University of Alberta

Density-Based Clustering of Spatial Data in the presence of Physical Constraints

by

Chi-Hoon Lee

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science.**

Department of Computing Science

Edmonton, Alberta
Fall 2002

University of Alberta

Library Release Form

Name of Author: Chi-Hoon Lee

Title of Thesis: Density-Based Clustering of Spatial Data in the presence of Physical Constraints

Degree: Master of Science

Year this Degree Granted: 2002

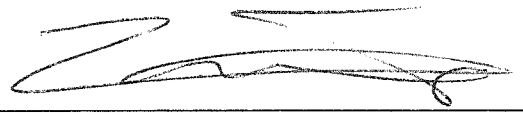
Chi-Hoon Lee
607F, Michener Park
Edmonton, Alberta
Canada, T6H 5A1

Date: 26 June, 2002

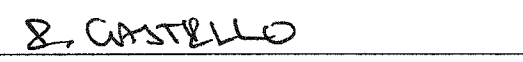
University of Alberta

Faculty of Graduate Studies and Research

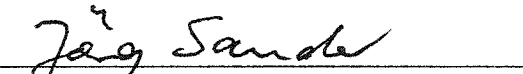
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Density-Based Clustering of Spatial Data in the presence of Physical Constraints** submitted by Chi-Hoon Lee in partial fulfillment of the requirements for the degree of **Master of Science**.



Dr. Osmar R. Zaïane



Dr. Ignacio Castillo



Dr. Jörg Sander

Dr. Terry Caelli

Date: 26 June 2002

Abstract

Clustering spatial data is a well-known problem that has been extensively studied. Grouping similar data in large 2-dimensional spaces to find hidden patterns or meaningful sub-groups has many applications such as in satellite imagery, geographic information systems, medical image analysis, marketing, computer vision, etc. Although many methods have been proposed in the literature, very few have considered constraints such as the fact that physical obstacles and bridges linking clusters may have significant consequences on the effectiveness of the clustering. Taking into account these constraints during the clustering process is costly, and the effective modeling of the constraints is of paramount importance for good performance. In this thesis, we define the clustering problem in the presence of constraints – obstacles and crossings – and investigate its efficiency and effectiveness for large databases. In addition, we introduce a new approach to model these constraints. We propose a strategy to prune the search space and reduce the number of polygons to test during clustering. Note that the approach minimizes user involvement for automatic procedures. We devise a density-based clustering algorithm, *DBCluC*, which takes advantage of our constraint modeling to efficiently cluster data objects while considering all physical constraints. The algorithm can detect clusters of arbitrary shape and is insensitive to noise and the input order. Its average running complexity is $O(N\log N)$ where N is the number of data points.

Acknowledgements

The first person I would like to thank is my supervisor Dr. Osmar R. Zaiane. I have been with him since 2000 when I started my MSc program. During these years I have known Dr. Osmar R. Zaiane as a sympathetic and principle-centered person. His overly enthusiasm and integral view on research and his mission for providing "only high-quality work and not less", has made a deep impression on me. I owe him lots of gratitude for having me shown Data mining research areas. He could not even realize how much I have learned from him.

I would like to thank Dr. Ryan Hayward who kept an eye on the progress of my work and always was available when I needed his advises as a professor and friend. Besides of being an excellent professor, Dr. Ryan Hayward was a good friend to me. I am really glad that I have come to get know Dr. Ryan Hayward in my life.

I would also like to thank the other members of my MSc committee who took effort in reading and providing me with valuable comments on earlier versions of this thesis: Dr. Ignacio Castillo, Dr. Jörg Sander, and Dr. Terry Caelli. I thank you all.

My DB lab colleagues from the Department support me in my research work. I want to thank them for all their help.

I have to thank my parents for their patience including financial support. My sister Jung-hi Lee encouraged me to pursue my scholar passion in spite of some constraints. Especially, I thank my parents, my wife's parents, my niece Hyun-Jee Lee, and my brother-in-law for taking care of my lovely son Samuel Jungsoo Lee. I would not forget their wish and support for my work and my health in my entire life.

My true heart also goes to my grandmother who has taken care of me for long time. She has been my best friend. I would like to express great gratitude to my uncle. Despite his challenged circumstances, he has done great jobs as a prominent teacher and respectable father.

Especially, I would like to give my special thanks to my wife Meejung Cheigh whose patient love enables me to complete this work. Her priceless advice and love encouraged me to go ahead with my thesis. She has provided me with endless lessons.

To GOD and my family

Contents

List of Figures

List of Tables

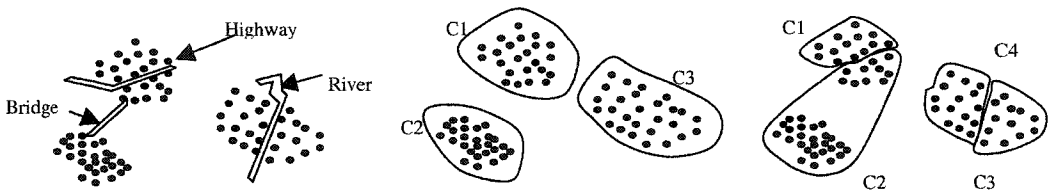
Chapter 1

Introduction

Unsupervised classification, also known as the clustering of objects into groups such that the similarity of objects in a group is maximized while the similarity between objects of different groups is minimized, is an interesting and influential problem that has attracted the attention of statisticians for many years because of its numerous potential applications. Recently, we have witnessed a resurgence of interest in new clustering techniques in the data mining community, and many effective and efficient methods have been proposed in the machine learning and data mining literature [28, 35, 7, 37, 49, 43, 52, 15, 2, 24, 8, 5, 38, 48, 42, 1, 41, 31]. The rapid increase in the availability of digitized spatial data has prompted considerable research into what is known as spatial data mining [32, 14]. Clustering analysis in two dimensional space, that is learning from data, in a two dimensional space, is considered spatial data mining and has wide range of applications in areas such as geographic information systems, pattern recognition, medical imaging, marketing analysis, weather forecasting, and current analysis.

1.1 Clustering in the presence of constraints

Clustering spatial data in two dimensional planar space has been an active research area, with most of the research focusing on effectiveness and scalability. To cluster data objects in *n-dimensions*, the *closeness* or so-called similarity is measured between data objects. There are various metrics [6] to measure dissimilarity – Manhattan, Euclidean, Maximum, weighted Euclidean,



(a) Data objects and constraints (b) Clusters ignoring constraints (c) Clusters with constraints

Figure 1.1: Clustering data objects with constraints

weighted Manhattan, and Ellipsoid. The application determines which metric is employed, depending on its characteristic. The most common metric in *n-dimensional* spatial database domains between two data objects o and q is *Euclidean* distance, which is defined as

$$distance(o, q) = \sqrt{\Sigma_{k=1}^{n}(o_k - q_k)^2} \tag{1.1}$$

The Euclidean distance between two data objects in turn indicates proximity in the spatial data mining context. Hence, a cluster groups data objects if the distances between data objects are *small*.

As pointed out earlier, clustering techniques [28, 35, 7, 37, 49, 43, 52, 15, 2, 24, 8, 5, 38, 48, 42, 1, 41, 31] introduced in the data mining literature have focused on performance in terms of effectiveness and efficiency for large databases. However, almost none of them have taken into account constraints that may be present in the data or constraints on the clustering procedure. These constraints have a significant influence on the quality and correctness of the clustering process of large amount of spatial data. In medical imaging, for example, while 2 points could be close together according to a distance measure, they should be restrained from being clustered together due to physical or biological constraints. In a GIS application studying the movement of pedestrians to identify optimal bank machine placements, for example, the presence of a highway hinders the movement of pedestrians and should be considered as an obstacle, while a pedway over this highway could be considered as a bridge. Figure 1.1 illustrates an example of a clustering problem in the presence of constraints.

Two constraint entities in Figure 1.1 – a highway and a river – force natural clusters to be isolated if the entities are correctly interpreted by clustering techniques. These entities have a disconnectivity functionality, which disconnects the closeness between data objects. The example in Figure 1.1(b) shows false-clusters in which each false-cluster groups data objects that should have not belonged to the cluster. In addition, an entity such as a bridge has a connectivity functionality which groups distant data objects together. Examples are depicted in Figure1.1(c). Generally, the nearest cluster assignment for each data object, i.e. close data objects grouped together, is not applicable to the clustering problem in the presence of constraints.

In this thesis, we propose a clustering algorithm, DBCluC, in the presence of constraints. However, large and complicated constraints ultimately degrade the performance of an algorithm. As a result, this thesis also proposes new modeling schemes for constraints using simple polygons and new concepts – Entry points and Entry edges.

1.2 Thesis organization

The remainder of this thesis is organized as follows: in Chapter 2, we briefly introduce novel clustering algorithms which do not consider constraints according to their taxonomy in order to familiarize readers with the state-of-the-art in clustering techniques. Chapter 2 also presents the notions of reachability and connectivity needed in the expansion process of DBSCAN [15] since the same reachability idea is adopted in DBCluC and provides the motivating concepts significant to this study. An educational tool for clustering algorithms is introduced in the same chapter. In Chapter 3, two clustering algorithms that take into account physical constraints such as obstacles are presented. In addition, a study that is focused on operational constraints is introduced. In Chapter 4, we show how we model the constraints – obstacles and crossings – using a simple polygon and new concepts – Entry edges and Entry points – and illustrate how the edges of the polygons are reduced to improve performance. The main clustering algorithm, DBCluC, which considers constraints

during the clustering, is introduced. The complexity analysis of DBCluC and the modeling schemes of constraints is presented in Chapter 5. Chapter 6 shows the performance of this algorithm and its clustering results by varying the number/difficulty of data objects and constraints. Finally, Section 6 concludes this study with some discussion of future work.

Chapter 2

Data Clustering

In this chapter we discuss clustering, also known in the literature as Unsupervised classification algorithms. Clustering multi-dimensional data objects is an interesting problem that has attracted the attention of statisticians for many years due to its valuable applications. Clustering analysis for data in multi-dimensional spaces is considered to be a form of spatial data mining and has numerous applications in geographic information systems, pattern recognition, medical imaging, marketing analysis, weather forecasting, ocean currents analysis, etc. Clustering approaches in applications are various due to their deployed methodology to discover interesting clusters. We illustrate herein algorithms for each approach in detail. Figure 2.1 presents a taxonomy of clustering algorithms.

The taxonomy in [20, 25] for clustering algorithms is described as in Figure 2.1 except for the constraint-based approach since most clustering algorithms

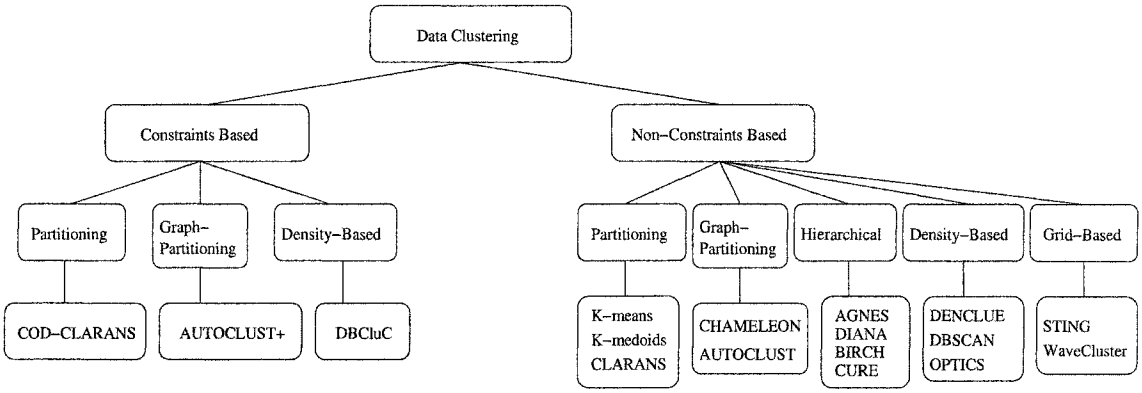


Figure 2.1: Taxonomy of clustering algorithms

have not taken into account constraints that may have a bearing on the effectiveness of clusters. Hence, we will also introduce clustering algorithms in the presence of constraints with a discussion of generalizations about constraints.

2.1 The Partitioning Approaches

In this section, we introduce clustering algorithms that group data objects by partitioning them in order to maximize intra-cluster similarity, whereas we maximize inter-cluster dissimilarity. All data objects in a planar space have their own memberships to a cluster such that noises and outliers are hard to exclude in clustering. Partitioning approaches require a parameter to start the partitioning of data objects; the number of clusters which require extensive prior knowledge.

2.1.1 K-means

The k-means is a well known clustering algorithm and was popular, before the data mining community started to become interested in it. The k-means studied in [33] partitions a set of data into k clusters such that members in a cluster are *"close,"* while members between clusters are *"not so close."* Each cluster is represented by a centre of gravity which is the "mean" value representing membership of the cluster. The method to evaluate clustering results on a planar space is a *squared-error function* E, where d is the data object in the database, m_i is the mean of cluster C_i, and k is the number of clusters.

$$E = \sum_{i=1}^{k} \sum_{\forall d \in database} |d - m_i|^2 \tag{2.1}$$

Initially, the algorithm randomly selects k points known as "centroids" from the given set of data objects. It then assigns the rest of each data point to a "nearest" centroid. After finishing the assignments, it locates new k centroids for each cluster calculated by the squared error function. Until there is no change in magnitude centroids, the forced assignment of memberships for

```
   Input  : A Database and k
   Output: A set of clusters
1  Randomly Select k data objects and set them as k centroids;
2  while there is any change in gravity centroids do
3  |  for all objects o in Database do
4  |  |  compute closeness between o_i and k centroids;
5  |  |  assign o_i to a closest centroids;
6  |  |
   |  endfor
7  |  compute mean values for each cluster and set them as new k cen-
   |  troids;
8  |
   endw
```

Algorithm 1: K-means Algorithm

newly composed k centroids is iterated. The k-means algorithm is illustrated as follows.

As described in Algorithm 1, the k-means randomly selects initial k centroids, while an inappropriate selection of centroids weakens the efficiency of the algorithm requiring unnecessary iterations. By iterating the Lines between Line 2 and Line 8, k-means algorithm attempts to minimize the square error E such that if there is no change in the centre of gravity, the algorithm halts.

A principal advantage of k-means is its running time, which is near $O(nkl)$, where n is the number of data objects and l is the number of iterations. Since there are n data objects to process, k clusters, l looping, and $k \ll n$ and $l \ll n$, the complexity could be re-articulated in the order of $O(n)$. On the other hand, one of the drawbacks of k-means is that it assumes that the number of data sets fits in the main memory, while this is not always true in real applications. k-means is very sensitive to noise and outliers, since noise and outliers substantially increase the value of the square error function for clustering. The k-means is not applicable to domains which represent a centroid with one data object for each cluster since centroids in k-means indicate the mean values of members in each cluster.

2.1.2 K-medoids: PAM and CLARA

k-medoids [28] is a clustering algorithm extended from k-means. The k-medoids represents clusters with *"k"* number of *medoids* that are the most centrally located data objects in each cluster, where k is a user's parameter. Due to the definition of *"medoid"*-a data object represent a cluster-k-medoids is less sensitive to outliers than k-means, decreasing the consequence of outliers and noise to the squared error function. The k-medoids initially selects k medoids from a given set of data objects. It then chooses a random-medoid from non-medoid data objects by examining whether the squared error Equation2.1 is reducible by replacing the non-medoid with one of the k medoids. It follows a forced-assign method which reassigns a data object to the closest medoid if a random-medoid decreases the square error function.

There are two typical k-medoids variants: *PAM* and *CLARA*. PAM [28] is a well-known k-medoids algorithm. It initially selects k medoids, then replaces one of the k medoids with a randomly selected non-medoid from the set of data objects if there is a *non-medoid* object that produces the lowest square error by swapping one of the k medoids with the *non-medoid.* If there is no such replacement after all the k clusters without changes in E have been looked through, then the algorithm halts with a local optimal. The difference between k-means and PAM lies in the method to assign data objects to their nearest centroids. However, the process of PAM is very inefficient because the comparison of all pairs of a centre and non-centres to examine whether cost is improved is very expensive. Its complexity is $O(k(n\text{-}k)^2)$, where n is the number of data objects. It is clear that PAM becomes expensive when n is increased.

CLARA (Clustering LARge Applications) [28] aims to overcome drawbacks of PAM such as memory management since PAM suffers from lack of scalability to large databases. CLARA employs a sampling approach rather than accessing whole data objects. Once it draws a sample from a database, CLARA applies the sample to PAM. CLARA performs multiple samples such that it

tries to avoid biased sampling. It then selects the most effective clustering by deploying PAM. Note that the quality of clustering is demarcated by the average dissimilarity of all members.

According to the experiments in [28], *5* samples of *40+2k* produce a reasonable quality of clustering. Yet CLARA is very sensitive to the following parameters: the number of samplings and the size of a sample, both of which may in turn induce skewed sampling. As seen in Algorithm 2, its complexity is $O(k(40+k)^2 + k(n-k))$. CLARA's contribution is efficient clustering which enables it to deal with large databases by employing the sampling approach.

```
   Input  : A Database and k
   Output: A set of clusters
   for 5 loops do
1  |  Randomly select a sample of 40+2k objects from data objects;
2  |  Apply PAM to discover k centers;
   |  Compute the average dissimilarity of the clustering from the previ-
   |  ous step;
   |  if the average dissimilarity is less than the current minimum then
   |  |  Set it to the current minimum;
   |  |  Keep the K medoids discovered in Line 2 for the best medoids
   |  |  so far;
   |  endif
   endfor
```

Algorithm 2: CLARA Algorithm

While k-medoids' variant algorithms PAM and CLARA are less sensitive to outliers than *k*-means, they suffer from a random selection problem of an initial centre of gravity and the halt criterion. K-medoids and K-means suffer from discovering arbitrary shaped clusters, while they well detect spherical shaped clusters. In general, K-medoids is useful in applications that need to represent each cluster with one of the data objects respectively. Figure 2.2 illustrates the difference between the *k-means* and the *k-medoids*, where an "x" notion represents each cluster in *k-means*, while a gray colored point represents a cluster in *k-medoids* .

```
Input  : A Database, k, numlocal, and maxneighbour
Output: A set of clusters
Get input parameters: numlocal and maxneighbour.;
Set min cost to a large number.;
while i ≤ numlocal do
1   | Set current to an arbitrary node in G_{n,k};
2   | Set j to 1;
3   | Take a random neighbour S of current;
    | Calculate the cost differential of the two nodes;
    | if S has a lower cost then
    |   | Set current to S;
    |   | Go to Line 2;
    | endif
    | j++;
    | if j is equal to or less than maxneighbour then
    |   | Go to Line 3;
    | endif
    | if j > maxneighbour then
    |   | // compare the cost of current with min cost;
    |   | if the cost of current < min cost then
    |   |   | Set min cost to the cost of current;
    |   |   | Set bestnode to current;
    |   | endif
    | endif
    | i++;
    | if i > numlocal then
    |   | Output bestnode and stop;
    | endif
endw
```

Algorithm 3: CLARANS Algorithm

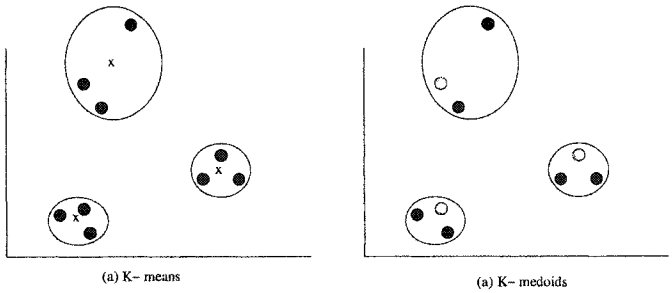


Figure 2.2: An overview of the K-means and K-medoids

2.1.3 CLARANS

CLARANS enhanced the k-medoids approach by overcoming disadvantages of PAM and CLARA such that efficiency and scalability are significantly improved. The performance is improved through building a graph and a sampling approach. The graph helps CLARANS project its steps to discover k-medoids by arbitrarily replacing a medoid with one from *(n-k)* data objects, where a node corresponds to a collection of k-medoids such that a clustering result is described by each node. Due to the notion of a graph model, CLARANS confines its search to a particular subgraph rather than examining all neighbours of a node. Notice that CLARA selects samples from all data objects in the beginning of the clustering process, whereas CLARANS does a sample of neighbours in each step. It is possible that CLARANS avoids skewed sampling, as it decreases the number of searches. However, the effectiveness of clustering is very sensitive to the parametric values *maxneighbour* and *numlocal*. "*Maxneighbour*" determines the maximum number of neighbours of a node to search. The purpose of "numlocal" is to choose the number of looping iterations as presented in Algorithm 3.

As seen in Algorithm 3, the performance of CLARANS relies on two input parameters. Obviously, it is not trivial to select impartial parameter values to ensure the best clustering quality. A high value of *maxneighbour* in CLARANS leads to clustering performance similar to that of PAM and CLARA, as the selection of *numlocal* also does. CLARANS may converge to a local optimum due to the nature of the partitioning approaches.

The main contribution of CLARANS is to enhance the efficiency of clustering procedures as proven in [35]. In addition, CLARANS outperforms PAM and CLARA with respect to effectiveness. Its complexity is reduced to O (n^2). CLARANS still inherits the shortcomings of CLARA and PAM: memory management to hold large databases, meticulous choice of parametric variables, arbitrarily initial selection of a node, assumption that clusters are spherical, etc.

To enhance the performance of CLARANS, [13, 12] discuss "Focusing Techniques" and "Focusing methods": Focus on Representative Objects, Focus on Relevant Clusters, and Focus on a Cluster. The focusing methods in [13, 12] are designed to improve the efficiency of CLARANS. Rather than accessing all non-medoid data objects, [13] processes the relevant data objects from a database adopting an indexing scheme R* tree [4].

2.2 The Hierarchical Approaches

Hierarchical clustering algorithms group data objects by constructing a *"dendogram"* based on the similarity between data objects. A dendogram refers to a tree structure to plot subgroups of data objects for each step in the course of clustering. A cluster is discovered in a *"dendogram"* by cutting at an explicit point.

Hierarchical clustering algorithms mainly follow variants: agglomerative and divisive. The agglomerative approach starts clustering data objects by merging similar clusters. In the initial step, each data object is itself a cluster. Each cluster is then grouped in relation to a pairwise similarity metric which has been employed by an agglomerative algorithm. Finally, it comes up with a cluster containing all data objects from a database, unless a halt criterion is addressed. On the other hand, the divisive approach starts from a cluster that encloses all data objects. It partitions a cluster into sub-clusters by evaluating the dissimilarity between intra-cluster members. A cluster in the divisive approach would be a data object itself as a cluster, unless a specific halt criteria is provided.

The single-link [43], the average-link [20], and the complete-link [30] measure similarity between clusters by computing the distance between them. The difference among them is a way how to measure the distance between clusters: a minimum distance, a distance between centroids, and a maximum distance. The single-link approach computes the minimum distance between all pairs of data objects from pairwise clusters, while the complete-link determines maximum distance. Note that the average-link measures the distance between centroids from each cluster. Two clusters are merged by the minimum distance to constitute a larger cluster.

2.2.1 AGNES and DIANA

AGNES (Agglomerative Nesting) [28] and DIANA (Divisive Analysis) [28] are well-known hierarchical clustering algorithms using bottom-up and top-down approach respectively. AGNES starts clustering each data object as a cluster. Each pair of cluster is merged to form a larger cluster. Its implementation deploys the Single-link method [43]. The dissimilarity matrix computes the conceptual correlations between clusters such that two clusters are merged if they are at least dissimilar. A cluster containing all members of a database would be composed, unless a specific halt condition is not provided.

DIANA groups data objects in a top-down approach such that one cluster containing all data objects is split into a subset of clusters. Consequently, each object forms a distinct cluster, unless a cutting point is cited. The partition occurs in a larger cluster to compose most dissimilar sub-clusters. It is unlikely that high dimensional data objects are applicable to DIANA due to the difficult and expensive problem of sub-dividing a cluster in each step. The following figure explains general steps in AGNES and DIANA.

2.2.2 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [52] aims to enhance efficient memory management in large databases and to lower I/O costs by compressing large databases. Note that these issues have been seriously taken into account by the data mining community. BIRCH collects

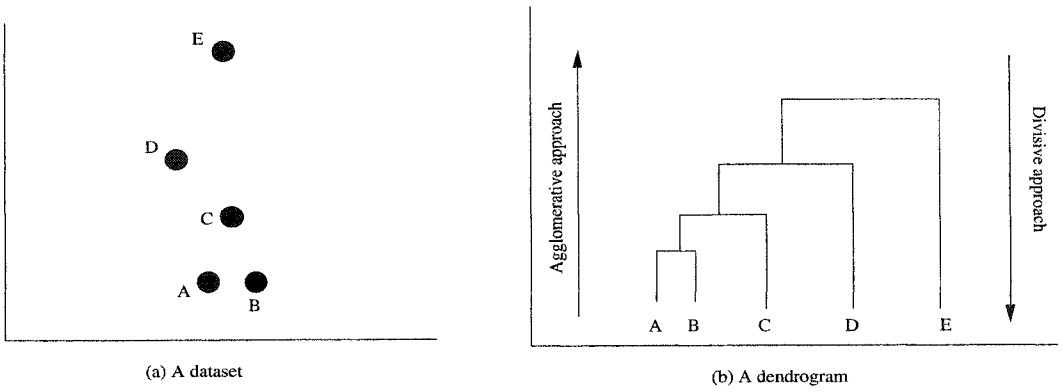


Figure 2.3: The overview of the AGNES and DIANA

the correlation of data objects by building a special data structure, a *CF-tree (Clustering Feature)*, in order to minimize the search spaces. The *CF (Clustering Feature)* is a triplet summarizing the information about a sub-cluster of data objects.

Given N d-dimensional data points in a cluster, the Clustering Feature (CF) entry of data objects in the cluster is represented as a triple *CF=(N, LS, SS)*, where *LS* is the linear sum of the N data objects, i.e. $\sum_{i=1}^{N} X_i$, and *SS* is the square sum of N data objects, i.e., $\sum_{i=1}^{N} X_i^2$. A CF-tree is a height-balanced tree requiring two parameters: a *branching factor* B and a *threshold* T. In addition, a CF-tree is dynamically constructed when the insertion operation occurs, if the insertion to a node is satisfied with B and T. The branching factor B determines the number of entries that each node holds. Each non-leaf node contains at most B of the form $[CF_i, child_i]$, where $child_i$ is a pointer to its i-th child node. A non-leaf node describes a sub-cluster composed of all the sub-clusters of its entries. In contrast, a leaf node contains at most L entries, and each entry is a *CF*. A threshold T in a *CF* indicates the diameter (radius) of each leaf entry, where the diameter of each leaf should be less than T. As the value T becomes larger, the tree becomes smaller. An example of a CF-tree structure is illustrated in Figure 2.4. Due to the utilization of the CF tree, the measurements in Table 2.1 are efficiently and incrementally computed to evaluate the closeness between clusters. Note that the last measurement indicates the quality of clustering.

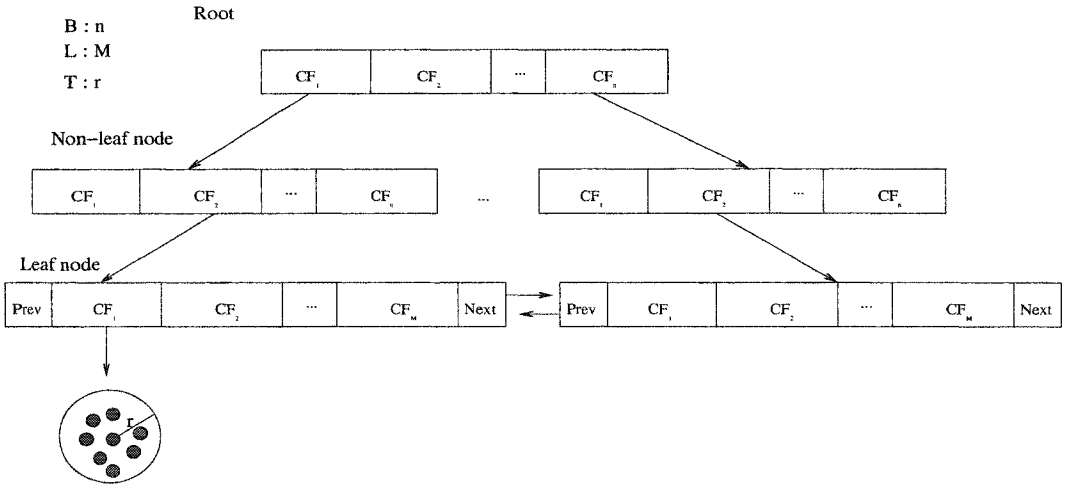


Figure 2.4: A CF tree

1. Centroids Euclidean(Manhattan) distance between two clusters.
2. Average inter-cluster(intra-cluster) distance.
3. Variance increase distance.
4. Weighted average cluster radius(diameter) square.

Table 2.1: Measurements of clustering

BIRCH is composed of 4 stages: Loading, Optional Condensing, Global Clustering, and Optional Refining. The first stage is to construct a CF-tree which fits in the main memory. The second step is the option to compact the CF-tree constructed in the first stage into a desirable range by building a smaller CF-tree. The smaller CF-tree where outliers are removed is reconstructed after scanning leaf entries in the initial CF-tree. In addition, a larger CF-tree is composed by grouping sub-clusters if the parameters are satisfied The third stage is to cluster all leaf nodes. Note that BIRCH makes use of other clustering algorithms in the third stage, while BIRCH adopted an agglomerative hierarchical algorithm as discussed in [52]. After clustering all leaf nodes, BIRCH tunes the clusters in the fourth step. The tuning is followed by using the centroids of the clusters produced by the third step as seeds and redistributing the data points to their closest seed to obtain a set of new clusters. This operation migrates data objects of a cluster as well as ensures that

all copies of a given data object are assigned to the same cluster. In addition, outliers might be discarded in this stage. Figure 2.5 illustrates the BIRCH algorithm.

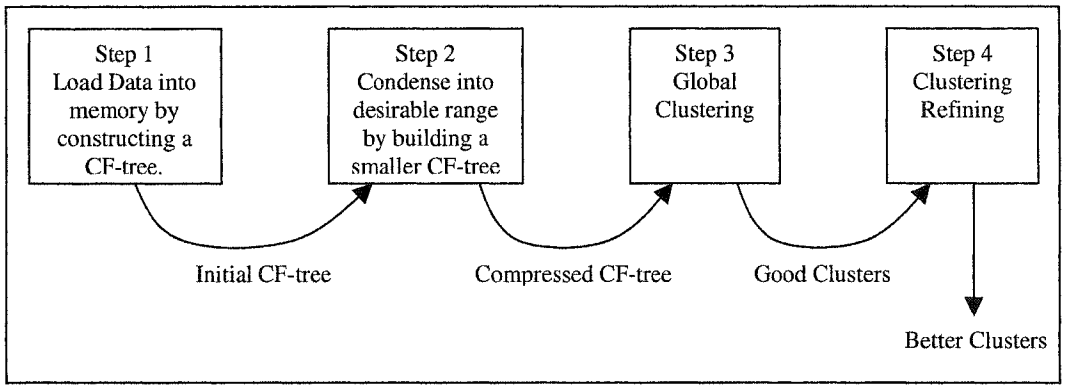


Figure 2.5: An Overview of BIRCH[52]

The I/O complexity of BIRCH is $O(n)$ where n is the number of data objects requiring only one scan of all data objects to construct a CF-tree. However, the complexity of BIRCH ignores the cost of the step 2, 3, and 4 in Figure 2.5. It is not trivial to select proper parameters: *branching factor* B and *threshold* T due to the sensitivity of clustering quality. Notice that the contribution of BIRCH is efficient memory management and reduced I/O costs by assuming that the CF-tree fits in main memory.

2.2.3 CURE

CURE integrates hierarchical and partitioning approaches [18]. The advantages of CURE are its ability to detect arbitrary shaped clusters, its decreased sensitivity to outliers, its linear storage requirements, and its efficient time complexity. CURE employs a novel hierarchical clustering approach that makes use of a middle ground between the centroids-based and the all-point extremes. A cluster is represented by a set of well scattered data objects. Based on a constant number c of well scattered data objects, a cluster shape and extent are captured, while it is shrunk to a centroid of the cluster by a *fraction* α. Due to the c representative approach, CURE is good at discovering a non-spherical shaped cluster. Note that a small value of c does not capture the

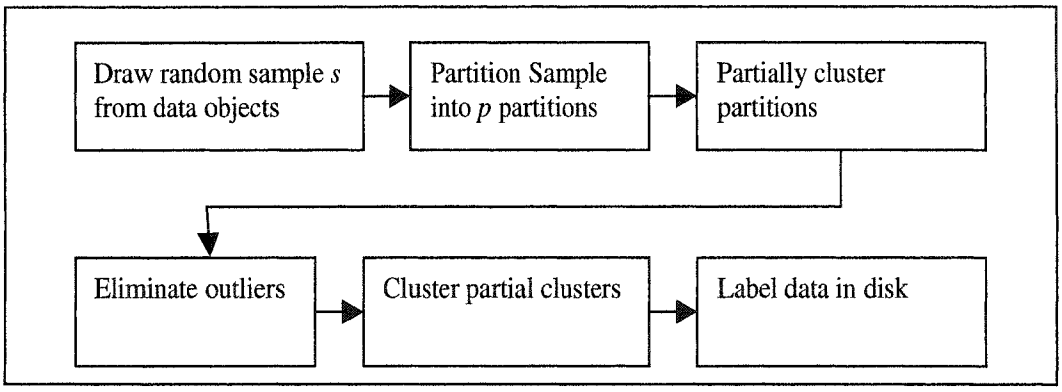


Figure 2.6: An Overview of CURE[18]

geometry of clusters properly. The deployment of a shrinking fraction α also alleviates the effects of outliers. As CURE pursues an agglomerative approach from the hierarchical method that starts with a data object as a distinct cluster, the clusters with the closest pair of representative data objects are merged at each stage. The similarity is calculated between c representatives from each cluster. An overview of CURE is presented in Figure 2.6.

CURE employs a random sampling and partitioning scheme that improves the quality of clustering through the filtering of outliers. As denoted in Figure 2.6, CURE partitions sample data objects and groups the partitions. Although CURE performs very well on a large database owing to its sampling method, it is sensitive to a possibly biased sampling and the parameters; α, c, s and p, where α is *fraction*,c is the number of well scattered data objects, s is the size of a sample, and p is the number of partitions for a sample space . For instance, if α=1, the clustering results are similar to those of BIRCH. The complexity of CURE is $O(s^2)$ for low dimensional data, where s is a number of random sample size from a database.

2.3 The Density Approaches

A controversial issue in partitioning and hierarchical approaches of clustering algorithms is how to provide the number of clusters and halt criteria respectively. Without knowledge about a database, it is not trivial to provide the number of clusters. In some cases, a natural cluster might be partitioned by a

user's parameter. Two non-similar clusters might be merged by a halt condition. These problems may be caused by ineffective *a priori* knowledge about the correlations between data objects. In this context, the partitioning and hierarchical methods are not very applicable to real applications. We will herein introduce algorithms that employ a notion *"Density."* It has been proven that a Density approach is very efficient at discovering natural correlations between data objects since a cluster has a high density distribution than noise or outliers do. In order to evaluate efficiently the correlations between data objects, a spatial index is useful to retrieve neighbours in a density-based approach.

2.3.1 DBSCAN

DBSCAN is a *density-based* clustering algorithm with two parameters, *Eps* and *MinPts*, both of which in turn determine clustering quality [15]. In order for data points to be grouped, there must be at least a minimum number of points *MinPts* in *Eps-neighbourhood*, $N_{Eps}(p)$, from a data point p, given a radius *Eps*. Its purpose is to detect natural clusters among data objects, as it discriminates noise and outliers from clusters. In DBSCAN, the following definitions are denoted.

Definition 1. (Directly density-reachable) A point p is directly density-reachable from a point q with respect to *Eps*, *MinPts* if

(1) $p \in N_{Eps}\ (q)$ and

(2) $|N_{Eps}\ (q)| \geq MinPts$, where $|N_{Eps}(q)|$ denotes the number of points in the circle of radius *Eps* and centre p.

Definition 2. (Density-reachable) A point p is density-reachable from a point q with respect to Eps and MinPts if there is a chain of points p_1, .., p_n, p_1 = q, , p_n = p such that p_{i+1} is directly density-reachable from p_i.

Definition 3. (Density-connected) A point p is density-connected to a point q with respect to *Eps* and *MinPts*, if there is a point o such that both p and q are density-reachable from o with respect to *Eps* and *MinPts*.

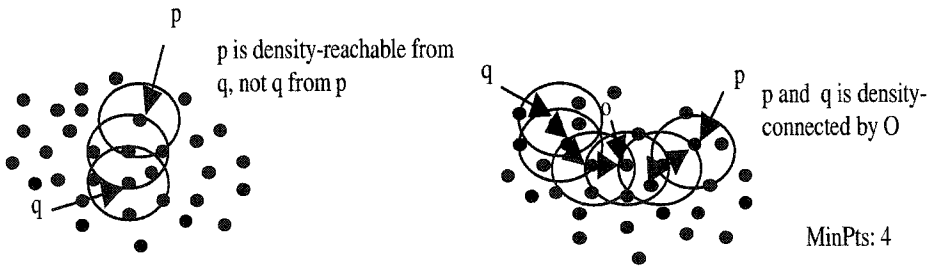


Figure 2.7: Density-reachable and Density-connected

Definition 4. (Cluster) Let D be a database of points. A cluster C with respect to *Eps* and *MinPts* is a non-empty subset of D satisfying the following conditions:

(1) Maximality: $\forall$ p, q if $p \in C$ and q is density-reachable from p with respect to *Eps* and *MinPts*, then $q \in C$.

(2) Connectivity: $\forall$ p, $q \in C$, p is density-connected to q with respect to *Eps* and *MinPts*.

Definition 5. (Noise) Let a data point $p \in D$. p is *noise*, if $p \notin C_i$, where C_i is i^{th} cluster, $0 \leq i \leq n$. And n is the number of clusters in D.

Figure 2.7 illustrates the above definitions. The detailed figures and discussion are found in [15].

Once two parameters *Eps* and *MinPts* are defined, DBSCAN starts to group data points from an arbitrary point. If a cluster cannot be expanded with respect to the notions *density reachable* and *density-connected*, it starts clustering data points for a new cluster. This procedure is iterated until there is no data point to be expanded and all data points in the dataset are clustered or labeled as a noise.

DBSCAN contributes to scalability, memory management for large databases, minimization of required domain knowledge, and effective clustering discovery while distinguishing clusters from noise or outliers, etc. One major issue with DBSCAN is sensitivity to the parameters *Eps* and *MinPts*. Even though

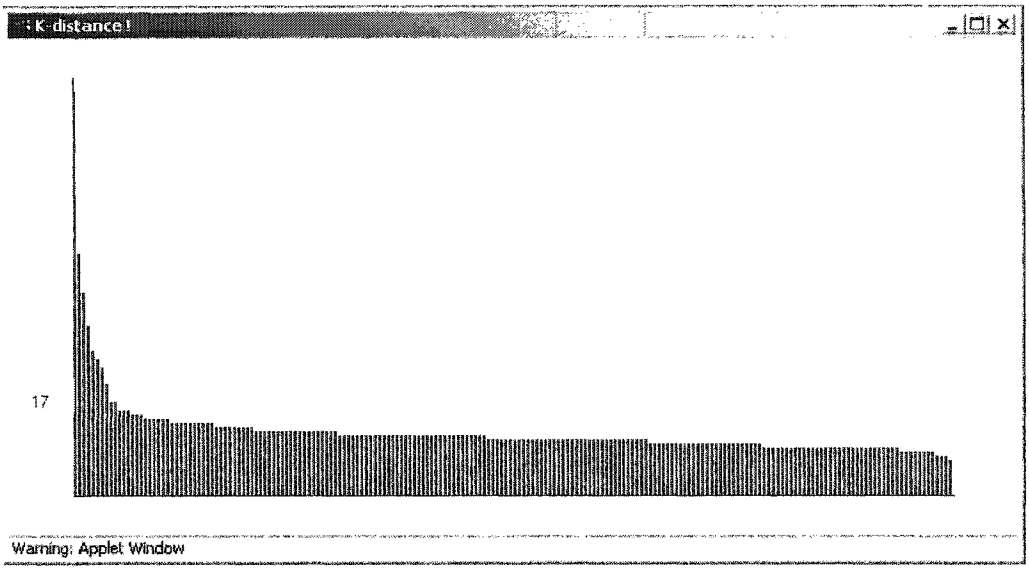


Figure 2.8: An example of a sorted $4-dist$ graph

DBSCAN requires minimized domain knowledge, it is not trivial to learn correlations between data objects. [50] illustrates the variant clustering results by changing the parameter values. In order to overcome the shortcoming an effective heuristic approach $k-dist$ graph is developed [15]. The observation of $k-distgraph$ is to define a $k-dist$ function from a data domain mapping every data object in the data domain to the distance from its k-nearest neighbour. If an arbitrary data object p is selected by setting Eps to $k-dist(p)$ and $MinPts$ to k, all points with an equal to or smaller than $k-dist$ value will be core points, where a point is a core point if there are at least $MinPts$ neighbour points within its Eps. Figure 2.8 illustrates the observation of the heuristic to determine the parametric values in DBSCAN. The other issue is the problem of the high dimensionality of data objects when looking for range queries to quickly identify points in the neighbourhood of another point. In [15], the authors indexed data objects using the R*tree [4], but this structure has a dimensionality limitation of 16 dimensions with respect to efficiency. As [9] reports, most of the running time $O(nlogn)$ in [15] is spent in searching neighbours with n data objects.

2.3.2 OPTICS

The idea behind OPTICS (Ordering Points to Identify the Clustering Structure) is extended from DBSCAN [15]. OPTICS presents an ordering of data objects rather than discovering a set of clusters given two parameters [2]. In DBSCAN, there are two users' parameters *Eps* and *MinPts*. As illustrated in [26], it is not trivial to find a "best" parameters for a large database without domain knowledge. Therefore, OPTICS allows users to choose the "best" parameters that represent correlations of data objects. OPTICS introduces the terms *core-distance* and *reachability-distance*.

(1) **Core-distance** (ϵ') of an data object p is the smallest value such that it makes p a core object that is defined in DBSCAN [15]. In other words, given a range of ϵ' from p, the number of neighbours of p is equal to or larger than *MinPts*. ϵ' is always smaller than ϵ, since ϵ' is not defined unless p is a core object.

(2) **Reachability distance** between two data objects p and q is the greater distance of the core-distance of p and the Euclidean distance between p and q. Therefore, the reachability distance is not defined if p is not a core object.

OPTICS creates an ordering of all data objects as it computes core-distance and a proper reachability distance for every data object. Note that OPTICS does not produce a set of clusters. Rather, it constructs a visualization plot such that a user is allowed to interact with the parameters to acquire explicable correlation information concerning data objects. This feature enables OPTICS to discover clusters with different densities and arbitrary shapes of clusters. OPTICS overcomes the shortcoming of DBSCAN, which is limited to discovering a globally dense area, and its performance is less sensitive to user inputs than DBSCAN. Figure 2.9 illustrated the result of OPTICS using two user inputs. However, the OPTICS still requires a parameter that may have influence on the clustering quality.

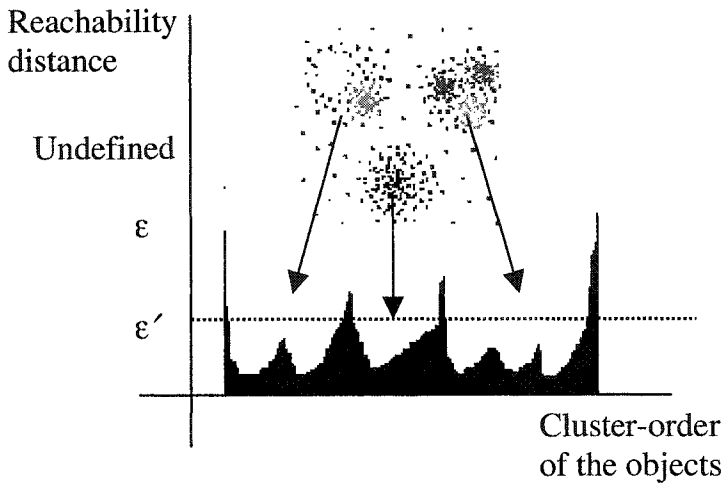


Figure 2.9: Orderings of data objects in OPTICS from [2]

OPTICS complexity is O($n$log$n$), which is equivalent to that of DBSCAN, where n is the number of data objects.

2.3.3 DENCLUE

DENCLUE (DENsity-based CLUstering) is designed to model the overall data object density, which is analytically determined by computing the sum of *influence functions* from data objects, where the *influence function* implies the impact of a data object on its neighbor [23]. A cluster in DENCLUE is determined by introducing *density-attractors*, which are local maxima of the overall density function and are determined by a hill-climbing procedure instructed by the gradient of the overall density function. The overall function is the sum of the influence functions for all data objects. Due to the fact that all data objects do not have an influence on the function, DENCLUE adopts a local density function to take into account data objects that contribute to the overall function. The *influence function* for $y \in F^d$ is a function $f_B^y : F^d \rightarrow R_0^+$, which is defined with respect to a basic influence function f_B as

$$f_B^y(x) = f_{B(x,y)} \tag{2.2}$$

The *density function* is defined as the sum of influence functions of all data objects. Given n data objects with a set of feature vectors D = $\{x_1, .., x_n\} \subset$

F^d, the *density function* is defined as

$$f_B^D(x) = \sum_{i=1}^{n} f_B^{x_i}(x) \tag{2.3}$$

Note that DENCLUE is able to arbitrarily select an influence function: parabolic, square wave, or Gaussian function. The following are examples of the influence functions in [23].

1. Square Wave Influence Function

$$f_{Square}(x, y) = \begin{cases} 0 & if\ d(x,y) > \sigma \\ 1 & otherwise \end{cases} \tag{2.4}$$

2. Gaussian Influence Function

$$f_{Gaussian}(x, y) = \sum_{i=1}^{n} e^{\frac{-d(x,x_i)^2}{2\sigma^2}} \tag{2.5}$$

In DENCLUE, two types of clusters are defined based on density-attractors: *the Center-Defined cluster* and *the Arbitrary-shape cluster.* For a density-attractor x*, a center-defined cluster which is density-attracted by x* and $f_B^D(x^*) \geq \epsilon$ is a subset of D, where ϵ is a threshold. Meanwhile, an arbitrary-shape cluster is a subset such that the density function for the set of density-attractors is no less than the threshold ϵ and there is a path P from x_1^* to x_2^* satisfying the requirement that the density function for all data points on the path is no less than ϵ. Outliers are determined if $f_B^D(x_0^*) < \epsilon$, where x_0^* is a local maximum.

Since DENCLUE requires user inputs σ and ϵ, extensive domain knowledge is required in advance. Although the clustering quality is sensitive depending on the inputs, it results in high-quality clusters in the presence of noise. Examples of clustering results are illustrated in [23]. Although DENCLUE is engaged in a grid-based approach, it discovers natural shaped clusters depending on a resolution of a grid. The complexity of DENCLUE is $O(n \log n)$, where n is the number of data objects. Even though the complexity of DENCLUE is equivalent to that of DBSCAN, DENCLUE is faster than DBSCAN by a factor of up to 45 thanks to its utilization of a grid-based approach.

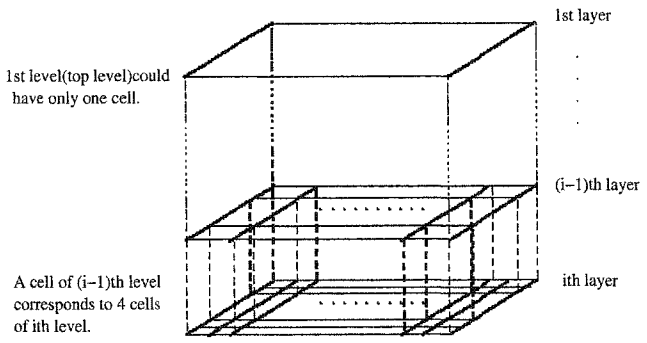


Figure 2.10: Hierarchical Structure of STING [48]

2.4 The Grid Approaches

In general, a Grid approach creates a finite number of cells for a given planar space. It attempts to induce correlations of data objects grid by grid. The clustering is done on the cells, not directly on data objects. Quantization of data objects enhances the efficiency of procedures. The efficiency is then determined by the resolution the grid approach quantizes.

2.4.1 STING

STING (STatistical INformation Grid-based method) deploys hierarchical and grid-based approaches to group data objects within a quantized cell [48]. Its conceptual idea is to compute query independent statistical information associated with each spatial grid(cell) in order to reduce the number of scans of data objects. The quantized grids are decomposed into a set of deeper layers. Each grid in a layer is associated with a different resolution. The resolution in turn decides the clustering efficiency. Statistical information for each grid is calculated and stored for further processing. Figure 2.10 illustrates the cell structure of STING.

As seen in Figure 2.10, a grid in level i corresponds to the union of the areas of its lower level $i+1$. The root grid, which is 1^{st} level, covers the whole spatial data plane. Each grid except for the leaves has 4 children, and each child is associated with one quadrant of the upper grid storing attribute-dependent and attribute-independent parameters such as n, m, s, min, max, and *distribution*,

where n is the number of objects in the grid,m is mean of all values in the cell, s is the standard deviation of all values of the attribute in a cell, *min(max)* is the minimum(maximum) value of the attribute in a cell, and distribution is the type of distribution that the attribute value in a cell. Note that distribution is a type of enumeration: normal, uniform, exponential, and so on. The parameters, except for distribution, are generated directly by the data objects whereas the distribution parameter is obtained by a user or by hypothesis tests such as the χ^2 test. Instead of examining all grids, STING queues the grids that are labeled as *"relevant."* The labeling of either *"relevant"* or *"not relevant"* is evaluated by the *confidence interval* (the estimated ranges), which implies how relevant a grid is to a given query. If a grid is labeled as *"not relevant,"* then it is further computed. The detailed calculation of the confidence interval (the estimated ranges) are discussed in [48].

The contributions of STING are the following: Statistical information allows STING to scan whole data objects only once; the computation speed of queries in n grid data structures is $O(n)$; and the independence of grid relations suits parallel processing. In spite of its multi-resolution approach, its clustering quality is not effective, since STING does not consider correlations between a child node and its neighbour nodes. As a result, clusters discovered by STING do not have diagonal boundaries, but are either horizontal or vertical.

2.4.2 WaveCluster

WaveCluster is a multi-resolution clustering approach such that multi-dimensional signals represent dense-regions in a planar space [42]. The 2-dimensional data in WaveCluster are converted into the frequency domain by employing *signal processing techniques* called *wavelet transforms.* The idea behind the techniques observes that 2-dimensional data objects can be represented in an n-dimensional *feature space* with *feature vectors* that are data objects in a feature space.

Wavelet transforms decomposes a signal into different frequency sub-bands utilized to discover the dense regions, which are considered as *clusters* in the

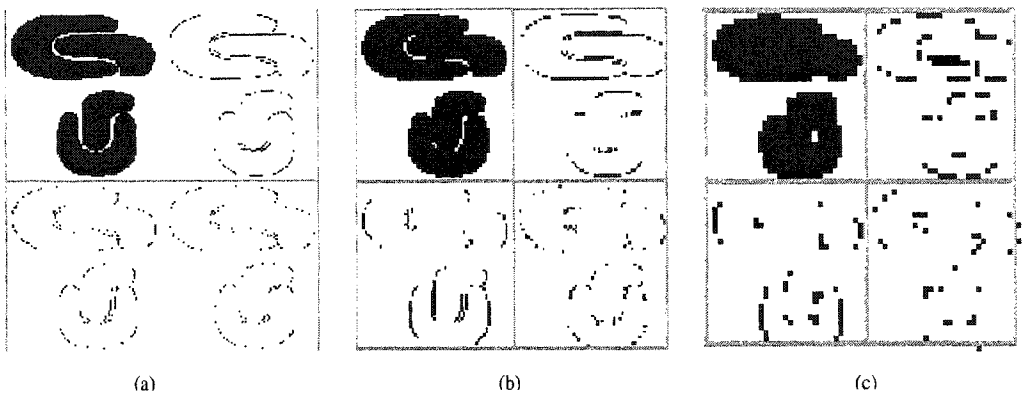


Figure 2.11: Multi-resolution wavelet representation of the feature space at (a)scale 1; (b) scale 2; (c) scale 3 from [42]

feature space. The clustering problem in WaveCluster is to find the distribution of patterns of feature vectors to represent the feature space with high frequency and low frequency areas of an n-dimensional signal.

Wavelet transform deployment has the following advantages: unsupervised clustering; effective removal of outliers; a multi-resolution approach; and cost efficiency. For instance, the *hat-shape* filters make regions distinct by hindering minor information in their boundaries. In other words, clusters in the feature space are automatically projected and represent regions excluding noises (outliers). The multi-resolution approach iteratively computes a coarser approximation of the one dimensional input signal using a low pass filter and down sampling that skips every other signal sample at different scales.

Figure 2.11 illustrates the iterative multi-resolution processes from a fine resolution (a) to a coarser resolution (b). For each resolution representation, 4 sub-bands are depicted: an *average signal(LL)* and *3 detailed signals (LH, HL, and HH)*, where *LL (wavelet approximation of original image)* is shown in the upper left quadrant, *LH (horizontal edges)* is shown in the upper right quadrant, *HL (vertical edges)* is presented in the lower left quadrant, *HL (vertical edges)* is presented in the lower right quadrant. By applying this multi-resolution process to feature vectors of data objects multiple times, a set of clusters is discovered. Notice that the performance of clustering is mainly influenced by the quantized feature space since the number of intervals that are quantized from the feature space are different at variant scales of transform.

Input : A Database, resolution degree, and signal threshold
Output: A set of clusters
Quantize feature space, then assign objects to the units;
Apply wavelet transform on the feature space;
Find the connected components(clusters) in the sub-bands of transformed feature space, at different levels;
Assign label to the units;
Make the lookup table;
Map the objects to the clusters;

Algorithm 4: Algorithm of WaveCluster [42]

In Algorithm 4, the first step is to quantize the feature space into a set of intervals that have influence on the performance of clustering. Discrete wavelet transform is then applied to the quantized space. The third step maps data objects in the transformed feature space to data objects in the original feature space by labeling data objects and creating a *Look up* table.

While WaveCluster satisfies the primary requirements of a good clustering algorithm- the detection of the arbitrary shapes of clusters; automatic clustering processes; and efficient time management with large databases– its employment of convolved and down sampled images with a low pass filter interprets the bridge-like set of data objects as a strong signal. In addition, the parameters such as degree of resolution and signal threshold have significant effects on clustering performance.

2.5 The Graph-Partitioning Approaches

The Graph-partitioning approaches employ a graph to model data objects and to represent the "closeness" of neighbours as an edge. The "close" neighbours have connectivity determined by weighted edges that are dynamically collected in the course of clustering. As a result, the correlations of data objects are automatically evaluated. However, due to the deployment of a graph that requires high processing costs, the clustering procedure is usually inefficient.

2.5.1 CHAMELEON

CHAMELEON [26] is a hierarchical clustering algorithm adopting dynamic modeling to merge sub-clusters. Whether two clusters are merged is determined by the similarity between the inter-connectivity and proximity of the merged cluster and the inter-connectivity and proximity of the two clusters before merging. CHAMELEON models data objects as a k-nearest neighbour graph $G_k(V, E)$, where V is a set of vertices and E is a set of edges. Each vertex in G is a data object in a database, and each edge is constructed between two vertices v_i and v_j if one of the k-nearest neighbours of v_i is v_j. Since the weight on an edge represents the closeness between two vertices, the edge weight of dense regions in G_k is large, whereas the edge weight of sparse regions is small. Using the k-nearest neighbour graph, operations to evaluate correlation from data objects are efficiently performed. For instance, extremely distant data objects in the k-nearest neighbour graph are naturally disconnected, whereas it captures correlations between neighborhoods dynamically and the density region is recorded as the weights of its edges.

The similarity between two clusters C_i and C_j is computed by two criteria- *Relative Inter-Connectivity(RI)* and *Relative Closeness (RC)*. *RI* is defined by the absolute inter-connectivity between C_i and C_j normalized with respect to the internal inter-connectivity of the two clusters, where the absolute inter-connectivity is the weighted sum of the cut between C_i and C_j. It is formalized as follows. Note that $EC(C_i, C_j)$ is the weighted sum of the cut between C_i and C_j. $EC(C_i)$ is the weighted sum of edges being cut in the graph that describes a cluster C_i to generate two bisector. *RI* allows CHAMELEON to consider the differences in shapes of the clusters and the degree of connectivity of different clusters.

$$RI(C_i, C_j) = \frac{EC(C_i, C_j)}{0.5(EC(C_i) + EC(C_j))} \tag{2.6}$$

RC is defined as the absolute closeness between C_i and C_j normalized with respect to the internal closeness of two clusters. The closeness of two clusters is measured by computing the average similarity between data objects in C_i that are connected to C_j in order to alleviate the effects of outliers and noises.

RC is computed as,

$$RC(C_i, C_j) = \frac{\overline{S}_{EC(C_i,C_j)}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC(C_i)} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC(C_j)}} \tag{2.7}$$

Note that $\overline{S}_{EC(C_i,C_j)}$ is the average weight of edges that connect vertices in C_i to vertices in C_j prior to the partition and $\overline{S}_{EC(C_i)}$ is the average weight of the minimum number of edges that need to be cut to create bisectors with C_i. *RC* helps CHAMELEON merge clusters whose merged cluster shows a consistency in the degree of closeness between data objects in the cluster if both the relative inter-connectivity and the relative closeness are high. This is formalized to maximize the function as follows:

$$RI(C_i, C_j) \cdot RC(C_i, C_j)^{\alpha} \tag{2.8}$$

The value α is a user parameter to distribute the magnitude ratio between *RI* and *RC*. Due to considerations of inter-connectivity and relative closeness, CHAMELEON provides better performance than other hierarchical methods such as CURE. Notice that CHAMELEON deploys a dynamic model such that clustering procedures are being automatically adjusted, whereas other hierarchical clustering method cannot interfere with clustering operations, once started. [26] has compared the experimental performance of CHAMELEON with CURE and DBSCAN, and found that CHAMELEON outperforms CURE with respect to detecting the natural shapes of clusters. However, its time complexity is not efficient. Its worst time complexity is $O(n^2)$, where n is the number of data objects. Besides, effectiveness is dependent on the requested parameters.

2.5.2 AUTOCLUST

AUTOCLUST [16] is a graph partitioning algorithm that does not require any user parameter such as the number of clusters, halt criteria, and density arguments. The parametric resources for a database are automatically collected by Delaunay diagram without *a priori* knowledge. In the Delaunay diagram, data objects residing in the boundary of a cluster imply a greater standard

deviation in the length of their incident edges since the boundary objects have both short edges and long edges. Short edges indicate the closeness of the data objects that are to be clustered together, whereas long edges correspond to a relation between clusters or between a cluster and noise.

AUTOCLUST involves three steps: finding boundaries, restoring and re-attaching, and detecting second-order inconsistency. Each phase performs an edge correction to adjust proximity between data objects. The connectivity between data objects in each step is determined by the following features: *Local_Mean(p)* and *Mean_St_Dev(P)*, where P is a set of data object and $p \in P$; *Local_Mean(p)* is the average length of edges incident to a data object p; and *Mean_St_Dev(P)* is the average of the standard deviations in the length of incident edges for all points in P. Initially, AUTOCLUST constructs granular clusters from the Delaunay diagram. It then removes edges from the Delaunay diagram. The elimination of edges as the first step is done by the proximity (closeness) derived from *Long_Edges(p)* and *Short_Edges(p)*. The *Long_Edges(p)* indicates that the edges are two long joining data objects from different clusters, while *Short_Edges(p)* connects two close data objects within a cluster. The second step is to recover edges in *Short_Edges(p)* that are intra-cluster links. The third step recalculates the local variation to remove inappropriate edges evaluated by the new indicator $Local\_Mean_{2,G}(p)$ which is the average length of edges whose paths are 2 neighbourhoods starting at p. Figure 2.12 presents the Delaunay structure and phases of AUTOCLUST.

The contributions of AUTOCLUST facilitate automatic clustering in the absence of *a priori* knowledge and detection of locally dense areas. However, it suffers from dimensionality problems as a result of its data modeling method. If data objects whose dimension is higher than 2-dimension are applied to AUTOCLUST, its performance is much degraded. For example, d-dimensional data objects require $O(n \log n + n^{\frac{d}{2}})$, which is almost a polynomial increment.

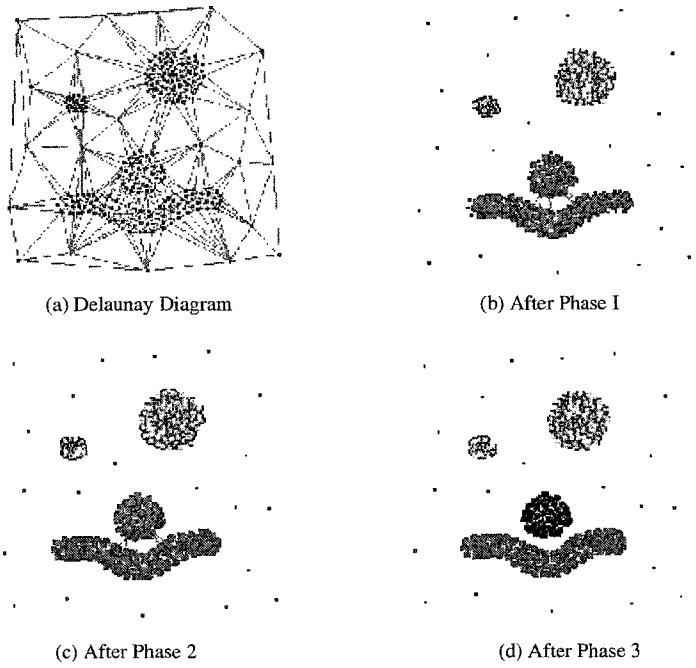


Figure 2.12: AUTOCLUST Illustration [16]

2.6 An Educational Applet of clustering algorithms

For experimental and educational purposes, a Java applet has been implemented to illustrate the performance of clustering algorithms in a two dimensional space. The algorithms coded in the applet are *K-means, K-medoids, DBSCAN,* and *CLIQUE.* The first two algorithms are typical partitioning algorithms, the second one is based on a density notion, and the last is based on a grid notion. A number of synthetic data sets are available in the applet such that it is easy for users to compare clustering quality by testing different algorithms on a data set. In the first row in the applet(see Figure 2.13), some parameters can be selected. The first choice option allows users to select the number of clusters for *K-means* and *K-medoids* algorithms. Hence, users can realize how hard it is to select k values, that is, the number of clusters without *a priori* domain knowledge. The second choice option is the list of datasets. The applet also accepts a dataset provided by users. A user simply clicks the panel to generate his own dataset. The third option controls the execution

interval of an algorithm. It is possible for users to see the process of an algorithm in detail. A user can select an algorithm from the last option choice. The three buttons-"Copy frame", "Restart", and "Show"- on the right side of the applet allow a user to store a clustering result ("Copy Frame"), to restart the applet after changing options ("Restart") and to show a clustering result after finishing the procedures of an algorithm("Show"). The text field at the bottom of the applet is designed to show the pseudo code of each algorithm. As well, each line of a pseudo code is highlighted as the line is executed by an algorithm. The following snapshots show the structure of the applet and examples of clustering results. Note that each cluster centre is represented by a red point in the *K-means* and *K-medoids* algorithms. In addition, noise is represented by a red point in *DBSCAN* and *CLIQUE*[1] algorithms.

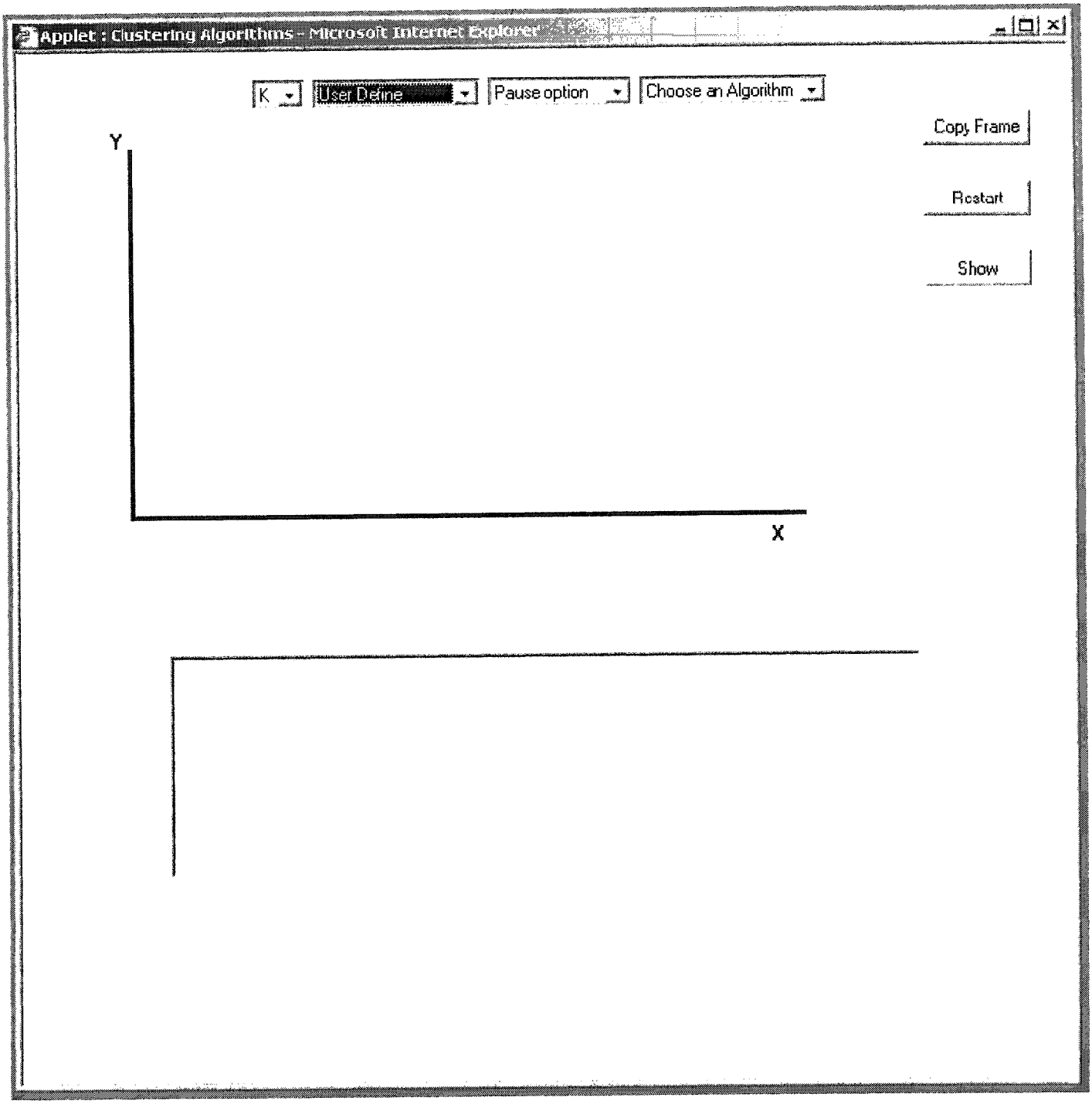


Figure 2.13: A snapshot of the applet

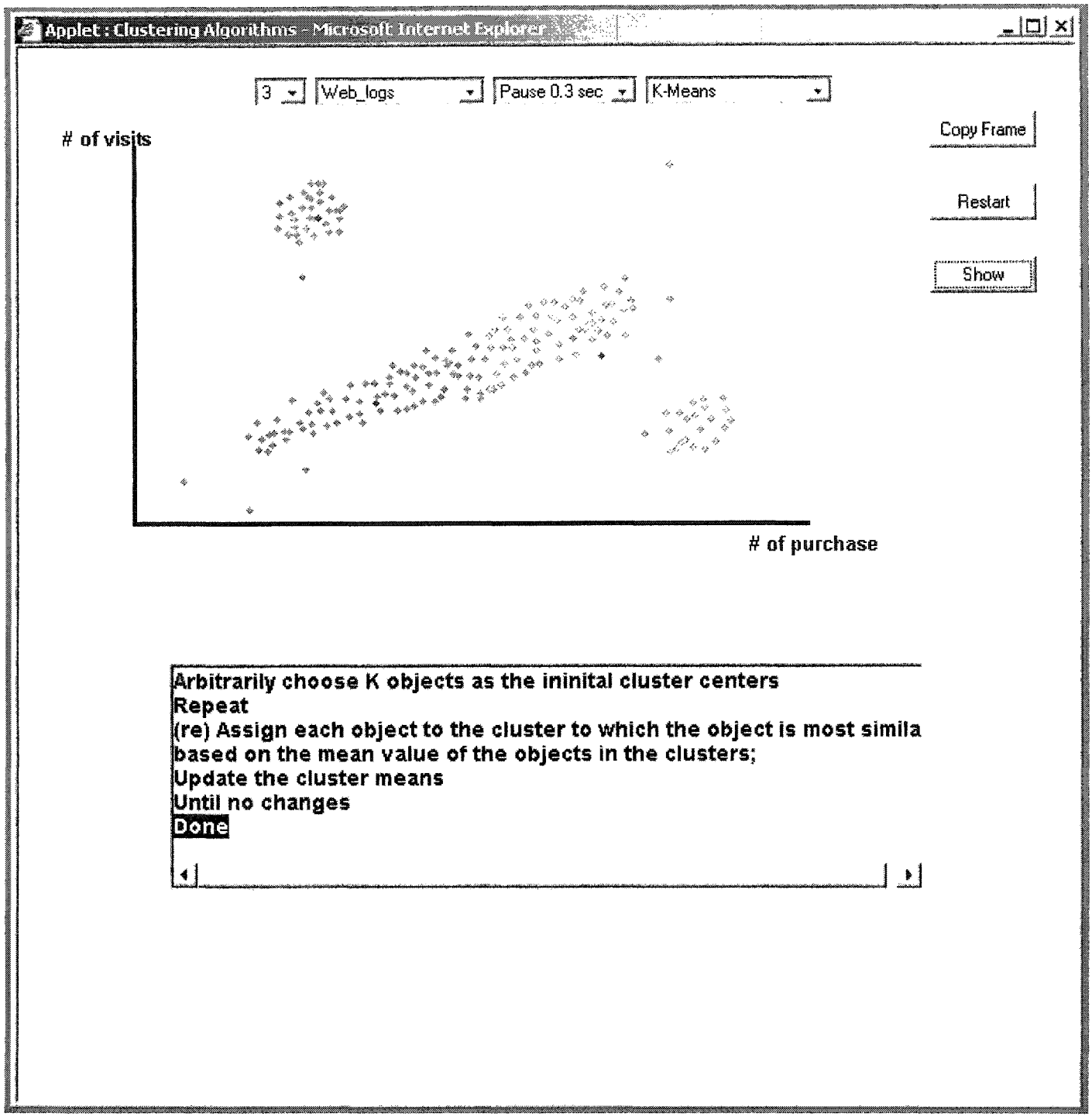


Figure 2.14: Clustering results in K-means

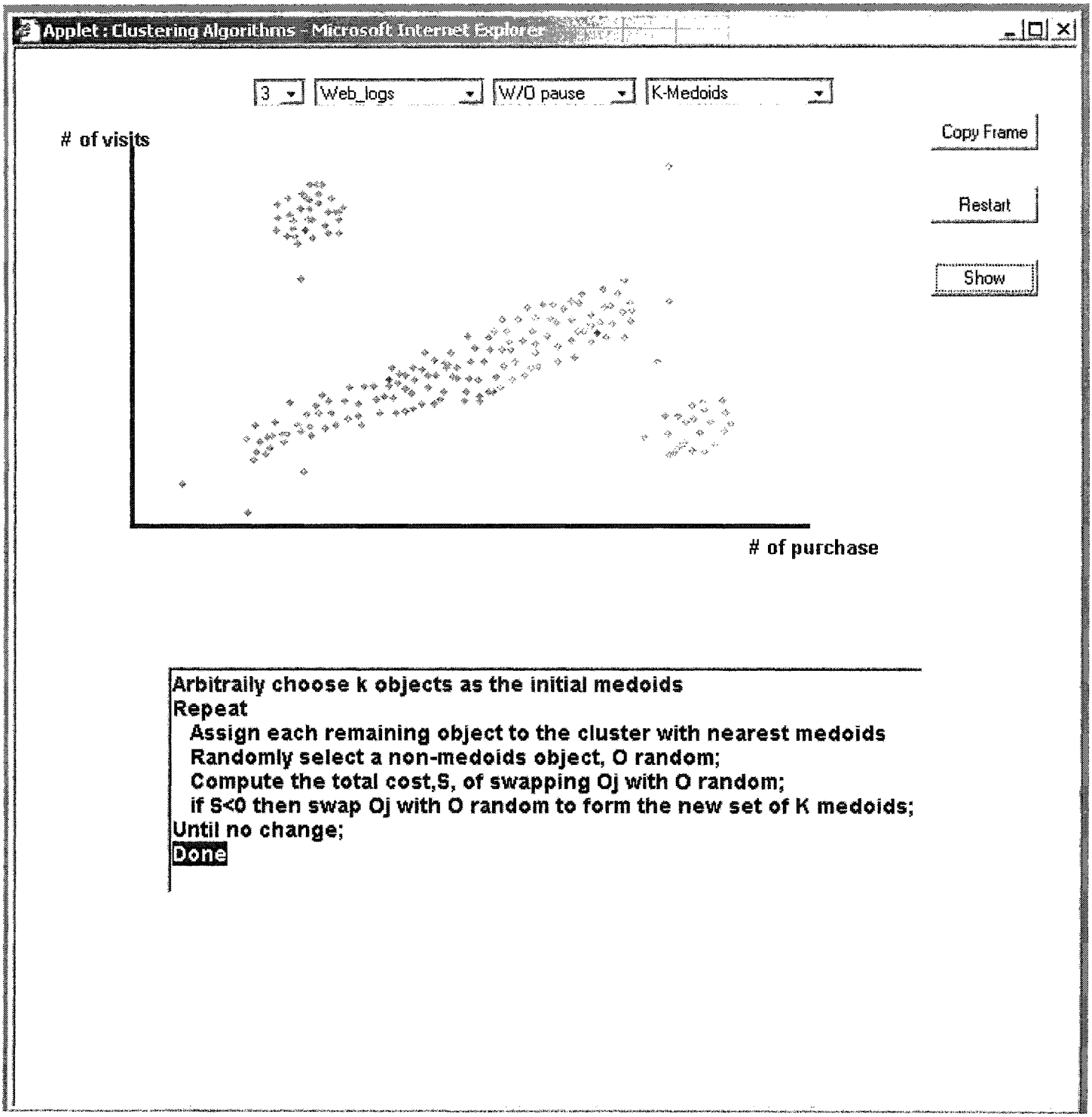


Figure 2.15: Clustering results in K-medoids

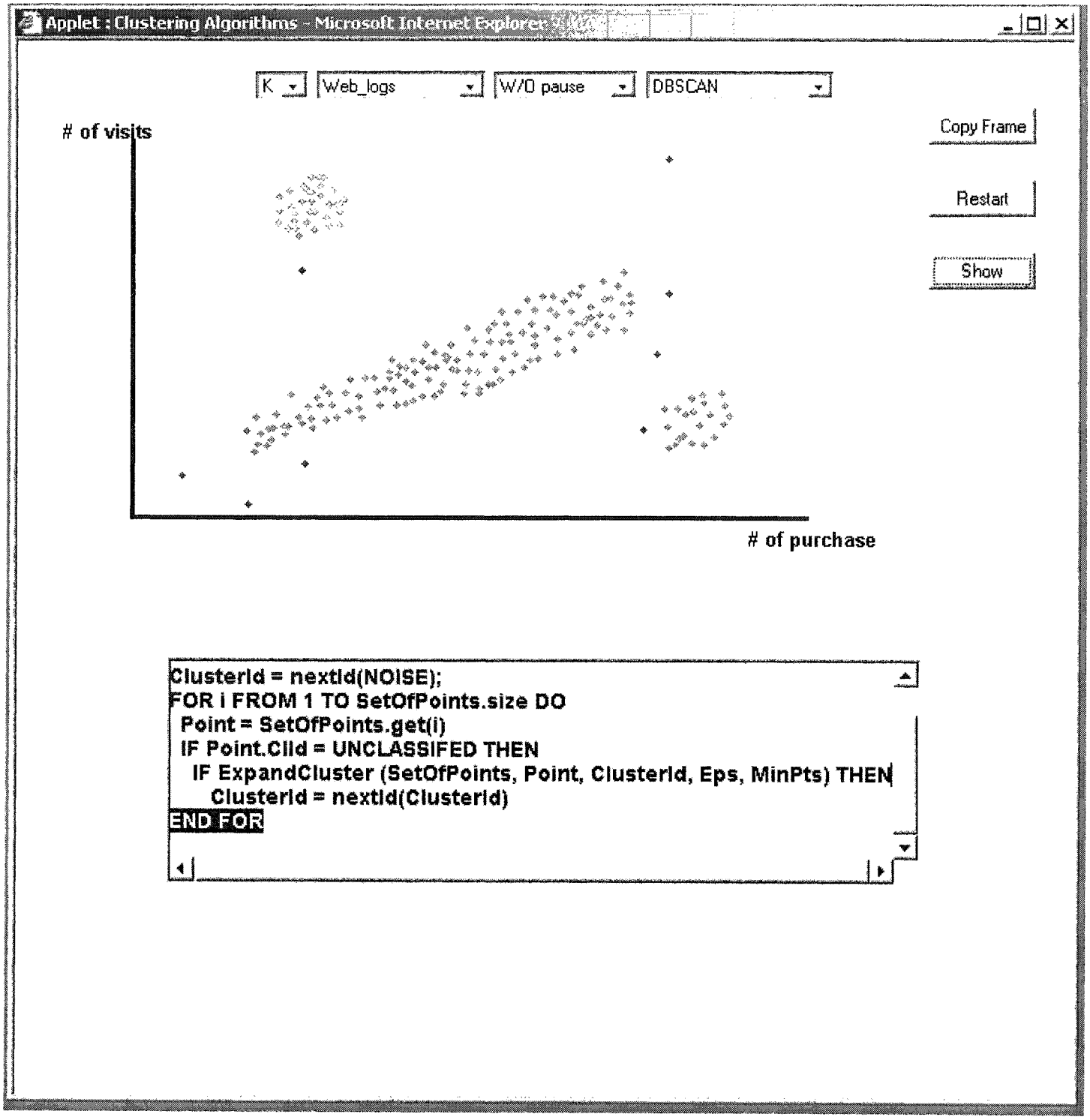


Figure 2.16: Clustering results in DBSCAN

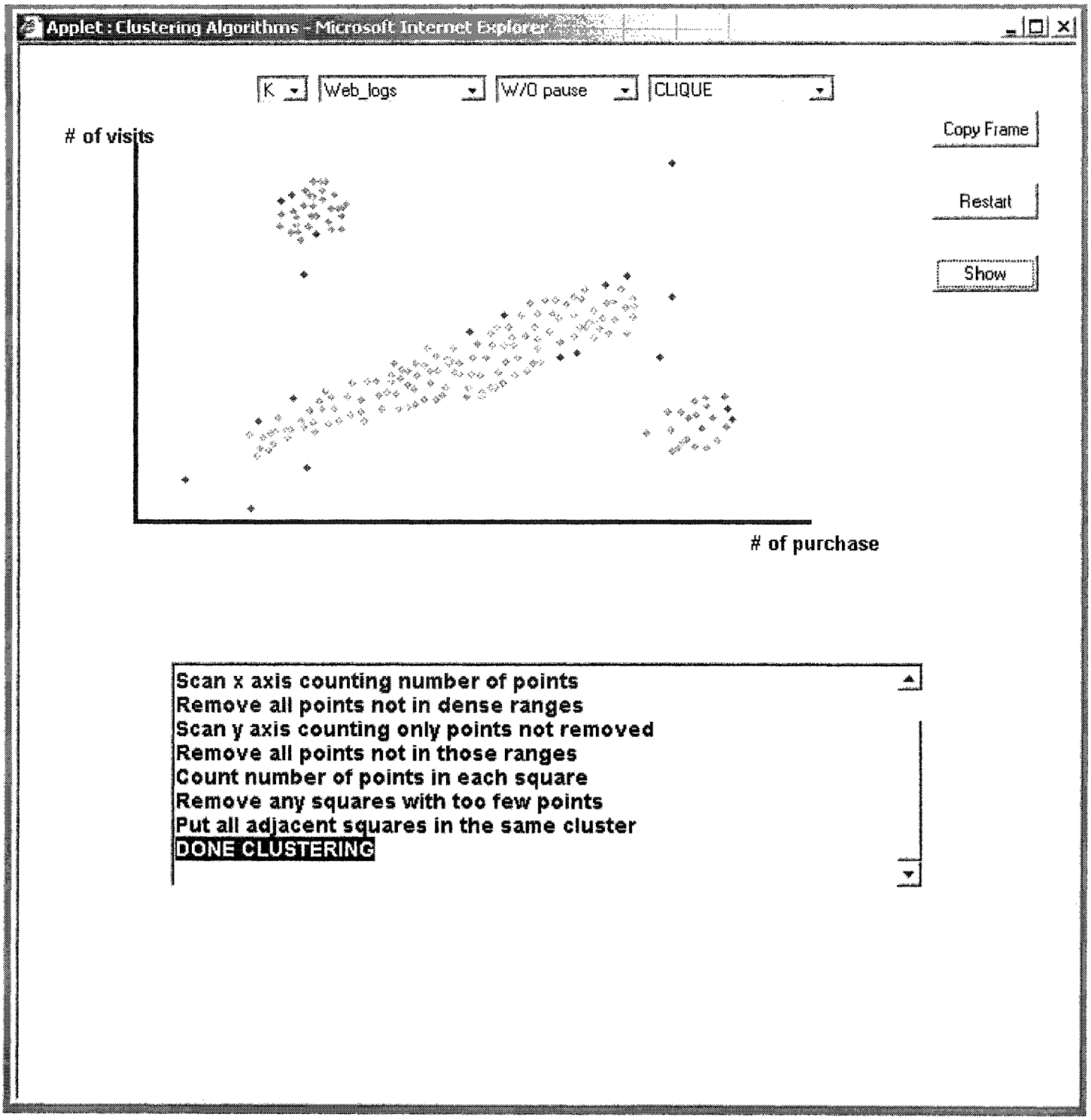


Figure 2.17: Clustering results in CLIQUE

Chapter 3

Data Clustering with Constraints

We have shown clustering algorithms that have focused on efficiency and effectiveness in clustering data objects in a planar space. Those algorithms principally describe the performance enhancement required to achieve better quality in a linear time, in order to handle large spatial multi-dimensional databases. However, none of the algorithms that have been introduced so far takes into account possible physical or operational constraints that may significantly influence the accuracy of the clustering of data objects, as well as the efficiency of clustering procedures, since such consideration in the course of clustering is very expensive. For example, when a company groups customers based on the distance to its branches, it may want to have a set of clusters such that each cluster should have a certain number of customers, in the presence/absence of physical constraints. In a GIS application studying the movement of pedestrians to identify optimal bank machine placements, for example, the presence of a highway hinders the movement of pedestrians and should be considered as an obstacle, whereas a pedway over this highway could be considered as a bridge. Hence, ignoring the constraints will significantly change the accuracy of clustering. It is imperative that we take into account constraints on clustering problems. In addition, [21] introduces a framework that integrates constraint-based and multidimensional mining scheme for effective and efficient data mining.

In this section, we shall show three clustering algorithms [46, 45, 17] that

focus on operational and physical constraints.

3.1 AUTOCLUST+

AUTOCLUST+ [17] has been proposed as an extension of the clustering algorithm, AUTOCLUST [16]. Due to the natural property of AUTOCLUST such as a special data structure to model data objects "Delaunay diagram", AUTOCLUST+ efficiently discovers clusters in the presence of obstacles. Notice that AUTOCLUST constructs clusters by means of a self-tuning procedure, that is, the essential parameters to evaluate correlations of data objects are generated automatically.

In AUTOCLUST+, an obstacle has been modeled by a set of simple line segments such that obstacles obstruct the edges from the Delaunay diagram, where each incident edge in the Delaunay diagram represents the closeness of data objects. The disconnectivity for each obstacle is applied to the Delaunay diagram by removing edges that are impeded by a line segment from an obstacle. A removed edge is replaced in the Delaunay diagram with a detour path that is defined as the shortest path between two data objects. The following summarizes the procedures of AUTOCLUST+:

```
Input  : A Database
Output: A set of clusters
Construct Delaunay Diagram;
Compute Mean_ST_Dev(P) ;
for all edges do
    if an edge intersects some obstacles then
        replace it with a detour path;
    endif
endfor
Apply AUTOCLUST;
```

Algorithm 5: Algorithm of AUTOCLUST+

The complexity of AUTOCLUST+ is O(n log n + $[m+R]$ log n), where n is the number of data objects, m is the number of line segments from all obstacles, and R is the number of edges removed from Delaunay diagram.

Thanks to the utilization of the Delaunay diagram structure, it is possible to efficiently validate if a line segment from an obstacle is intersected with edges from the Delaunay diagram. However, reconstructing the diagram after the removal of edges is not explicitly taken into account in the complexity computation. The Delaunay diagram searches a detour path for every line segment intersected with the diagram. This eventually degrades the performance of the algorithm. In addition, the effect of noise that is depicted by a set of close data objects limits the effectiveness of AUTOCLUST+ in real applications. The direct evaluation of correlations between data objects from a database will avoid biased clustering results.

3.2 COD-CLARANS

COD-CLARANS (Clustering with Obstructed Distance) [45] was introduced by *A. K. H. Tung et al.* in 2001. The goal of COD-CLARANS is to discover clusters from large databases by minimizing the sum of distance-error, such that data objects stay in their "closest" centres in the presence of obstacles. Note that the "closest" denotes the closest distance that detour obstacles.

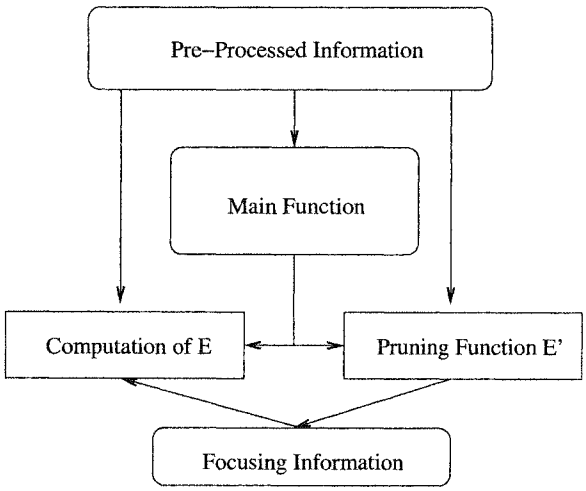


Figure 3.1: Overview of COD-CLARANS [45]

COD-CLARANS has been derived from CLARANS [35], which is a variant of k-medoids approach to ameliorate deficiency of PAM [28] and CLARA [28]. CLARANS constructs a graph to project steps to discover k-medoids (cen-

troids). In the graph, a node corresponds to a collection of k-medoids such that a clustering result is described by each node. Therefore, a cost for each node is computed by the dissimilarity between every object and the medoid of its cluster. The cost is re-calculated whenever a centroid changes. This ultimately degrades the performance of the algorithm. COD-CLARANS examines the obstacles by integrating an optimization scheme into the distance-error function in the course of the algorithm.

COD-CLARANS is composed of four phases: pre-processing, the main algorithm, computation of the distance-error E, and a pruning function E'. Each step is illustrated in Figure 3.1.

Obstacles in COD-CLARANS are modeled with *a visibility graph, VG = (V, E),* such that each vertex of the obstacles has a corresponding node in V, and two nodes (v_1 and v_2 in V) are connected by an edge in E, if and only if the corresponding vertices they represent are visible to each other. The visibility graph is pre-processed to compute the obstructed distance between two data objects in a given planar space. The obstructed distance lies in a detoured distance between two data objects via the visibility graph. Once the visibility graph is constructed, COD-CLARANS applies *Micro-clustering* approach to all data objects. Micro-clustering represents a cluster with a set of data objects from an intra-cluster. The advantage of the micro-clustering approach is to minimize the number of data objects to be processed and to fit into the main memory, which is not available in CLARA. To overcome a drawback of the micro-clustering method, such as *information loss of data objects*, a user's parameter, *radius of a micro-cluster*, is required. Note that an imprudent selection of the parameter results in a cluster that contains only one data object.

The visibility graph construction and micro-clustering are considered a pre-processing phase in COD-CLARANS. After the pre-processing, COD-CLARANS randomly selects the centres of k clusters from the micro-clusters. It then iterates for a randomly selected centre out of non-centre data objects, to replace a current centres. If the k clusters remain unchanged after iterating *MaxNeighbour* times, then COD-CLARANS maintains the sum of

distance-error value and cluster assignment. The procedure is repeated up to *NumOfLocal* times, as the minimized square-error is assessed. Note that *MaxNeighbour* and *NumOfLocal* are the input parameters. The exploratory time and the clustering results are extremely sensitive to those parameters.

The contribution of COD-CLARANS is to consider obstacles modeled by a visibility graph for efficient processing of the clustering problem. In addition, an optimization technique is adopted in the course of deploying COD-CLARANS: a pruning function E'. E' is computed by the distance between O_{random} and the micro-clusters, where the distance is a direct Euclidean distance, not an obstructed distance, and O_{random} is a randomly selected centre, to be replaced with one of the current centres. The pruning function E' prunes searching costs by inspecting the lower bound of the distance-error E. Furthermore, the pruning function addresses *focusing information* for efficient computation of E. On the other hand, COD-CLARANS inherits the shortcomings of CLARANS such as sensitivity to noise and to parameter choice, significant a prior knowledge of data objects and obstacles, the loss of clustering quality, etc.

3.3 Constraint-based clustering in large databases

While physical constraints such as obstacles or crossings are important in this performance of clustering algorithms, the operational requirements are also critical to clustering effectiveness. For this purpose, a scalable constraint-clustering algorithm [46] that is a variant of k-means is introduced, presenting the taxonomy of constraints in clustering issues according to the nature of constraints and application domains, as follows:

1. `Constraints on individual objects`
2. `Obstacle objects as constraints.`
3. `Clustering parameters as constraints.`
4. `Constraints imposed on each individual cluster.`

The goal of [46] is to satisfy *existential constraints*, that is, to discover a set of k clusters, such that each cluster has at least c *pivot objects* minimizing a

given distance-error function to k centroids, where pivot objects are specified via constraints or other predicates, through input parameters. Therefore, the NRP (Nearest Representative Property) is not compromised in the constraint-clustering context. Even though objects are close to a centroid, they cannot be assigned to the centroids due to the existential constraints.

[46] develops an algorithm called *cluster refinement* under an existential constraint.The cluster refinement algorithm in the constraint space is modeled by a *clustering locality graph.* A node in clustering locality graph represents a set of k clusters. An edge between two nodes of the graph lies in if and only if they are different by only one pivot object. Note that there are two classes of labeled nodes: *valid* and *invalid*, where the former satisfies an existential constraint and the latter does not. Since the decision problem corresponding to the k-means algorithm is NP-complete, the graph is designed to discover a local optimum by a node movement that gives the greatest decrease in distance-error function. This movement, called *cluster refinement process*, is iterated until no node of lower distance-error is found.

In brief, the algorithm consists of two phases: *pivot movement* and *deadlock resolution.* The algorithm constructs a *Pivot Movement Graph* to search the best-optimized pivot movement that minimizes distance-error function, where a *Pivot Movement Graph* is a directed graph in which each cluster is represented by a node. Given a *Pivot Movement Graph*, a pivot movement is converted to a problem by computing a schedule of movements for the pivot objects in the graph, such that the distance-error function is minimized. The purpose of deadlock resolution phase is to solve a *deadlock-cycle*, when a given existential constraint is too tight to be satisfied. Some heuristics, such as *random-heuristic* and *a look-ahead*, are employed because two steps are NP-complete.

As demonstrated in both phases, it is very expensive to perform this algorithm with large databases. To overcome the high cost, the algorithm has adopted a *micro-clustering* approach [45]. The micro-clustering method, however, may not be properly efficient due to the movement of micro-clusters, rather than a pivot object in order to satisfy an existential constraint.

Discussion of *an existential constraint* examines the possible extension of constraints: existential-like, universal, averaging, and summation. Universal constraints are constraints in which a specific condition must be satisfied by all objects in a cluster. Existential-like constraints are very similar to existential constraints, as the existential-like constraints moves "hole" to minimize the distance-error function to clusters. The averaging and summation constraints are constraints that require average and sum arithmetics for numeric attributes. This computation is a NP-hard problem.

Most recently a survey [50] on the Data clustering has been published. It discusses the latest clustering algorithms comparing their clustering qualities. In addition, [50] generalizes the validation of the clustering quality. One major contribution of [50] is to compare the performance of each clustering algorithm based on the same data sets. Hence it is very clear for readers to evaluate the performance, especially the sensitivity of parametric values that are required by each clustering algorithm.

Chapter 4

Modeling Physical Constraints

Many research areas such as spatial data mining, computational geometry, computer graphics, robot navigations, etc, have employed polygons for object generalization. Applications in research contexts model a polygon with simpler components such as a set of edges, decomposed triangles, etc. One of the dominant modeling schemes for polygons is *"Polygon Decomposition"*, which has also been actively discussed in the literature of computational geometry and various other disciplines [29, 34, 11]. However, ideas in the current literature of the polygon decomposition do not explicitly enhance the performance of applications which describe polygons as constraints and evaluate correlations among data objects in the presence of the constraints by checking visibilities between data objects. For instance, traversing a data domain to find a target destination in the presence of constraints demands the shortest path between a starting point and a destination point in order to minimize tour costs under specified constraints. Of course, the evaluation of the constraints has substantial effects on the efficiency of the navigation. Another example is exploring correlations between data objects from a very large database in the presence of obstacles [45, 17, 51].

The main purpose of the scheme that this thesis addresses – to model an obstacle using a polygon – is to cut search spaces of the applications, as the scheme converts a polygon to a set of primitive lines. Indeed, the set of primitive lines generated by the polygon reduction algorithm preserves correlations between objects in an applied domain. Notice that there are two

classes of polygons with respect to mathematical context, as illustrated in [22]: a simple polygon and a crossing polygon. A simple polygon is a polygon in which every point on the boundary in the polygon belongs only to one edge of the polygon, whereas a crossing polygon has some point of its boundary that belong to several edges shown in Figure 4.2. In this thesis, we will describe a simple polygon as a polygon since it is a dominant object in spatial applications and a crossing polygon is simply disintegrated into a set of simple polygons. We also present convexity test techniques to determine whether a polygon is convex or concave since it is essential for an application to automatically process procedures in order to minimize user involvement. In addition, the convexity tests determine the type of each point in a polygon. Such tests, in turn, have a key role in representing a polygon with a set of primitive edges.

Before we address clustering problems in the presence of constraints, we present some definitions to formalize correlations between data objects and a polygon(an obstacle).

Definition 6. (Visibility) Let *P(V, E)* be a polygon with V vertices and E edges. *Visibility* is the relation between two vertices in a planar space, if an edge drawn from one vertex to the other is not intersected by P. Given a database D of n data objects D=$\{d_1, d_2, d_3, \ldots, d_n\}$, an edge l joining vertices d_i and d_j where d_i, $d_j \in$ D, i$\neq$j, and i and j $\in [1..n]$, d_i is *visible* to d_j, if l is not intersected by any $e_k \in$E.

Definition 7. (Visible Space) Given a set D of n data objects with a polygon $P(V, E)$, a visible space S is a space that has a set D' of data objects satisfying the following

(1) Space S is defined by three edges: the first edge(edges) $e \in E$ connects two minimal convex points v_i, $v_j \in V$, the second edge f is the extension of the line connecting v_i and its other adjacent point $v_k \in V$, and the third edge g is the extension of the line connecting v_j and its other adjacent point $v_l \in V$.

(2) $\forall\ p, q \in D'$, p and q are visible from each other in S. Thus, $D' \subseteq D$.

(3) S is not visible to any other visible space S'. Thus, $S \cap S = \emptyset$.

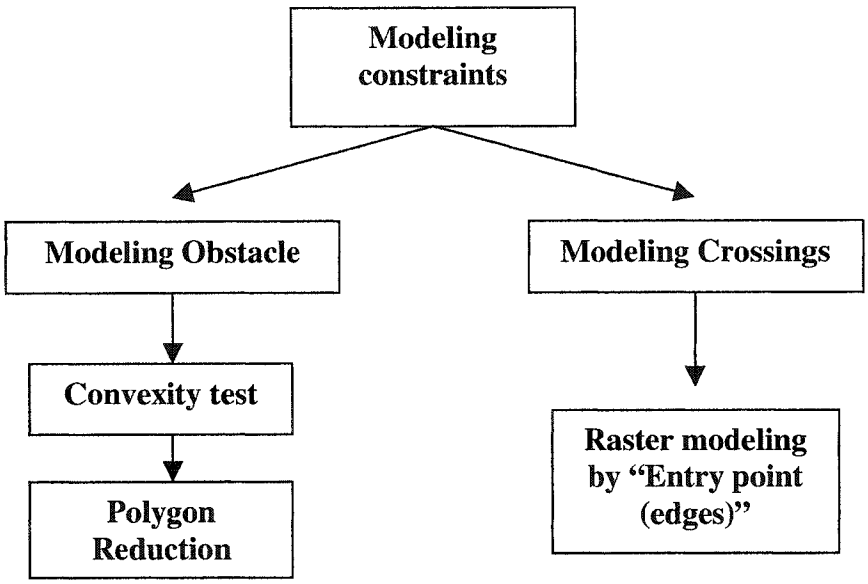


Figure 4.1: Overview of Modeling Constraints

This chapter is organized as follows. In the next section, the convexity test techniques are introduced with definitions to formalize the Polygon Reduction algorithm, and the following section presents the Polygon Reduction algorithm. Note that the Polygon Reduction algorithm is post-stage since convex or concave types of points form different correlations with relations to data objects in two dimensional planar space. An overview of modeling constraint phases is illustrated in Figure. 4.1

4.1 Convexity Test

An obstacle is described as a polygon in many applications. A polygon is represented by a set of simple components such as edges. The set of edges in a polygon forms a set of visible spaces (Definition 7) in accordance with the shape and type of the polygon associated with data domains. Thus, a polygon with its visible spaces is decomposed into a set of primitive lines called a set of " obstruction lines." The modeling procedure "Polygon Reduction" does not compromise Definition 6. It enables applications that adopt the polygon reduction algorithm to augment their efficiency.

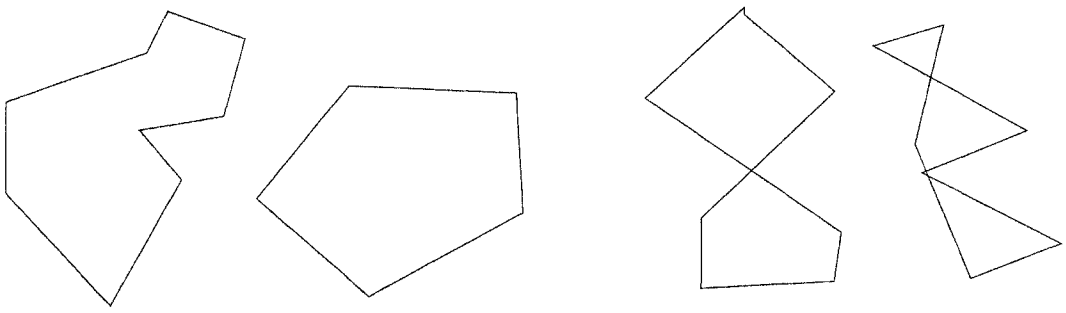
Prior to the deployment of the Polygon Reduction algorithm, the Convexity

Test is executed to categorize the type of a given polygon since the number of obstruction lines depends on the type of the polygon. In addition, we need to label the type of each vertex in a polygon through the convexity test. This in turn gives advantages to the Polygon Reduction algorithm to improve efficiency.

An applicable approach to determine the type of a polygon is the *Convex Hull algorithms* [29, 34, 3, 11] which can retrieve a set of points that enclose a given finite set of points. They are, however, limited in that they only label the type of polygon, while the type of each vertex is also required for further investigation. We herein show two methods to perform the Convexity test – the Turning Directional Approach and the Externality Approach.

4.1.1 Turning Directional Approach

The "Turning Directional" approach has already been introduced in [44]. The Turning directional approach is to evaluate the convexity of a polygon using the definition of a polygon. A polygon is classified as either a simple polygon or a crossing polygon. A simple polygon is one such that every edge in the polygon is not intersected by other edges from the polygon. A crossing polygon is one such that there is an edge that is intersected by other edges from the polygon. Examples of polygons are illustrated in Figure 4.2.



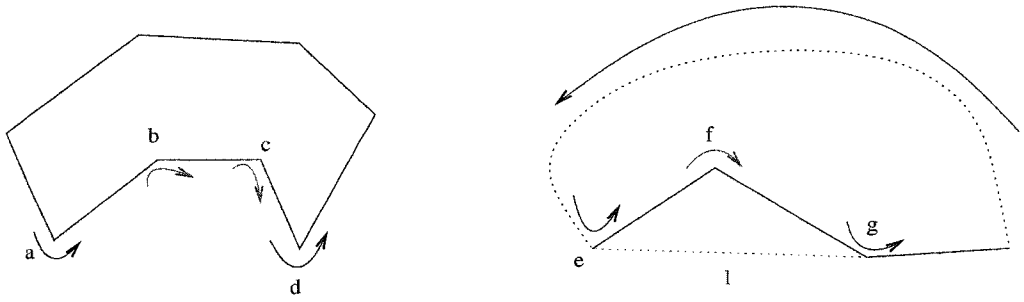
(a) Simple polygons (b) Crossing polygons

Figure 4.2: Examples of polygons

We claim that a polygon is a convex polygon if and only if all vertices from the polygon make the same directional turn. This claim can be easily proved. Suppose a polygon P does not satisfy the claim. It is obvious then that P is

not convex. As shown in Figure 4.3 (a), vertices b and c are concave vertices that make P concave. Note that the arrows in Figure 4.3 indicate the direction of each angle for each vertex.

Let P be a convex polygon. Then all possible line segment that join two non-consecutive vertices from P should be interior to P. Hence, vertex f in Figure 4.3(b) must be pulled out at least up to the line l in order for P to be convex. If the vertex f lies on the line l, then a convex polygon is composed by removing vertex f. Note that a different shape of a polygon is drawn if f lies on the line l.



(a) A concave polygon (b) Turning directions

Figure 4.3: Turning in polygons

In order to test a turning direction for 3 consecutive vertices, the sign of the triangle area of 3 vertices is examined by computing a determinant. As a result, the sign of the determinant evaluates the turning direction as either clockwise or counterclockwise. Note that we assume that all vertices in a polygon are enumerated in either clockwise or counterclockwise order. Hence, we can easily identify the type of a polygon as well as the type of each vertex in the polygon in a linear time $O(n)$, where n is the number of vertices in a polygon. Algorithm 6 illustrates the Turning Direction procedure.

4.1.2 Externality Approach

Observe that the property of a convex or a concave vertex using a line segment called an *assessment edge* whose two end vertices respectively lie in two adjacent lines of a vertex from a polygon. A vertex v is convex, if its assessment edge is interior, or concave if it is exterior to the polygon of the vertex.

Input : Three consecutive vertices *X*, *Y*, and *Z* from a polygon
Output: A direction: Clockwise, Counterclockwise, or SameLine
determinant = (X.x-Y.x)*(Y.y-Z.y)-(X.y-Y.y)*(Y.x-Z.x);
if *determinant* < *0* **then**
  Return Clockwise;
endif
if *determinant* > *0* **then**
  Return Counterclockwise;
endif
if *determinant* == *0* **then**
  Return SameLine;
endif

Algorithm 6: Convexity Test Algorithm [44]

Figure 4.4: Convex and Concave examples

As illustrated in Figure 4.4, a vertex P is a convex vertex since its edge l_1 whose two end vertices are respectively on two adjacent edges of P is always interior to a polygon. In contrast, a vertex Q is a concave vertex since its edge l_2 is exterior to a polygon.

Note that there are three possible affiliations in the course of the convexity test between a given polygon and *"an assessment edge"* which is drawn from two end vertices which respectively lie in two adjacent edges of a query vertex: *interior*, *exterior*, or *intersected* to the polygon. We must ensure that an *intersected* instance does not occur in the course of the convexity test to avoid an incorrect classification of a type of a vertex from a polygon. The *intersected* instance occurs if any vertex from a polygon exists in the triangular area that

is composed by three vertices: a tested query point and its two neighbours. In order to compute a correct *assessment edge* for the purpose of testing for the type of a vertex, we need to set a value α. Value α directs the convexity test for a vertex in a polygon to calculate an *assessment edge* that is not intersected with the polygon. Once an *assessment edge* for a query vertex in a polygon is found, it is trivial to identify a type of the query vertex in the polygon.

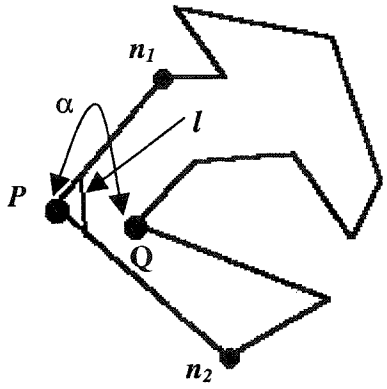


Figure 4.5: Convexity Test

Figure 4.5 illustrates the convexity test that categorizes a type of a vertex for a polygon without the enumeration of each vertex. In order to test whether the query vertex P in Fig.4.5 is convex or concave, we first need to examine whether there is a vertex Q from a given polygon in the triangular area *(P,* n_1, n_2*)* that is composed of P and its two neighbour n_1 and n_2. If there is a vertex Q in the triangle area, then we can find the perpendicular distance between P and Q, which induces an assessment edge for P. Note that if there are more than one vertices found in the triangle area, then the closest one to the query vertex is evaluated to induce value α. Value α is the distance between the query vertex and the closest vertex in the triangle area. The triangle area then allows the value α to induce an assessment edge, avoiding an *"intersected"* incident. Note that the *assessment edge* l is therefore constructed whose perpendicular distance to P is less than the value α.

Observe that the status of an *assessment edge* either interior or exterior to a given polygon is extended from the *Point Location* problem, which is discussed in [47, 22, 36]. Since an *assessment edge* is either interior or exterior

to a polygon, if any point on an assessment edge is interior (exterior), the edge is interior (exterior) to a polygon.

By examining all vertices from a given polygon, we can classify the type of a given polygon. If there is a concave vertex in a given polygon, then it is a concave polygon. Otherwise, it is convex. Algorithm 7 is illustrated to explain the External approach.

```
Input  : A Polygon
Output: The type of the polygon and the type of each vertex in the
        polygon
for all vertices queryVertex_i in the polygon do
    GetNeighbours_Storeinto(queryVertex_i, neigh_1, neigh_2);
    if there is any vertex inside of the triangle area of queryVertex_i,
    neigh_1, and neigh_2 then
        vertex = findClosestVertexTotheQueryVertexInTriangleArea
        (queryVertex_i, neigh_1, neigh_2);
        alpha = getDistanceBetween (queryVertex_i, vertex);
        line = buildAssessmentEdge(alpha);
    endif
    else
        line = drawLine(neigh_1, neigh_2);
    endif
    if assessmentEdgeIsExterior(point) then
        queryVertex_i=CONCAVE;
    endif
    else
        queryVertex_i=CONVEX;
    endif
endfor
```

Algorithm 7: Convexity Test Algorithm

4.2 Polygon Reduction Algorithm

In order to model a polygon with a set of primitive edges, we have initially categorized the type of the polygon by the convexity test we presented in the previous section. Once we have labeled the type of a polygon as well as the type of vertex for all vertices, we construct a set of primitive edges to maintain visible spaces (Definition 6). It is clear that a convex polygon has the same

number of visible spaces (Definition 7) as the number of vertices in the convex polygon since each convex vertex blocks visibility against its adjacent visible spaces. In contrast, a concave vertex does not create two visible spaces, only one visible space.

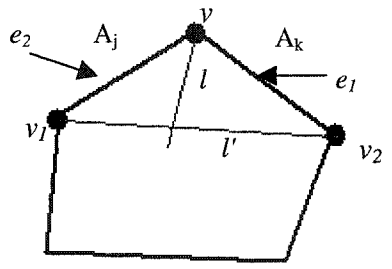


Figure 4.6: A polygon and its visible spaces

Given a polygon *P(V,E)* with V vertices and E edges, Figure 4.6 shows how two adjacent edges $e_1 \in E$ and $e_2 \in E$ from a convex point $v \in V$ obstruct visibility between visible spaces A_j and A_k, where A_j and A_k are visible spaces that are created by the convex vertex v, j and $k \in [1..n]$, $j \neq k$, and n is the number of visible spaces from the polygon. Figure 4.6 demonstrates that two edges e_1 and e_2 are replaceable with *obstruction edges* l and l'.

We observe the fact that two adjacent edges sharing a convex vertex in a polygon are interchangeable with two edges such that one of them obstructs visibility in a dimension between two adjacent visible spaces that are created by the convex vertex and the other impedes visibility between two adjacent visible spaces and the rest of visible spaces created by the polygon. As a consequence, the initial polygon is to be represented as a loss-less set of primitive edges with respect to visibility (Definition 6).

Lemma 1 *Let P (V, E) be a polygon with a set V of n points and a set E of n edges. An edge l one of whose end vertices is a convex vertex* $v \in V$ *that creates two visible spaces* A_j *and* A_k *from two adjacent edges* $e_i \in E$ *and* $e_m \in E$ *obstructs visible spaces* A_j *and* A_k *for a dimension, if the projected length of l onto the dimension is longer or at least equal to the sum of the projected length of* e_i *and* e_m, $j \neq k$.

The proof for Lemma 1 is trivial since every line that is intersected with e_i

and e_m in a dimension, both of which share one convex vertex, is intersected with an edge that exists between e_i and e_m, and shares the one vertex shared by e_i and e_m. In Figure 4.6, the edges l and l' replace two edges e_1 and e_2 according to Lemma 1. By following Lemma 1, we model a polygon with a set of primitive edges to cut search spaces.

We introduce the following definition according to Lemma 1.

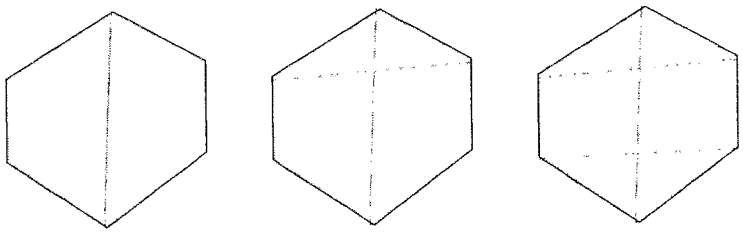
Definition 8. (An Obstruction line) Let *P (V, E)* be a polygon with a set of V vertices and a set of E lines. An obstruction line l is an edge that is introduced by *Lemma 1* and whose two end vertices are points $v_i \in V$ and $v_m \in V$, while it is interior to P and not intersected with any $e \in E$. An obstruction line of a convex point $v \in V$ from the polygon P obstructs in a dimension the two visible spaces A_j and A_k created by two adjacent edges e_1 and e_2 of v.

It is sufficient to discover a set of obstruction lines that blocks visible spaces for every convex vertex from a polygon by minimizing the number of obstruction lines. It is obvious that we reduce N lines for a convex polygon to

$$\lceil \frac{N}{2} \rceil$$

obstruction lines. Using a bi-partition method that divides a set of convex vertices into two sets according to an enumeration order – clockwise or count-clockwise, it is straightforward to compose a set of obstruction lines by joining two vertices from each partition. In addition, the bi-partition method achieves the loss-less reduction of a polygon. Figure 4.7 for a convex polygon illustrates steps to generate a set of obstruction lines that are red coloured.

An obstruction line is easily drawn between two convex vertices in a convex polygon. Doing so is , however, not trivial for a concave due to its shape. It is necessary in the case of a concave polygon to check whether a line, as a possible obstruction line candidate, which is constructed by the bi-partition method, is intersected with an edge from the concave polygon or is exterior to the concave polygon. Recall that every line segment created by the bi-partition method for a convex polygon is admissible for an obstruction line.



(a) Step1 (b) Step2 (c) Step3

Figure 4.7: Steps of Polygon Reduction

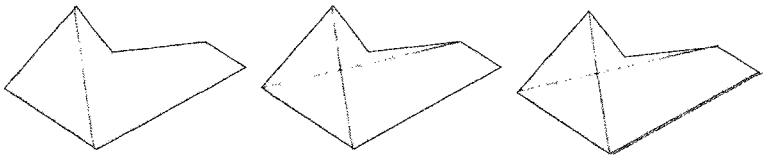
We need to predefine a set of possible obstruction line candidates for a concave polygon. For instance, if a possible obstruction line candidate is intersected by an edge to a given polygon or is exterior to it, then it is replaced with a set of obstruction lines that can be edges of the concave polygon or be interior to and not intersected by the concave.

In order to construct a set of obstruction edges from a concave polygon, when a possible obstruction edge candidate is not admissible, we employ a single-source shortest path algorithm [10] which converts a concave *P(V, E)* into a weighted graph whose edges are a set of admissible obstruction lines drawn from each vertex in the concave and the weight of whose edge is the distance between a source and a destination. Note that the source and the destination vertex are two end vertices from a non-admissible obstruction line generated by the bi-partition method. The weight of an edge is 1. The weight of a path from v_1 and v_n is then defined as follows:

$$Weight(\overrightarrow{v_1 v_n}) = \sum_{i=1}^{n-1} Edge_i \tag{4.1}$$

In Equation 4.1, $Edge_i$ is the weight of an i^{th} edge $\in \overrightarrow{vw}$, and the path $\overrightarrow{vw}$ is a set of admissible obstruction lines. Figure 4.8 illustrates steps of the polygon reduction algorithm for a concave polygon.

We describe Polygon Reduction algorithm in detail in Algorithm 8. Line 1 in Algorithm 8 takes advantage of the convexity test to obtain a vertex type using the Convexity Test previously performed since a convex vertex only forms visible spaces. Constructing a bi-partition enhances efficiency and effectiveness when building a set of obstruction edges in view of the fact that bi-partitioning convex vertices assures the loss-less decomposition of a polygon in



(a) Step1 (b) Step2 (c) Step3

Figure 4.8: Steps of the Polygon Reduction algorithm

a two dimension plane, easily allowing obstruction line candidates to be drawn, particular for a convex polygon. Once a possible obstruction line candidate in Line 4 is outlined, it is necessary to verify whether the edge is admissible to its polygon, as illustrated in Line 5. For a convex polygon, the verification is not necessary due to its property that $\forall$ v_i and v_j are visible to each other, where v_i and v_j are vertices that exist inside of a convex polygon. On the other hand, a concave polygon should complete the affirmation, that is, an obstruction line is admissible. An obstruction line candidate that fails the verification in Line 5 is substituted for with a set of minimal obstruction lines retrieved by the transformation from a concave polygon to a weighted graph. Those steps from Line 3 to Line 10 are iterated until all convex vertices are processed from the partitions. The last obstruction line is drawn in Line 11.

Algorithm 9 explains the step in Line 6 of Algorithm 8, which generates a set of admissible obstruction lines.

Lemma 2 *Given a polygon P(V, E) with n_{cv} number of convex vertices, a set of obstruction lines E' generated by Algorithm 9 preserves visibility relations formed by an input edge l joining the two end vertices s and d.*

Proof: Note that the bi-partition method allows the Polygon Reduction algorithm to prune the search spaces efficiently. The first obstruction line joining the first convex vertex and the middle convex vertex from P covers $\lfloor \frac{n_{cv}}{2} \rfloor \cdot (n_{cv} - \lfloor \frac{n_{cv}}{2} \rfloor)$, the number of visible spaces created by P. In addition, the first obstruction line allows each partition to find its set of obstruction lines such that the set of obstruction lines covers the visible spaces in each partition. Therefore, it is sufficient to construct a set of obstruction lines that cover the adjacent visible spaces of each convex vertex from each partition.

```
Input  : A Polygon
Output: A set of obstruction lines
1  Get a convex vertex list and create bi partition by an enumeration
   order;
2  In the initial of for loop, create an obstruction line between first vertex
   in first partition and the middle convex vertex from the convex list
   which follows an enumeration order, and check if it is admissible. If
   not, then do Line 6 and Line 7;
3  for all elements from each partition do
4  |  create an obstruction line candidate from the bi-partitioned sets and
   |  check if the line is admissible;
5  |  if the line is not admissible then
6  |  |  find a set of obstruction lines to replace the obstruction line
   |  |  candidate;
7  |  |  put them into obstruction line list;
   |  endif
8  |  else
9  |  |  put it into obstruction line list;
   |  endif
10 |
   endfor
11 draw an admissible obstruction line for the last convex vertex and put
   it into obstruction line list;
12 Return obstruction line list;
```

Algorithm 8: Polygon Reduction Algorithm

Note that each obstruction line is drawn by joining two vertices from two partitions respectively. Since the types of two vertices s and d are convex, as shown in Figure 4.9, there are 4 possible visible spaces,vp_1, vp_2, vp_3, and vp_4, around the vertices if there is a convex vertex between s and d. Note that visible spaces are enumerated in an order such that a visible relation between a precedent visible space and its succeeding visible spaces is covered first. In other words, the visible relation between a visible space and its precedent visible spaces are not required to be examined. Hence, the non-admissible obstruction line joining $s \in V$ and $d \in V$ has the role of impeding the visible relations between vp_1 *and* vp_2 and between vp_3 *and* vp_4. The visible relation between vp_1 *and* vp_2 is covered by s, and d covers the visible relation between vp_3 *and* vp_4. A set of admissible obstruction line that in turn replaces the

Input : a source S and a destination D vertices in a concave polygon with its adjacency matrix

Output: a shortest path between S and D

build a weighted graph from the input polygon sort the vertices topologically u = S;

for *each vertex* $v \in Adj[u]$ **do**

  Select a path uv with minimum weight;

1 **if** *each vertex* $v' \in Adj[v]$ *has not been inserted into* $Adj[u]$ **then**

    Put vertex v' into Adj[u] ;

    //The weight of $\overrightarrow{uv'}$ is v+1;

    //since v' is adjacent to v;

    Set a weight(u, v', w);

    if $v' == D$ **then**

      break;

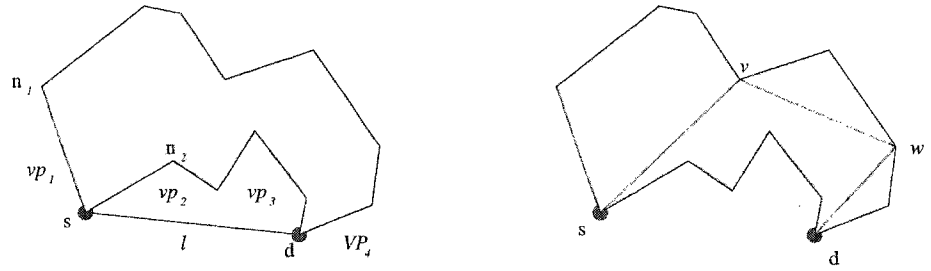
    endif

  endif

endfor

Return a shortest path between S and D;

Algorithm 9: Shortest Path between two vertices



(a) A non-admissible edge (b)A set of admissible edges

Figure 4.9: A polygon and its visible spaces

non-admissible obstruction line should have the same role as s and d. An edge $e'_i \in E'$ joining two vertices s and $v \in V$ maintains visible relations that are created by two adjacent edges of s since every line segment that is intersected by two adjacent edges of s intersects with e'_i. By following the same step, the visible relations that are constructed by d are covered.

Note that a concave polygon is represented by an adjacency matrix that provides a set of paths that connects two vertices directly and is built on the stage of the convexity test. According to Algorithm 9, given a source and a destination vertex, the weight on a path from the adjacency is being updated,

as the shortest path between two vertices is computed in $O(E')$, where E' is the number of edges from the adjacent matrix, and edges are admissible. Each vertex that is for the first time visited is inserted into an adjacency list with its weight. And the added path is extended by choosing the minimum weighted path with a minimum increase of a weight, that is, 1. Again the weight of a path is not the *"Euclidean"* distance, but the number of precedent vertices from a source(Refer to Equation 4.1).

The proof that Algorithm 9 finds the shortest paths between two vertices is trivial since any adjacent vertices to a source vertex are relaxed by updating the weight of the path between a source and its adjacent vertices. The update is expanded by the vertices that are adjacent to the adjacent vertices of a source. Suppose the shortest path $s, \ldots, v, d$ is found by Algorithm 9. Then the path $\overrightarrow{sv}$ is the shortest path that has a minimum weight discovered by Algorithm 9. The adjacent vertices of v are examined in Line 1. If one vertex d that satisfies the condition of Line 1 is inserted and the weight is assigned to the path $\overrightarrow{sd}$, the minimum increment, which is 1, is applied to the path and the destination vertex has been reached.

In this section, we re-articulate the Definition 4 since the problem this thesis investigates considers obstacles that influence the precision of clustering data objects. Prior to modifing the concept of the Definition 4, we need to define the following definitions. Those definitions are extended from DBSCAN in order to take into account disconnectivity constraints. The illustration of examples of each notion is shown in Figure 4.10.

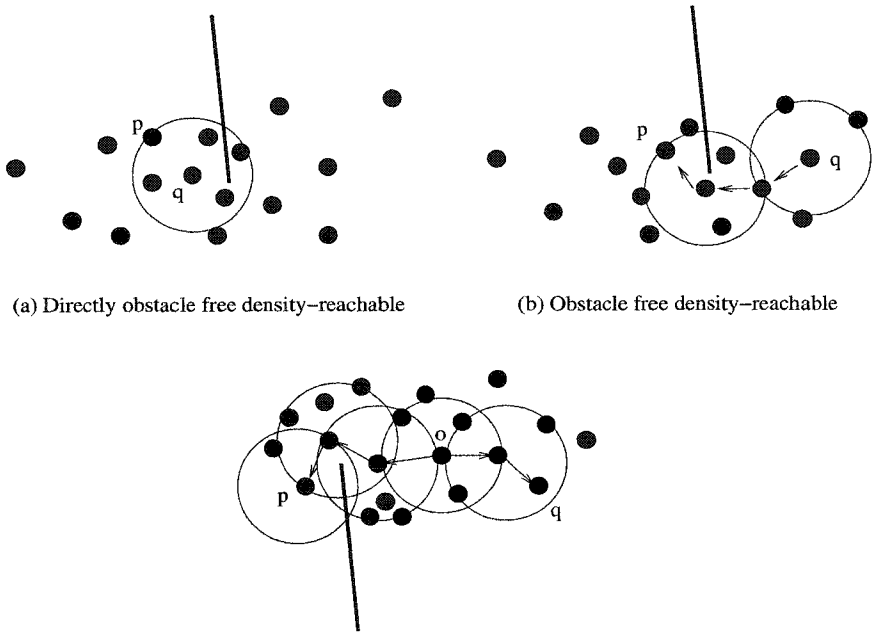
Definition 9. (Directly obstacle free density-reachable) A point p is directly obstacle free density-reachable from a point q with respect to *Eps*, *MinPts* if

(1) $p \in N_{Eps}\ (q)$ and

(2) $|N_{Eps}\ (q)| \geq MinPts$, where $|N_{Eps}(q)|$ denotes the number of points in the circle of radius *Eps* and centre q

(3) an edge joining p and q is not intersected with any obstacle

Figure 4.10 (a) shows that directly density reachable from DBSCAN is ex-



(a) Directly obstacle free density–reachable

(b) Obstacle free density–reachable

(c) Obstacle free density–connected

Figure 4.10: Obstacle free density notions(*Eps=2cm* and *MinPts=4*)

tended to *directly obstacle free density-reachable.* Data point p is directly density reachable from q with respect to *Eps* and *MinPts.* In addition, p and q are visible from each other in spite of *an obstruction line* which is present in *Eps-neighbourhood* of q. Therefore, p is *directly obstacle free density-reachable* from q. In Figure 4.11 (a), the data point p is not *directly obstacle free density-reachable* from the data point q since an obstruction line k blocks visibility between p and q, although p is directly density reachable from q.

Definition 10. (Obstacle free density-reachable) A point p is obstacle free density-reachable from a point q with respect to Eps and MinPts if there is a chain of points p_1, .., p_n, p_1 = q, , p_n = p such that p_{i+1} is directly obstacle free density-reachable from p_i.

Note that data points p and q, which are not visible from each other, are *obstacle free density-reachable* in Figure 4.10 (b) since there is a chain that makes p and q *directly obstacle free density-reachable.* Figure 4.11 (b) illustrates a case where p is not *obstacle free density-reachable* from q due to obstruction line l. Note that obstruction lines k and l in Figure 4.11 extend beyond dense areas

such that two data points p and q are not reachable with respect to density notions.

Definition 11. (Obstacle free Density-connected) A point p is obstacle free density-connected to a point q with respect to *Eps* and *MinPts*, if there is a point o such that both p and q are obstacle free density-reachable from o with respect to *Eps* and *MinPts*.

Figure 4.10 (c) illustrates that two data points p and q are *obstacle free density-connected* by a data point o since both p and q are *obstacle free density-reachable* from o despite the presence of an obstruction line. Data points p and q in Figure 4.11 (c) are not *obstacle free density-connected* due to obstruction line m, although q is *obstacle free density-reachable* from o. Hence, the Definition 4 are redefined to integrate obstacle entities.

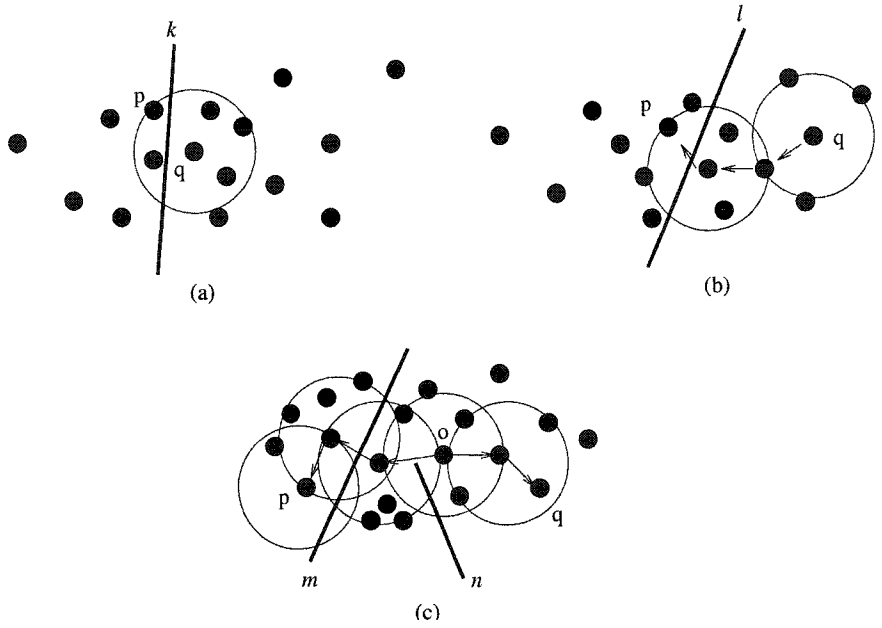


Figure 4.11: Examples of non obstacle free density-reachable(*Eps=2cm* and *MinPts=4*)

Definition 12. (Cluster) Given a set D of n data objects D=$\{d_1, d_2, d_3, \ldots, d_n\}$, a cluster is a set C of c data objects C=$\{ c_1, c_2, c_3, \ldots, c_c \}$, where C $\subseteq$ D. Let D be a database of points. A cluster C with respect to *Eps* and *MinPts* is a non-empty subset of D satisfying the following conditions: Let i and j $\in$ [1..n] such that i $\neq$ j.

(1) Maximality. $\forall$ d_i, d_j q if $d_i \in C$ and d_j is obstacle free density-reachable from d_i with respect to *Eps* and *MinPts*, then $d_j \in C$.

(2) Connectivity. $\forall$ c_i, $c_j \in C$, c_i is obstacle free density-connected to c_j with respect to *Eps* and *MinPts*.

4.2.1 Correctness of the Polygon Reduction algorithm

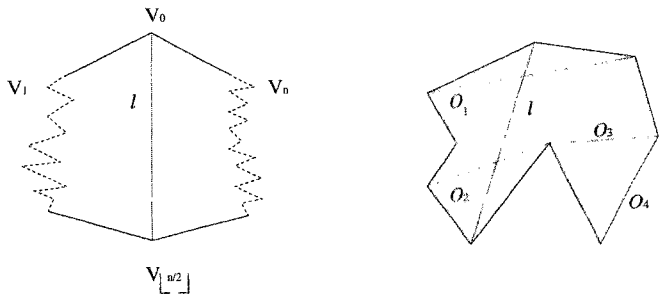
Theorem 1 *Given a polygon P(V, E), the Polygon Reduction algorithm models P with a set of obstruction lines, preserving visibility relations between visible spaces S formed by P.*

Proof : Given a polygon P, the convexity test algorithm classifies the type of P – *convex* or *concave* – where V is a set of vertices and E is a set of edges in P. Vertices are enumerated in order – either clockwise or counterclockwise. Based on Lemma 2, a set of obstruction lines generated by Algorithm 8 preserves the visibility relations of visible spaces that are created by a non-admissible edge in Algorithm 9.

Let n, n_{cc}, and n_{cv} be the number of vertices, the number of concave vertices, and the number of convex vertices in P respectively. The total number of visible relations produced between visible spaces in P is then

$$\frac{n_{cv} \cdot (n_{cv} - 1)}{2} \tag{4.2}$$

Note that the first obstruction line created by the bi-partition method completely obstructs the visible relations of the visible spaces between two partitions since the visible spaces in one partition are blocked from the visible spaces from the other by the first obstruction line. Consider n visible spaces in a polygon $P(V, E)$. The bi-partition method creates two partitions, Bp_1 and Bp_2, of V. The first obstruction line is drawn by joining two vertices $v_0 \in Bp_1$ and $v_{\lfloor \frac{n}{2} \rfloor} \in Bp_2$. The visible spaces that are created by vertices $v_j \in [v_0, v_{\lfloor \frac{n}{2} \rfloor})$ are completely invisible to the visible spaces that are created by vertices $v_k \in [v_{\lfloor \frac{n}{2} \rfloor}, v_n)$. See Figure 4.12 (a). Obstruction line l impedes the visibility between two partitions. In other words, the visible spaces composed by P are blocked for a dimension between Bp_1 and Bp_2.



(a) An initial obstruction line (b) Obstruction lines

Figure 4.12: A polygon and an obstruction line

After the first obstruction line is drawn, the Polygon Reduction algorithm is sufficient to build a set of obstruction lines such that each obstruction line for a convex vertex in a partition impedes adjacent visible spaces. Each obstruction line intersects with the first obstruction line, since each obstruction line joins two vertices from two partitions respectively. This in turn ensures that the visible relations between visible spaces in each partition are maintained for two dimensional space. Figure 4.12 (b) provides an example of a set of obstruction lines. Once the initial obstruction line l is drawn, other obstruction lines O_i, i$\in$ [1..4], joining two convex vertices from each partition are constructed.

Hence, it is sufficient to show that the obstruction lines generated by the Polygon Reduction algorithm preserve the same number of visible relations. The first obstruction edge O_1 from Line 3 in Algorithm 8 covers

$$\lfloor\frac{n_{cv}}{2}\rfloor\cdot(n_{cv}-\lfloor\frac{n_{cv}}{2}\rfloor) \tag{4.3}$$

visible relations due to an advantage of bi-partitioning. O_i covers

$$\lfloor\frac{n_{cv}}{2}\rfloor-(i-1)+\lceil\frac{n_{cv}}{2}\rceil-(i-1), 2\leq i\leq\lceil\frac{n_{cv}}{2}\rceil \tag{4.4}$$

visible relations. The total number of visible relations that have been covered by Algorithm 8 are as follows.

$$\lfloor\frac{n_{cv}}{2}\rfloor\cdot(n_{cv}-\lfloor\frac{n_{cv}}{2}\rfloor)+\sum_{i=1}^{\lfloor\frac{n_{cv}}{2}\rfloor}(\lfloor\frac{n_{cv}}{2}\rfloor-i)+\sum_{i=1}^{\lceil\frac{n_{cv}}{2}\rceil}(\lceil\frac{n_{cv}}{2}\rceil-i) \tag{4.5}$$

Note that equations 4.2 and 4.5 are equal. Therefore, the total number of visible relations produced by P are completely maintained by Algorithm 8.

In this section, we have addressed the problem of modeling polygons (obstacles) into a set of obstruction lines as a sub-procedure of the clustering problem in the presence of constraints. This thesis presents two algorithms: the Convexity Test and the Polygon Reduction algorithms. To the best of our knowledge, there is no ongoing research to model a polygon into a set of obstruction edges, preserving visible relations created by the relations between the polygon and data objects. In many domains dealing with complex data types such as geographical information systems, spatial data mining, computer graphics, CAD, etc, it is critical to preclude search spaces due to the limitations of the algorithms themselves. With respect to visibility relations between polygons and data objects in a two dimensional planar space, the Polygon Reduction algorithm as a pre-processing stage significantly improves the pruning of search spaces by modeling a polygon into a set of obstruction edges.

In addition, the Convexity Test succeeds in minimizing user interactions to enhance automatic procedures. Note that it is essential to analyze problems avoiding external users' involvement.

The algorithms introduced in this thesis, however, suffers from the problem of the dimensionality of a polygon since they apply only to two dimensional planes, while many complex data have multidimensional structures.

We believe that more deliberation should be given to multidimensional planes for real applications. For instance, objects in three dimensional spaces can be considered with obstruction hyper-planes that block visible relations between objects. Note that we do not integrate an indexing structure for edge objects in this thesis since doing so goes beyond the main idea of modeling a polygon as a set of obstruction edges. An indexing scheme, however, would enhance the processes of the algorithm improving the search costs, that is, to check whether an edge is intersected by an edge from a polygon in $O(E')$, where E' is the number of edges in an adjacency matrix for a polygon.

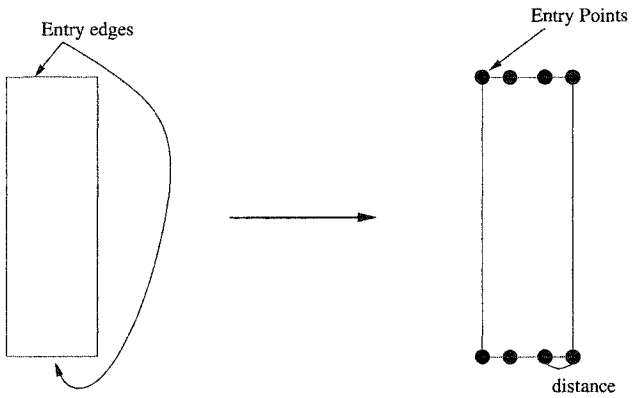
4.3 Modeling Crossing

In this section, we present a modeling scheme of a constraint *Crossing (Bridge)* in a two dimensional planar space. Before formalizing a crossing that can connect data points from different clusters, we need a modeling scheme to consign connectivity functionality of a bridge as well as to control connectivity flow for a wide range of applications. For this purpose, we introduce *"Entry point"* and *"Entry edge"* notions. An *Entry point* is a point on the perimeter of the polygon crossing when it is density-reachable (Definition 2) from a given point p with respect to *Eps*. As a result p becomes reachable by any other point x density-reachable from any other Entry point of the same crossing with respect to *Eps*. In other words, given two different Entry points, p_1 and p_2, at two extremities of a crossing; a point a is density-reachable to p_1 with respect to *Eps*; and a point b is density-reachable to p_2 with respect to *Eps*, a and b are then density connected (Definition 3). An *Entry edge* is an edge of a crossing polygon with a set of Entry points starting from one endpoint of the edge to the other separated by an interval value i_e where $i_e \leq Eps$. The descriptions of Entry points and Entry edges are amalgamated with the definition of crossings as follows.

Definition 13. (Crossing) A crossing (or bridge) is a set B of m points generated from all Entry edges. By definition any point $b_m \in B$ is reachable by all other points in B. Before a bridge is modeled, the bridge B is denoted by *B(P, E)*, where P is a set of Entry points and a set of Entry edges E. Thus a bridge *"connects"* objects such as clusters or data points that are $Eps - reachable$ from all Entry points generated from the bridge where $Eps - reachable$ of an Entry point is any data point which is in an Eps-neighbourhood. The $Eps - reachable$ are not affected by any obstacle entities. In other words, crossing entities have a priority over obstacle entities, unless otherwise specified.

According to the properties of a bridge, there are two significant concepts: an Entry point and an Entry edge. The functionality of a crossing is effected

by Entry points and Entry edges (Definition 13). In other words, it is not necessary to define every edge in a crossing as an Entry edge. Once a set of Entry edges is defined, it is replaced with a set of Entry points, starting from one endpoint of the edge to the other, separated by an interval value i_e where $i_e \leq Eps$. Thus, every Entry point from a crossing expands a reachable cluster. Notice that the flow of connectivity is controlled by how Entry edges are defined in a crossing. That implies the malfunction of a bridge, such that a bridge is in turn impeded, if an obstacle traverses a bridge. As a result, the bridge is no longer a *"bridge"*. The Entry edge and Entry point allow their applications to control priority between an obstacle and a crossing; to control connectivity flow or connectivity coverage, such that connectivity functionality is able to be applied in either a direction or bi-direction, and a subset of entry edges can be engineered with the connective function.



(a) Before modeling a crossing (b) After modeling a crossing

Figure 4.13: Illustrating modeling a crossing: Entry edges and Entry points

Figure 4.13 illustrates how a bridge has been modeled applying Entry point and Entry edge concepts. Once a crossing is modeled, all Entry points whose interval *distance* i_e is $i_e \leq Eps$ merge clusters for all possible data points and clusters. Note that two edges in Figure 4.13 (a) are not labeled as *Entry edges* such that they are not assigned connectivity functionality.

Chapter 5

DBCluC

The algorithm that this thesis presents, DBCluC (Density-Based Clustering with Constraints, pronounced DB-clu-see), is based on DBSCAN [15] a density-based clustering algorithm that significantly outperforms, in terms of its effectiveness and efficiency, CLARANS [35]. Note that COD-CLARANS incorporate CLARANS into its main frame. DBSCAN's performance is better not only in terms of time complexity, but also in terms of clustering quality, for example, the detection of natural cluster shapes and noise (outliers) sensitivity.

As previously mentioned, the significance and effects of constraints, especially physical constraints in clustering issues, should be taken into account in clustering procedures in a cost-effective way. For instance, indexing schemes for data objects such as points, lines, and polygons will significantly help augment processing phases. However, none of the indexing schemes is able to compress the number of data objects such that a loss of data information does not occur. Hence, this thesis investigates the clustering algorithm DBCluC, which efficiently takes into account constraints and effectively clusters large databases.

DBCluC initially manipulates constraints such that a set of obstacles is modeled by the Polygon Reduction algorithm, which is introduced in Chapter 4. Upon finishing the modeling procedure, DBCluC groups data objects in the presence of the modeled obstacles and crossings that are formed by the new concepts Entry points and Entry edges. Note that crossings are considered in

the course of the clustering step. The following figure illustrates the general procedures of DBCluC.

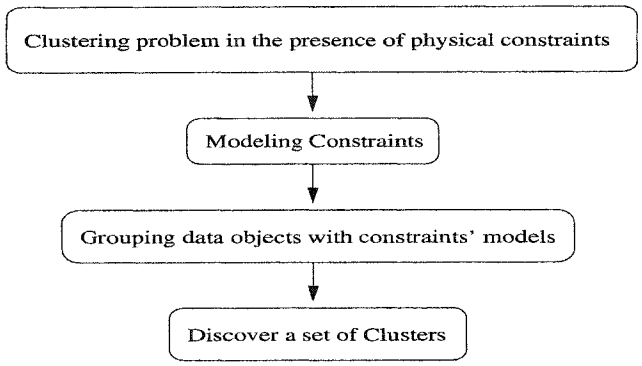


Figure 5.1: An overview of DBCluC

5.1 DBCluC Algorithm

We have seen how to model constraints – obstacles and crossings. Obstacles are modeled by the Polygon Reduction algorithm to prune search spaces. Crossings are modeled with respect to two principles: Entry edge and Entry point. These in turn allow to improve the flexibility of the algorithm, for instance, controlling data flow and assigning priority. Recall that Figure 5.1 generalizes the entire procedures of the DBCluC algorithm.

Once we have condensed obstacles using the polygon reduction algorithm and modeled crossing constraints, DBCluC starts the clustering procedure from an arbitrary data point. This is the advantage of DBCluC in that the performance is not sensitive to an input order. Due to the arbitrary selection of an initial starting point, DBCluC can consider crossing constraints *after or while* clustering data points. This enables DBCluC to be flexible in revising discovered clusters. The clustering procedure in DBCluC is similar to that of DBSCAN [15], with respect to the density notion. Hence, all definitions introduced in Section 2.3.1 are inherited in DBCluC.

Normally, clusters discovered by [15] that are not satisfied by Definition 12, and where the distance between them is larger than *Eps*, are forced to be apart. Consequently, DBSCAN does not correctly produce groups of data points in

the presence of constraints. In order for an algorithm to resolve the clustering problem in the presence of constraints, the constraints should be efficiently processed to produce correct clustering result. Using the Polygon Reduction algorithm, DBCluC efficiently performs the clustering of data objects with obstacles. In addition, DBCluC groups distant clusters with *Entry edges* and *Entry points* from crossing constraints.

```
   Input  : Database, Crossings, and Obstacles
   Output: A set of clusters
1  // While clustering, bridges are taken into account;
2  Start clustering from Entry points of crossings ;
3  for Remaining Data Points Point from Database do
4  |  if ExpandCluster(Database,Point,ClusterId,Eps,MinPts,Obstacles)
   |  then
5  |  |  ClusterId = nextId(ClusterId);
   |  endif
   endfor
```

Algorithm 10: DBCluC

In Algorithm 10, crossing constraints are taken into account while clustering data objects. *DBCluC* maximally expands a set of clusters such that all data points that are reachable by crossings are grouped together. Note that *DBCluC* can also consider crossing constraints after clustering. However, when it comes to dynamic evaluation of correlations between data objects and constraints, the crossing constraints must be processed in the course of clustering.

"Database" is a set of data points to be clustered in Algorithm 10. In this thesis, the database is limited to two dimensional space for experimental purposes. Line 2 initiates the clustering procedure from a set of entry points that are modeled from crossing constraints. Thus, a set of data objects is maximally grouped according to the crossing connectivity defined by a set of entry points in crossing constraints. Note that Line 2 uses the module of ExpandCluster for all entry points that are modeled from crossing entities. Once a maximum set of clusters is discovered after Line 2, Line 3 builds up a cluster from data objects that are not reachable by the crossing connectivity

```
   Input  : Database, a data point Point, ClusterId, Eps, MinPts, and
            Obstacles
   Output: True or False
 1 SEED = RetrieveNeighbours(Point, Eps, Obstacles);
 2 if size of seed is less than MinPts then
 3 |   Classify Point as NOISE;
 4 |   Return False;
   endif
 5 change clustered of all elements in SEED into ClusterId;
 6 delete Point from SEED;
 7 while SEED.SIZE < 0 do
 8 |   CurrentPoint = SEED.first();
 9 |   RESULT = RetrieveNeighbours(CurrentPoint,Eps,Obstacles);
10 |   if RESULT.SIZE ≥ MinPts then
11 |   |   for element ∈ RESULT do
12 |   |   |   if element is UNCLASSIFIED then
13 |   |   |   |   put it into SEED;
14 |   |   |   |   set its cluster id to ClusterId;
   |   |   |   endif
15 |   |   |   if element is NOISE then
16 |   |   |   |   set its cluster id to ClusterId;
   |   |   |   endif
   |   |   endfor
   |   endif
17 |   delete CurrentPoint from SEED;
   endw
18 Return True;
```

Algorithm 11: ExpandCluster(Database,Point,ClusterId,Eps,MinPts,Obstacles)

in the database. In the course of clustering, Line 5 assigns a new cluster id for the next expandable cluster.

The ExpandCluster in Algorithm 11 may seem similar to the function of the DBSCAN. However, the distinction is that obstacles are considered in RetrieveNeighbours (Point, Eps, Obstacles), illustrated by Algorithm 12. Given a query point, neighbours of the query point are retrieved using SR-tree. In DBCluC, we have adopted the range neighbour query approach instead of the nearest neighbour query approach from SR-tree, since it is extremely difficult for the latter to expand a set of clusters if a density of data objects is high. The average run time of a neighbour query in SR-tree is $O(\log N)$, where

N is the number of data objects. Notice that the range search in SR-tree is very expensive, especially when the density is very high with a large database. Once neighbours of a query point have been retrieved, it is trivial to evaluate visibilities between a query point and its neighbours. The visibility between two data objects in the presence of obstacles is computed using a line segment joining two data objects. If any obstruction line representing an obstacle is intersected with a line segment joining a query point and a data object, then the data point is excluded from N_{eps} for the query point, since they are not visible from each other according to Definition 7. However, if a priority of a bridge is higher than that of an obstacle, then the data objects are grouped in spite of the visible relations between two objects. This is one key advantage in the DBCluC for wide applicable domains. It is not available to COD-CLARANS and AUTOCLUST+. Those accepted neighbours defined as the SEED that are retrieved by RetrieveNeighbours of Algorithm 11 continue to expand a cluster from elements of the SEED, if the number of elements in the SEED is not less than MinPts. A data object is labeled by an assigned cluster id, if retrieved neighbours are satisfied with the parameter *MinPts* excluding outliers. Note that Line 10 in Algorithm 11 excludes a noise from being an element of the SEED in order to enhance query efficiency.

```
   Input  : a data object Point, Eps, and Obstacles
   Output: A set of data points
1  RESULT = getNeighbour(Point, Eps);
2  for element e RESULT do
3  |   if CheckVisibility_with(e, Point, Obstacles) then
4  |   |   RESULT.delete(e);
   |   endif
   endfor
5  Return RESULT;
```

Algorithm 12: RetrieveNeighbours(Point, Eps, Obstacles)

The "RESULT" in Algorithm 12 is a set of data objects that are neighbours of given query objects. The elements in the RESULT are collected and the obstacles are evaluated by Algorithm 12. The RESULT elements are constructed by removing data objects that are not visible from a query point

Point because of the blockage by obstacles. This task is performed by Line 3 in Algorithm 12. Notice that line 1 in Algorithm 12 retrieves neighbours of a given query point, using SR tree [27].

5.2 Complexity

As discussed in the previous chapters, the Polygon Reduction algorithm models obstacles and bridges by classifying an obstacle as either convex or concave. Let n be the number of points (edges) of a polygon P, and n_{cc} and n_{cv} be the number of concave and convex points, respectively. Then,

$$n = n_{cc} + n_{cv} \tag{5.1}$$

For the Turning Direction approach in the Convexity test, the complexity to classify a type of P as well as to label a type of every point in P is $O(n)$, and the complexity of the Externality method is $O(n^2)$. The Polygon Reduction algorithm requires a weighted graph to replace a non-admissible obstruction line segment with a set of admissible line segments. The complexity in the replacement is, in the worst case, $O(E)$ using an adjacency matrix that represents admissible connectivity between vertices from a polygon, where E is the number of all possible obstruction edges, including a set of line segments that lies in P. Note that E is less than $\alpha \cdot n$, where $\alpha \ll n$. Therefore, the Polygon Reduction algorithm for P requires

$$O(n \cdot \log n + n_{cv} \cdot I \cdot E) \tag{5.2}$$

in the worst case, where I is the number of non-admissible line segments to be replaced. In the contrary, if a polygon is convex, then its complexity is $O(n)$. The upper bound of the polygon reduction algorithm is as follows:

$$O(n \cdot \log n + n_{cv} \cdot I \cdot n) = O(n \cdot \log n + n_{cv} \cdot I \cdot (n_{cc} + n_{cv}) = O(n \cdot \log n + n_{cv} \cdot n_{cc} + n_{cv}^2) \tag{5.3}$$

Since we evaluate a set of polygons, and n_{cc} and I are on average far smaller than n, the complexity of the polygon reduction is absolutely less than $O(n^2)$.

Note that the Polygon Reduction algorithm is a pre-processing phase that precedes the clustering step. The complexity of the clustering algorithm alone is in the order of $O(N \cdot \log N \cdot L)$, where L is the number of obstruction lines generated by the polygon reduction algorithm, and N is the number of points in the database. The complexity can, however, be reduced to $O(N \cdot \log N)$, if we adopt an indexing scheme for obstacles. Currently, all obstruction lines are tested to check the visibility between two data points. We can reduce the number of obstruction lines, which are examined by evaluating lines that only traverse neighborhoods of a query. Therefore, the total complexity of DBCluC, including the pre-processing stage, would be $O(N \cdot \log N)$.

Chapter 6

Experiments and Evaluations

In this chapter, we evaluate the performance of the algorithm with respect to effectiveness and scalability. Although COD-CLARANS and AUTOCLUST+ investigate the clustering problem in the presence of obstacles, it is hard to compare quantitatively the performance of DBCluC with that of other approaches, because of the different data sets tested. To realistically compare the algorithms, we ought to use the exact data sets with the same constraints. Since we do not have access to these data sets and technically cannot generate the same data sets, we cannot accurately compare the performance of COD-CLARANS and AUTOCLUST+ with that of DBCluC. Yet, it is known that a density-based clustering algorithm such as DBSCAN outperforms a partitioning algorithm such as CLARANS, when it comes to efficiency and effectiveness. For instance, CLARANS cannot properly detect a non-sphere shaped cluster, whereas DBSCAN can find an arbitrary shaped cluster. In addition, memory management and insensitivity to noise are well handled by DBSCAN, while CLARANS shows poor management. Hence, we clearly infer DBCluC's performance, based on density-based clustering, is of better quality than that of COD-CLARANS, since COD-CLARANS inherits the shortcomings of CLARANS. We have evaluated DBCluC by generating synthetic data sets with complex cluster shapes and by varying the size of data as well as the number and difficulty of the physical constraints.

6.1 Experiments

For the purpose of the experiments, we have generated synthetic datasets. We report four of them herein DS1, DS2, DS3, and DS4. Bridges and obstacles such as rivers, lakes, and highways are also simulated in these datasets. DS1 containing 434 data points with four obstacles, is used for illustration purposes. Figure 6.1 shows the 16 polygon line segments reduced to 8 obstruction lines. Since DS1 is sparse, it is primarily grouped into one cluster. Adding obstacles creates four distinct clusters (Figure 6.1(c)). DS2 and DS3 presents clusters in the presence of obstacle and crossing entities. DS2 has 1063 data points, 4 obstacles, and 2 crossings. There are visually 6 clusters ignoring obstacles and crossing constraints, as shown in Figure 6.2(b). DS3 has 11775 data points with 6 obstacles that consist of 29 line segments and 2 bridges. The initial 29 line segments from simulated obstacles in DS3 are replaced with 15 obstruction lines. DS2 and DS3 show how variously crossings are defined such that crossings control their own connectivity flows. DS4 has 1296 data points, and 6 obstacles. Note that 6 obstacles are composed of 92 line segments. DS4 illustrates the experiment of DBCluC's performance with respect to non-spherical shaped clusters and more complicated obstacles in the absence of noise.

Figures 6.2 and 6.3 illustrate the effectiveness of DBCluC in the presence of obstacles and crossings. For convenient comparison of clustering results, Figure 6.2 and 6.3 illustrate sequentially: (a) data points, obstacles, and crossings, before clustering; (b) clustering results in the absence of constraints; (c) clusters in the presence of crossings; (d) clusters in the presence of obstacles; and (e) clustering results in the presence of both types of constraints – obstacles and crossings. The red lines from obstacles in all datasets are the obstruction lines replacing the initial polygons that are illustrated by blue. Crossings from DS2, and DS3 are drawn in red lines and black lines implying entry edges and non-entry edges respectively. DS2 represents the primary intuitive solution of the problem we have investigated in this paper. The correct clustering shows 8 groups of data points. Although a cluster is close enough to access a crossing

from (c) in Figure 6.2, it is not merged with other clusters by the bridge owing to the modeling of the crossings.

DS3 in Figure 6.3(c) shows five clusters merged into two clusters by crossings, illustrating how crossings are differently modeled comparing to those of DS2. Depending upon where the entry edges are defined on a bridge, entry edges are in red in the figure. Clusters close to the longitudinal side of the bridge can also be pulled into the merger. In Figure 6.3(c), due to the bridge entry edge, one such cluster close to a crossing is merged, while the other is not. Moreover, as depicted in the figures, there is priority defined between an obstacle and a bridge. Even though an obstacle is drawn over a bridge, a bridge has a priority over an obstacle, unless otherwise specified.

Figure 6.4 shows an example of DBCluC's flexibility in handling complex cluster shapes and constraints. The obstacles are reduced to 40 obstruction lines. The number of obstruction lines that represent obstacle entities are dependent on the shape or complexity of the obstacle entities.

In this thesis, we simulate a real spatial data in order to experiment capability of DBCluC on a real data set. The real spatial data set is shown as an image format in Figure 6.5 (a). The map image describes the city centre of Edmonton, Canada. Note that the main roads, rivers, hills, and bridges are represented by sets of polygons for the experiment purpose of DBCluC. Some of the roads from the map are not modeled since they do not have significant effects on the clustering task of DBCluC. By ignoring all constraints, data objects are hard to be clustered such that they can be grouped together, as seen in Figure 6.5 (c). Bridge entities in Figure 6.6 (d) do not affect the clustering results since the density of data objects is too sparse to be grouped with respect to closeness between data objects. Figure 6.6 (e) shows the clustering output in the presence of obstacles, not considering crossing entities. One can observe that six clusters are merged into three clusters by three crossings which were not considered in Figure 6.6 (e). DBCluC finally generates a set of clusters such that the members in a group are comparable to each other taking into account the functionality of constraints: connectivity and disconnectivity.

6.2 Evaluations

We also conducted evaluations varying the size of the dataset and the number of obstacles in order to demonstrate the scalability of DBCluC. Figure 6.7 represents the execution time in seconds for eight datasets varying in size from 25K to 200K, with an increment of 25K data points, showing good scalability. The execution time is almost linear to the number of data objects.

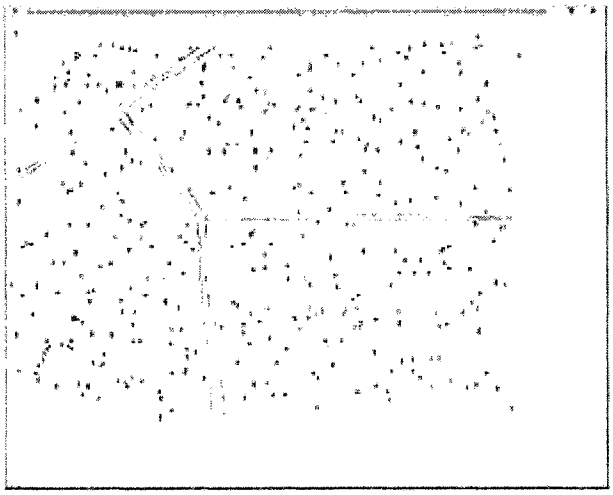
Table 6.1 shows run time varying the number of obstacles for clustering 38K data objects. The number of reduced line segments from polygons are almost half of the number of initial line segments from the polygons.

Number of line segments in obstacles	31	60	91	121	300	600	900
Number of obstruction lines	15	27	45	72	135	270	405
Time (sec)	58.89	65.03	73.68	86.13	116.53	180.42	242.11

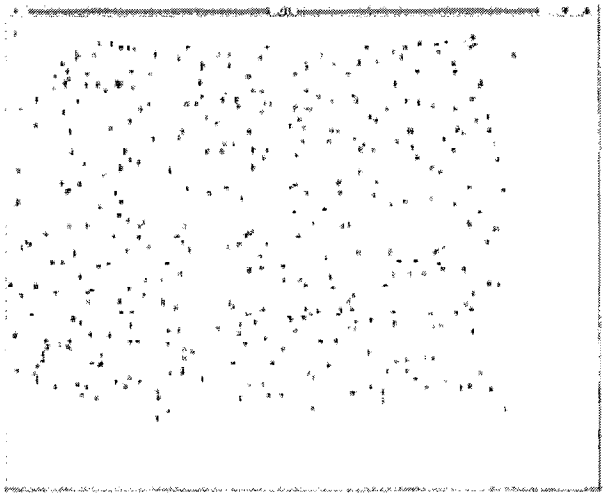
Number of line segments in obstacles	1200	1500	1800	2100	2400
Number of obstruction lines	540	675	810	945	1080
Time (sec)	306.08	368.33	431.01	497.53	558.89

Table 6.1: Run time varying the number of obstacles

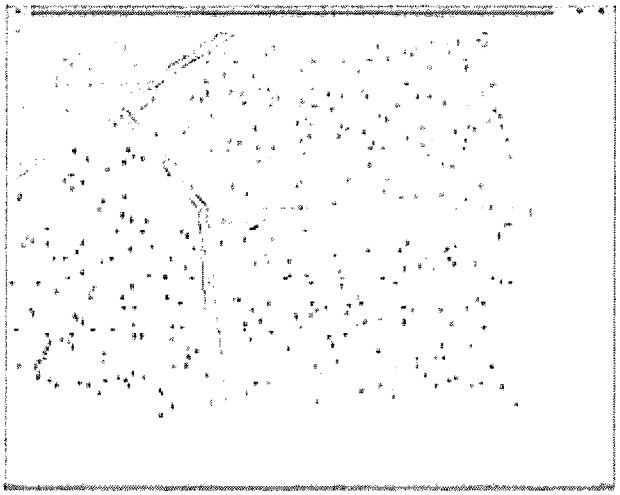
Figure 6.8 presents the execution time in seconds by varying the number of obstacles. The numbers in the X-axis represent the total number of polygon edges and the respective obstruction lines. Observe that the incremental ratio of X-axis to Y-axis in Table 6.1 is variable: the increment in Y-axis is constant, whereas the increment in X-axis is not. The increment ratio of the first three coordinates in X-axis becomes higher than that of the other coordinates in X-axis. Hence, the first three coordinates in X-axis are omitted in Figure 6.8 to show the scalability. However, in the proportion of the increment in the number of polygons, the execution time is almost linear. Thus, DBCluC is scalable for large databases with complicated obstacles and bridges in terms of size of the database and the number of constraints.



(a) Before clustering

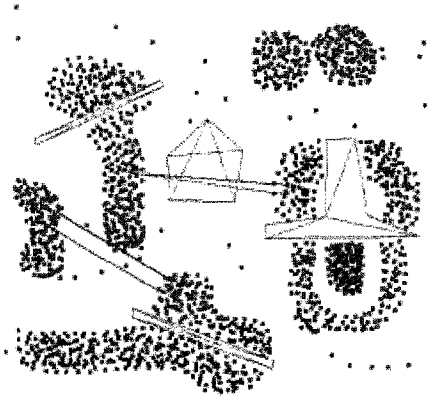


(b) Clustering without constraints

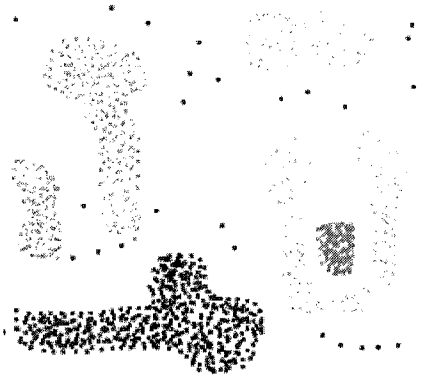


(c) Clustering with constraints

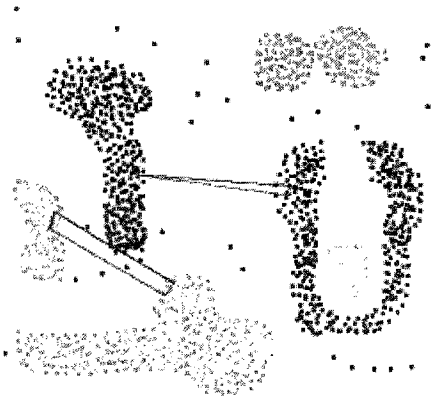
Figure 6.1: Clustering dataset DS1



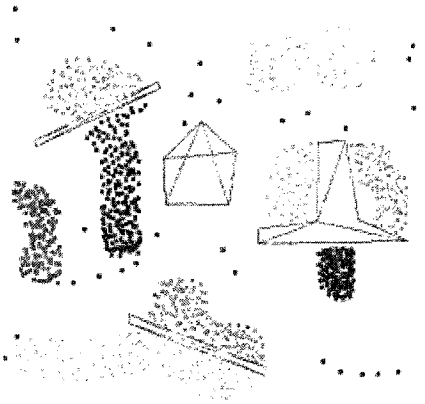
(a) Before clustering



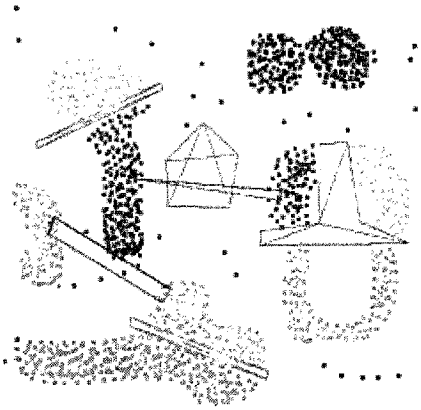
(b) Clustering without constrains



(c) Clustering with bridges

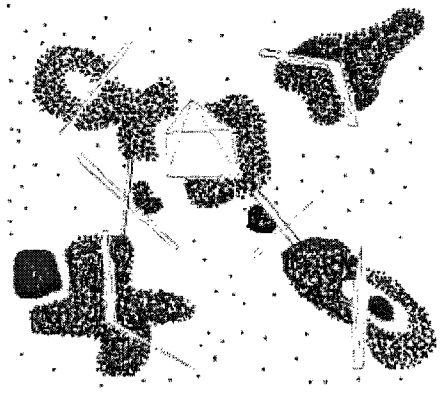


(d) Clustering with obstacles

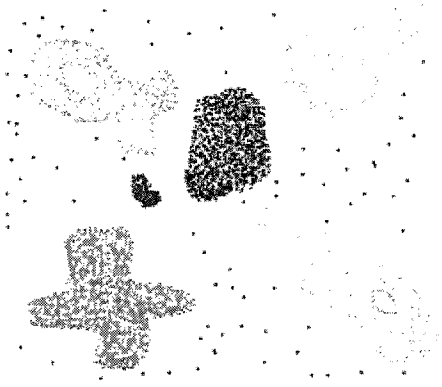


(e) Clustering with bridges and obstacles

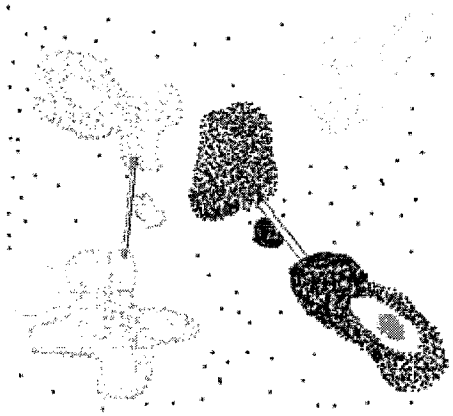
Figure 6.2: Clustering dataset DS2



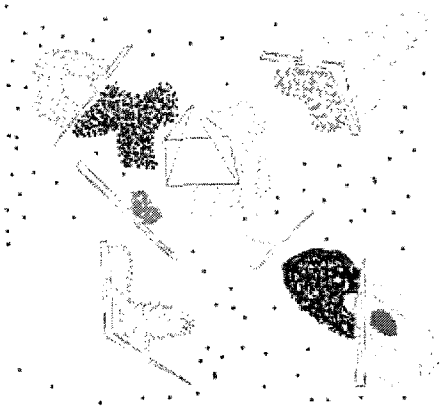
(a) Before clustering



(b) Clustering without constraints



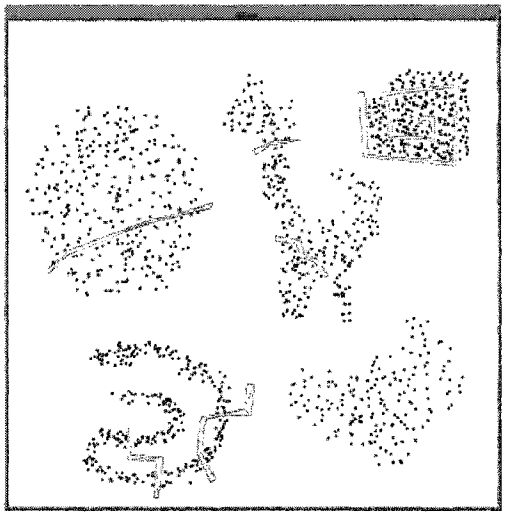
(c) Clustering with bridges



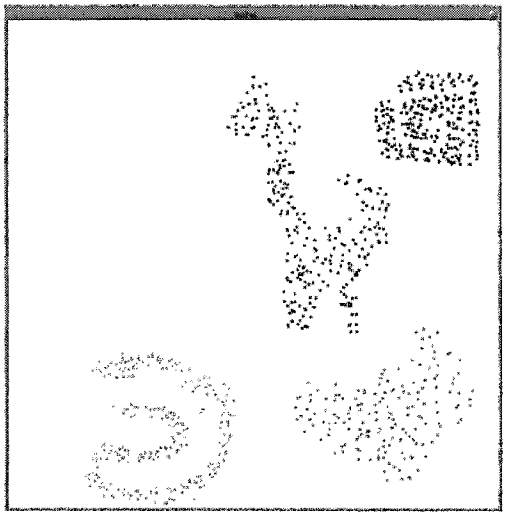
(d) Clustering with obstacles

(e) Clustering with bridges and obstacles

Figure 6.3: Clustering dataset DS3



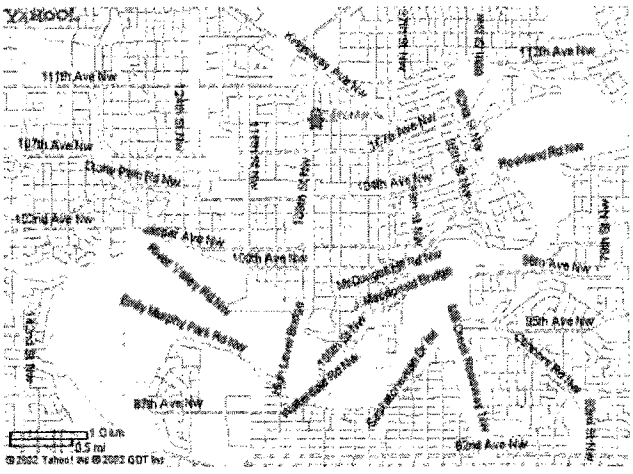
(a) Before clustering



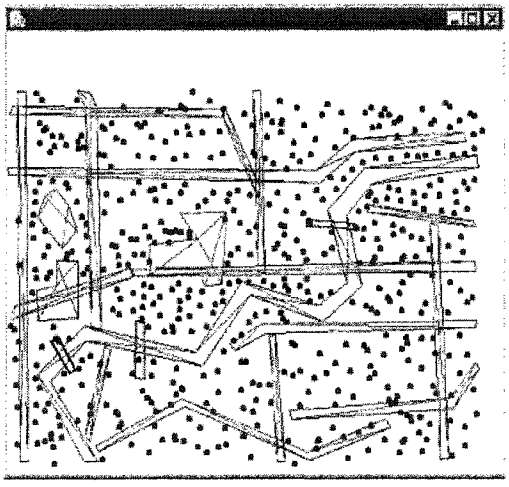
(b) Clustering without constraints

(c) Clusering with constraints

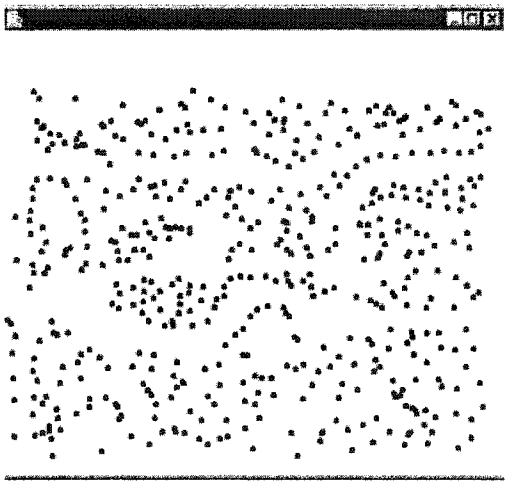
Figure 6.4: Clustering dataset DS4



(a) An example map of Edmonton

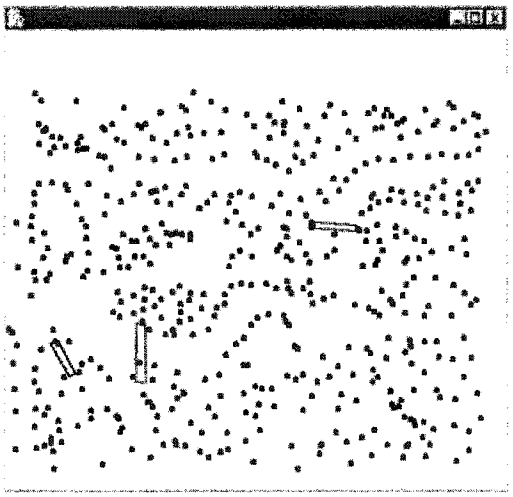


(b) Before clustering

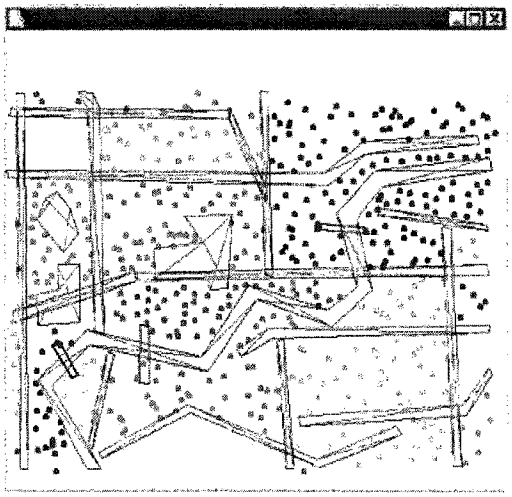


(c) Clustering without constraints

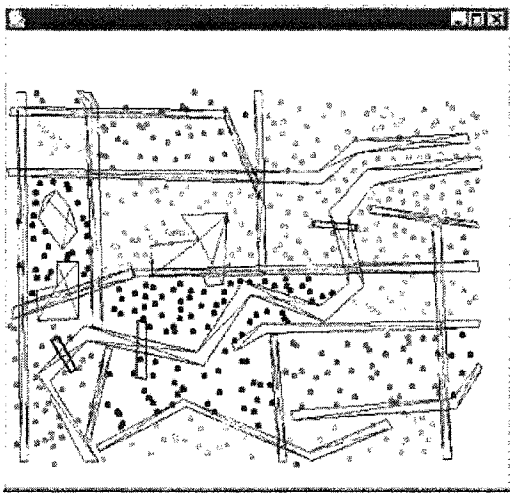
Figure 6.5: Clustering dataset DS5



(d) Clustering with bridges



(e) Clustering with obstacles



(f) Clustering with bridges and obstacles

Figure 6.6: Clustering dataset DS5(continued)

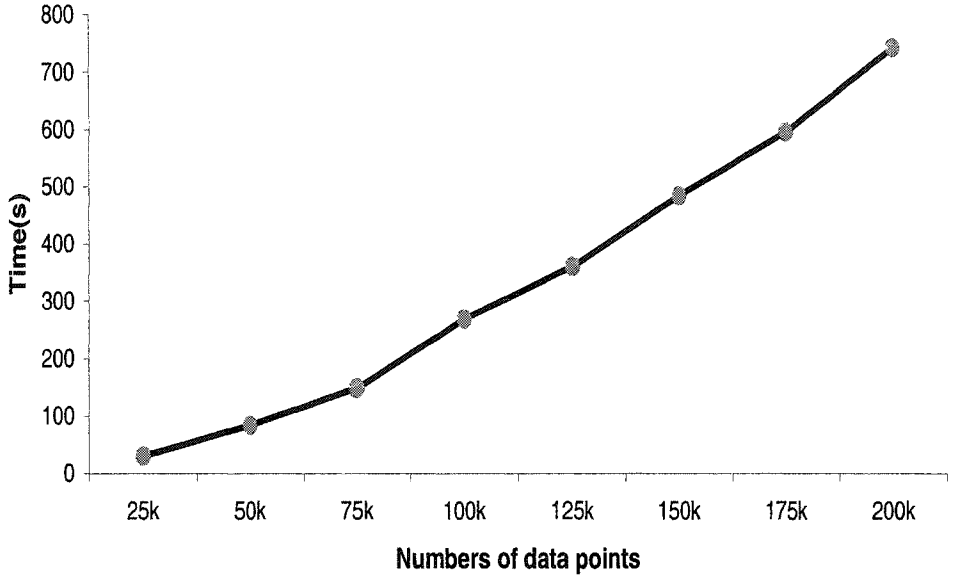


Figure 6.7: Algorithm Run Time by varying the number of data points

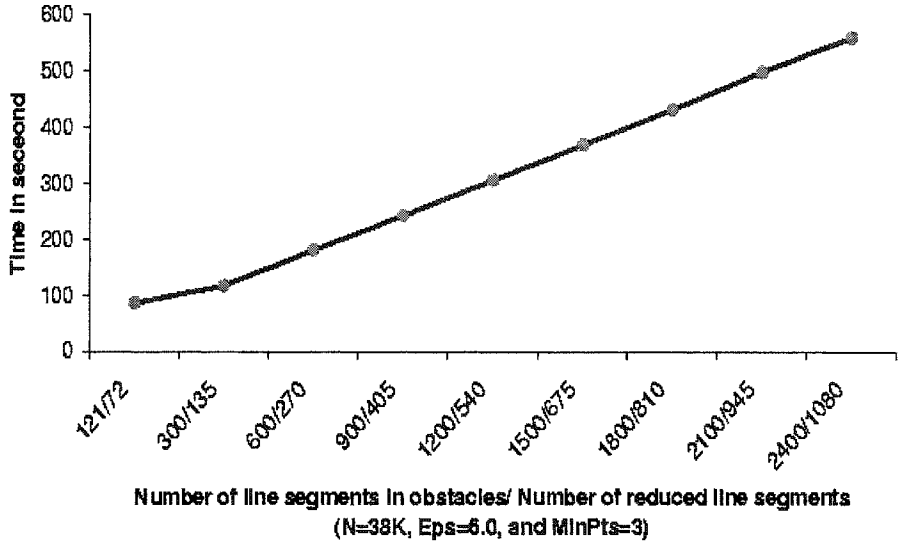


Figure 6.8: Algorithm Run Time by varying the number obstacles

Chapter 7

Conclusions and Future work

In this chapter, we summarize the works we have introduced thus far. In addition, some discussion about future work is presented.

7.1 Conclusions

In this thesis, we have addressed the problem of clustering spatial data in the presence of physical constraints in two dimensional planar space. The constraints we considered are not only obstacles such as rivers, highways, mountain ranges, etc., but also crossings such as bridges, pedestrian-overpasses, viaducts, etc. We have proposed model schemes for these constraints using polygons and have devised a method to prune the edges of polygons representing obstacles by identifying a minimum set of line segments, called obstruction lines, that does not compromise the visibility spaces. The polygon reduction algorithm reduces the number of lines representing a polygon by half and thus prunes the search space by half. We have also defined the concept of reachability in the context of obstacles and crossings and have used it in the designation of the clustering process. Finally, we have developed a density-based clustering algorithm, DBCluC, that takes constraints into account during the clustering process. Owing to the effectiveness of the density-based approach, DBCluC finds clusters of arbitrary shapes and sizes with a minimum of domain knowledge. In particular, it is not necessary to know the number of clusters to be discovered. In addition, experiments have shown the scalability of DBCluC in terms of the size of the database and the number and complexity of physical

constraints.

The following summarizes the major contributions of this thesis.

1. Defining a clustering problem in the presence of physical constraints: obstacles and crossings.
2. Modeling schemes of obstacles and crossings.
   - the Polygon Reduction algorithm using polygons.
   - Crossing modeling using Entry points and Entry edges.
3. Pruning searching spaces.
4. Controlling connectivity flow.

7.2 Future work

We have defined a clustering problem in the presence of physical constraints. The algorithm DBCluC in this thesis is useful to discover knowledge from large spatial databases in two dimensional space. Although DBCluC enhances efficiency and effectiveness in solving real clustering problems, there are still issues that have to be investigated in the future. This section briefly introduces key issues for future work.

7.2.1 Efficiency issues

In this thesis, obstacles are not indexed. This necessitates the checking of all obstruction lines before expanding the reachability of any point. While the number of line segments to test is reduced significantly thanks to the polygon reduction algorithm, this number can still be further reduced with a better indexing of the obstruction lines. Indeed, it suffices to test only the lines traversing the neighbourhood of a data point to be expanded. Data structures such as Quadtree, PM variants Quadtree, and R-tree are introduced to index line objects in the literature [39, 40, 19]. However, since lines represented by their end-points whose end-points are close to range queries can be relatively distant, it is difficult to index such lines for range queries. With a good

indexing scheme for the obstruction lines or polygons, the complexity of the clustering algorithm can be reduced to $O(N \log N)$. Moreover, most of the execution time in the current implementation is spent retrieving neighbours with range queries in the SR-tree structure, which indexes data objects. The SR-tree indexing scheme performs well for the k-NN type of queries instead. Note that k-NN search queries are not useful to expand clusters if the density of data distribution is high. For spatial databases, with an index structure optimized for range queries of two dimensional data objects, the run time of DBCluC could be dramatically improved.

7.2.2 Effectiveness issues

The main issue with respect to effectiveness in the clustering problems is the dimensionality of data objects. As we have shown in previous chapters, this thesis focuses on a two dimensional planar space. Although some real geographical spatial data objects are in a two dimensional space, it is critical for applications to support high-dimensional data objects such as the altitude of a geographical location. This in turn would enable the modeling schemes introduced in this thesis to investigate their multi-dimensional flexibility. We could then apply the schemes to wide range of applications.

The model we propose for crossings does not allow directional crossings and assumes that bridges are bi-directional. We need to investigate a new model for crossings with entry-points and exit-points forcing the expansion over a bridge to go from entry-point to exit-point, thus allowing more flexibility in the definitions of constraints. In addition, a new crossing model would also consider the length of bridges in the reachability definition. Doing so would restrict the maximal length of a bridge model to assign a semantic functionality since a relatively long crossing might lose its connectivity functionality. One could also add time constraints, which are very useful when analyzing traffic flow, for example. In applications studying flooding of fields or cities, the heights of obstacles are also important attributes to consider since at different heights, some obstacles may become irrelevant.

Operational constraints [46, 21], for example, user specified constraints or

a specific number of members for a cluster, which are not considered in this thesis, have a key role with respect to the effectiveness of clustering results, even though they require expensive processing.

As we have seen so far in this thesis, solving a clustering problem in the presence of constraints from large spatial databases is not trivial with respect to effectiveness and efficiency. This is a multi disciplinary problem requiring wide and deep investigation. We believe that further research in this area will solve constraints-based clustering problems much more effectively and efficiently, widening applicable domains.

Bibliography

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications, 1998.

[2] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: ordering points to identify the clustering structure. In *ACM-SIGMOD Int. Conf. Management of Data (SIGMOD' 99)*, pages 49–60, 1999.

[3] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proc. of the 1990 ACM SIGMOD Intl. Conf.*, pages 332–331, 1990.

[5] J.C. Bezdek and R.J. Hathaway. Numerical convergence and interpretation of the fuzzy c-shells clustering algorithm. *IEEE Transactions on Neural Networks*, 3(5):787–793, 1992.

[6] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces - index structures for improving the performance of multimedia databases. In *To be appear 2001 ACM Surveys*, 2001.

[7] P. S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining*, pages 9–15, 1998.

[8] V.L. Brailovsky. A probabilistic approach to clustering. *Pattern Recognition Letters*, 12(4):193–198, 1991.

[9] N.G. Colossi and M.A. Nascimento. Benchmarking access structures for high-dimensional multimedia data. In *Proc. IEEE Intl. Conf. on Multimedia and Expo (ICME'2000)*, pages 1215–1218, 2000.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[11] M. de Berg, M. van Kreveld, M. Vermars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer press, 1998.

[12] M. Ester, H. P. Kriegel, and X. W. Xu. Knowledge discovery in large spatial databases - focusing techniques for efficient class identification. *Lecture Notes In Computer Science*, 951:67–82, 1995.

[13] Martin Ester, Alexander Frommelt, Hans-Peter Kriegel, and Jorg Sander. Spatial data mining: Database primitives, algorithms and efficient dbms support.

[14] Martin Ester, Hans-Peter Kriegel, and Jorg Sander. Knowledge discovery in spatial databases. In *KI - Kunstliche Intelligenz*, pages 61–74, 1999.

[15] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[16] V. Estivill-Castro and I. Lee. Autoclust: Automatic clustering via boundary extraction for mining massive point-data sets. In *In Proceedings of the 5th International Conference on Geocomputation, 2000.*, 2000.

[17] Vladimir Estivill-Castro and IckJai Lee. Autoclust+: Automatic clustering of point-data sets in the presence of obstacles. In *International Workshop on Temporal and Spatial and Spatio-Temporal Data Mining (TSDM2000)*, pages 133–146, 2000.

[18] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *ACM-SIGMOD International Conference Management of Data*, pages 73–84, 1998.

[19] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[20] J. Han and M. KamberK. *Data Mining: Concepts and Techniques.* Morgan Kaufman, 2000.

[21] Jiawei Han, Laks V.S. Lakshmanan, and Raymond T. Ng. Constraint-based mulitdimensional data mining. *Computer*, 32(8):46–50, 1999.

[22] Paul S. Heckbert. *Graphics Gems(4).* Academic Press, 1994.

[23] A. Hinneburg and D. Keim. A general approach to clustering in large multimedia databases with noise, 1998.

[24] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Knowledge Discovery and Data Mining*, pages 58–65, 1998.

[25] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[26] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE*, 32(8):68–75, 1999.

[27] Norio Katayama and Shin'ichi Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In *Proc. of the 1997 ACM SIGMOD Intl. Conf.*, pages 369–380, 1997.

[28] L. Kaufman and P.Rousseeuw. Finding groups in data: an introduction to cluster analysis. In *Finding Groups in Data: An Introduction to Cluster Analysis.* Wiley, New York, 1990.

[29] J. Mark Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14(4):799–817, 1985.

[30] B. King. Step-wise clustering procedures, 1967.

[31] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[32] K. Koperski, J. Adhikary, and J. Han. Spatial data mining: Progress and challenges survey paper, 1996.

[33] J. MacQueen. Some methods of classification and analysis of multivariate observations, 1967.

[34] S. Nahar and S. Sahni. Fast algorithm for polygon decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7:473–483, 1988.

[35] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. of VLDB Conf.*, pages 144–155, 1994.

[36] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, Berlin, Germany, 1985.

[37] H. Ralambondrainy. A conceptual version of the k-means algorithm. *Pattern Recognition Letters*, 16(11):1147–1157, 1995.

[38] Enrique H. Ruspini. A new approach to clustering. *Information and Control*, 15(1):22–32, 1969.

[39] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[40] Hanan Samet and Robert E. Webber. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics*, 4(3):182–222, 1985.

[41] J. W. Shavlik and T. G. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann, 1990.

[42] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 428–439, 24–27 1998.

[43] P. Sneath and R. Sokal. Numerical taxonomy, 1973.

[44] M.G. Stone. A mnemonic for areas of polygons. *AMER. MATH. MONTHLY*, 93:479–480, 1986.

[45] A. K. H. Tung, J. Hou, and J. Han. Spatial clustering in the presence of obstacles. In *Proc. 2001 Int. Conf. On Data Engineering(ICDE'01)*, 2001.

[46] Anthony K. H. Tung, Raymond T. Ng, Laks V. S. Lakshmanan, and Jiawei Han. Constraint-based clustering in large databases. In *ICDT*, pages 405–419, 2001.

[47] Robert J. Walker and Jack Snoeyink. Practical point-in-polygon tests using CSG representations of polygons, 12, 1999.

[48] Wei Wang, Jiong Yang, and Richard R. Muntz. STING: A statistical information grid approach to spatial data mining. In *The VLDB Journal*, pages 186–195, 1997.

[49] C. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. In *IEEE Transactions on Computers*, pages 20:68–86, 1971.

[50] Osmar R. Zaïane, Andrew Foss, Chi-Hoon Lee, and Weinan Wang. On data clustering analysis: Scalability, constraints and validation. *Lecture Notes in AI 2336*, pages 28–39, May 2002.

[51] Osmar R. Zaïane and Chi-Hoon Lee. Clustering spatial data in the presence of obstacles. In *Sixth International Database Engineering and Applications Symposium*, July 2002.

[52] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *ACM-SIGMOD International Conference Management of Data*, pages 103–114, June 1996.