering algorithm needs to be first implemented on a network so as to remove or at least reduce the overheads involved.

The Clustering Techniques inmplemented and studied by me help in creating an efficient clustering on the network. Further, more known or unknown algorithms may be implemented that reduce the overheads to a better degree and making routing more efficient.

In Summary, clustering is an interesting, useful and challenging problem. It has great potential in applications like pattern recognition, image segmentation, and information filtering and retrieval. However, it is to exploit this potential only after making several designs choices carefully. Such an attempt was made by me.

BIBLIOGRAPHY

- An Overview of Clustering Methods by Sorin Istrail Informatics Research Celera Genomics.

- Clustering methods by Lior Rokach ,Department of Industrial Engineering Tel-Aviv University and Oded Maimon ,Department of Industrial Engineering Tel-Aviv University.

- Data Clustering and Its Applications by Raza Ali ,Usman Ghani, Aasim Saeed.

- Data Clustering Techniques by Periklis Andritsos University of Toronto March 11, 2002.

- Data mining concepts and techniques by Jiawei Han and Micheline Kamber.

- Density-based clustering algorithms – DBSCAN and SNN Version 1.0, 25.07.2005 Adriano Moreira, Maribel Y. Santos and Sofia Carneiro University of Minho –Portugal.