Spatial Clustering Methods in Data Mining: A Survey*

Jiawei Han, Micheline Kamber and Anthony K. H. Tung
School of Computing Science, Simon Fraser University
Burnaby, BC Canada V5A 1S6
E-mail: {han, kamber, khtung}@cs.sfu.ca

Abstract

Spatial clustering, which groups similar spatial objects into classes, is an important component of spatial data mining (Han & Kamber, 2000). Spatial clustering can be used in the identification of areas of similar land usage in an earth observation database or in merging regions with similar weather patterns, etc. As a data mining function, spatial clustering can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. It may also serve as a preprocessing step for other algorithms, such as classification and characterization, which will operate on the detected clusters.

Due to its immense applications in various areas, spatial clustering has been a highly active topic in data mining research, with fruitful, scalable clustering methods developed recently. These spatial clustering methods can be classified into four categories: partitioning method, hierarchical method, density-based method and grid-based method. In this paper, we will introduce each of these categories and present some representative algorithms from them. In addition, we will also discuss some open problems and future research in the field of clustering.

1 Introduction

Spatial clustering is the process of grouping a set of objects into classes or *clusters* so that objects within a cluster have high similarity in comparison to one another, but are dissimilar to objects in other clusters. As a branch of statistics, cluster analysis has been studied extensively for many years, focusing mainly on distance-based cluster analysis. Cluster analysis tools based on k-means, k-medoids, and several other methods have also been built into many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS. Clustering has also been studied in the field of machine learning as a type of *unsupervised learning* because it does not rely on predefined class and class-labeled training examples. However, efforts to perform effective and efficient clustering on large databases only started in recent years with the emergence of *data mining*.

* Research was supported in part by research grants from the Natural Sciences and Engineering Research Council of Canada, and grants NCE:IRIS3 and NCE:GEOID from the Networks of Centres of Excellence of Canada.

Data mining or *knowledge discovery in database (KDD)* refers to the non-trivial process of *discovering interesting, implicit, and previously unknown knowledge from large databases* (Fayyad *et al.*, 1996). For the geographical community, data mining promises many new and useful tools for the analysis of geographical data. Spatial clustering is one of these tools. Examples of the application of spatial clustering include the identification of areas of similar land usage in an earth observation database and the merging of regions with similar weather patterns. Despite being a sub-branch of data mining, the immense applications of spatial clustering has resulted in tremendous growth of the field, making it worth of a dedicated, comprehensive overview.

In order to choose a clustering algorithm that is suitable for a particular application, many factors have to be considered. These include:

1. **Application goal**
   The goal of an application will often affect the type of clustering algorithms being used. For example, when trying to discover some good locations for setting up their stores, a supermarket chain may like to cluster their customers such that the sum of the distance to the cluster center is minimized. For such applications where the distance to the cluster center is desired to be short, partitioning algorithms like k-means and k-medoids are often used. On the other hand, in applications like raster data analysis and image recognition, it is often desirable to find natural clusters, i.e., clusters which are perceived as crowded together by human eye. In such a case, clusters discovered should have certain uniformity in density, colors, etc. and can be of arbitrary shape and size. Since algorithms like k-means and k-medoids tend to discover clusters with spherical shape and similar size, density-based algorithms will be more suitable for these applications.

2. **Tradeoff between quality and speed**
   As a rule of thumb, there will usually be tradeoffs between the speed of a clustering algorithm and the quality of the clusters it produces. A suitable clustering algorithm for an application must satisfy both the quality and speed requirements. Often, the size of the data being clustered plays an important factor in the running time of a clustering algorithm. A clustering algorithm that produces good quality clusters may however be unable to handle large datasets, making it suitable only for applications with a small database. To handle large databases, a common approach is to perform some form of compression on the initial databases and then cluster the compressed data. As the compression is usually lossy, the quality of the resulting clusters will often drop. The challenge in this approach is to compress the data in such a way that the speed of clustering increases substantially with minimal drop in cluster quality.

3. **Characteristics of the data**
   The characteristics of the data being clustered serve as another important factor in determining the clustering algorithm being applied. These characteristics include:

   - *The types of data attributes*
     The similarity between two data objects is judged by the difference in their data attributes. When all these attributes are numeric, distance measures

like Euclidean distance and Manhattan distance can be easily computed. However, when binary, categorical and ordinal attributes are involved, the computation of distance measure is much more complicated. Presently, most clustering algorithms are based on numeric attributes.

- *Dimensionality*
  The dimensionality of the data refers to the number of attributes in a data object. Many clustering algorithms which perform well on low-dimensional data degenerate when the number of dimensions increases. The degeneration can come in two forms: increase in running time or decrease in cluster quality. In order to choose a clustering algorithm for high-dimensional data clustering, the running time and cluster quality produced by the algorithm at high dimension must first be assessed to meet the requirement of the application.
- *Amount of noise in data*
  As some of the clustering algorithms are very sensitive to noise and outliers, careful choice must be made if the data in the application contains a large amount of noise.

In our discussion, we will focus on clustering algorithms which work reasonably well for large geographical databases. These algorithms mostly work on numerical attributes and can be separated into four general categories: *partitioning method*, *hierarchical method*, *density-based method* and *grid-based method*. In this paper, we will first look at the partitioning method in Section 2 following by hierarchical method in Section 3. In Section 4, we will introduce the density-based clustering algorithms. The grid-based method will be discussed in Section 5. In Section 6, we will look at some recent work that can handle real life constraints when performing clustering. We conclude our survey with Section 7.

2 Partitioning methods

Partitioning algorithms had long been popular clustering algorithms before the emergence of data mining. Given a set D of n objects in a d-dimensional space and an input parameter k, a partitioning algorithm organizes the objects into k clusters such that the total deviation of each object from its cluster center or from a *cluster distribution* is minimized. The deviation of a point can be computed differently in different algorithms and is more commonly called a *similarity function*.

In this section, we will introduce three partitioning algorithms: **k-means** algorithm (MacQueen, 1967), **EM (Expectation Maximization)** algorithm (Dempster *et al.*, 1977; Yu *et al.*, 1998; Bradley *et al.*, 1998) and **k-medoid** algorithm (Kaufman & Rousseeuw, 1990). The three algorithms have different ways of representing their clusters. The k-means algorithm uses the centroid (or the mean) of the objects in the cluster as the cluster center while k-medoids algorithm uses the most centrally located object in the cluster. Unlike k-means and k-medoids, EM clustering uses a distribution consisting of a mean and a $d \times d$ covariance matrix to represent each cluster. Instead of assigning each object to a dedicated cluster, EM clustering assigns each object to a cluster according to a probability of membership which is computed

Algorithm 2.1 (Iterative Relocation) The generalized iterative relocation algorithm.

Input: The number of clusters k, and a database containing n objects.

Output: A set of k clusters which minimizes a criterion function E.

Method:
1) arbitrarily choose k centers/distributions as the initial solution;
2) repeat
3) (re)compute membership the objects according to present solution.
4) update some/all cluster centers/distributions according to new memberships of the objects;
5) until (no change to E);

□

Figure 1: Generalized iterative relocation.

from the distribution of each cluster. In this way, each object has a certain probability of belonging to each cluster, making EM clustering a fuzzy partitioning technique.

Despite the difference in the representation of the clusters, the three partitioning algorithms share the same general approach when computing their solutions. To see this similarity, we first observe that the three algorithms are effectively trying to find the k centers or distributions that will optimize an objective criterion. Once the optimal k centers or distributions are found, the membership of the n objects within the k clusters are automatically determined. However, to find the global optimal k centers or k distributions is known to be NP-hard (Garey & Johnson, 1979). Instead the three algorithms adopt an *iterative relocation technique* which will find a local optimal. This technique is shown in Figure 1. The three algorithms, however, differ in the criterion function and in the way they handle Steps 3 and 4 of Algorithm 2.1. The general weaknesses of partitioning-based algorithms include a requirement to specify the parameter k and their inability to find arbitrarily shaped clusters. We will look at the three algorithms in more details.

2.1 The k-means method

The k-means algorithm (MacQueen, 1967) uses the *mean* value of the objects in a cluster as the cluster center. The objective criterion used in the algorithm is typically the squared-error function defined as

$$E = \Sigma_{i=1}^{k} \Sigma_{x \in C_i} |x - m_i|^2, \tag{2.1}$$

where x is the point in space representing the given object, and m_i is the mean of cluster C_i.

The k-means algorithm basically follows the structure of Algorithm 2.1. In Step 3 of the algorithm, k-means assigns each object to its nearest center forming a new set of clusters. In Step 4, all the centers of these new clusters are then computed by taking the mean of all the objects in each cluster. This is repeated until the criterion function E does not change after an iteration.

The k-means algorithm is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, $k \ll n$ and $t \ll n$. The method often terminates at a local optimum.

Besides the general weakness of partitioning-based algorithm, the k-means algorithm is also very sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value.

2.2 The EM (Expectation Maximization) algorithm

Instead of representing each cluster using a single point, the EM (Expectation Maximization) algorithm represents each cluster using a probability distribution. Typically, the Gaussian probability distribution is used because according to density estimation theory, any density distribution can be effectively approximated by a mixture of Gaussian (Scott, 1992; Silverman, 1986). A d-dimensional Gaussian distribution representing a cluster C_i is parameterized by the mean of the cluster μ_i and a $d \times d$ covariance matrix M_i. Given a cluster distribution for C_i, the probability of an object occurring at location x is denoted as $P(x|i)$ where:

$$P(x|i) = \frac{1}{\sqrt{(2\pi)^d |M_i|}} e^{\frac{1}{2}(x-\mu_i)^T (M_i)^{-1}(x-\mu_i)} \tag{2.2}$$

where the superscript ^{T} indicates transpose to a row vector, $|M_i|$ is the determinant of M_i and M_i^{-1} is its matrix inverse. By combining the effect of the different cluster distributions at x, the mixture model probability density function will be:

$$P(x) = \sum_{i=1}^{k} W_i P(x|i) \tag{2.3}$$

where W_i is the fraction of the database represented by C_i. Unlike the other two partitioning algorithms, an object in EM clustering can be a member of each of the k clusters with different probabilities of membership. Probability of an object at x belonging to cluster C_i can be computed as:

$$P(i|x) = W_i \frac{P(x|i)}{P(x)} \tag{2.4}$$

Referring back to Algorithm 2.1, after the random initialization of the model, the EM algorithm will compute the membership of each object in Step 3 by applying formulae 2.2, 2.3 and 2.4. The new values of W_i, μ_i and M_i are then computed in Step 4 using the following formulae:

$$W_i = \frac{1}{n} \sum_{x \in D} P(i|x) \tag{2.5}$$

$$\mu_i = \frac{\sum_{x \in D} x.P(i|x)}{\sum_{x \in D} P(i|x)} \tag{2.6}$$

$$M_i = \frac{\sum_{x \in D} P(i|x)(x - \mu_i)(x - \mu_i)^T}{\sum_{x \in D} P(i|x)} \tag{2.7}$$

As a criterion function, EM clustering tries to maximize the log likelihood of the mixture model computed as

$$E = \sum_{x \in D} log(P(x)) \tag{2.8}$$

When the increase in the log likelihood between two successive iterations is negligible, the algorithm terminates. Like the other two algorithms, EM clustering converge to a local optimal.

2.3 The k-medoids method

Unlike the k-means and the EM algorithm, the k-medoids method uses the most centrally located object (*medoids*)in a cluster to be the cluster center instead of taking the mean value of the objects in a cluster. Because of this, the k-medoids method is less sensitive to noise and outliers. This, however, results in a higher running time.

The initializing state of the k-medoids algorithm follows that of Algorithm 2.1 in which k objects are randomly selected to be the cluster centers. Like the k-means algorithm, k-medoids clustering assigns an object to its nearest center. However, this is performed as part of the process in Step 4 of the iterative relocation algorithm making Step 3 redundant. The k-medoids algorithm differs from the other two algorithms in that at most one center will be changed in Step 4 for each iteration. This change in center must result in a decrease in the criterion function which is typically the squared error function.

To accomplish Step 4, an earlier k-medoids algorithm like **PAM** (Partitioning Around Medoids) (Kaufman & Rousseeuw, 1990) iterates through all the k cluster centers and tries to replace each of them with one of the other $(n - k)$ objects. For every such replacement, if the squared error function E decreases, the replacement will take place, causing the next iteration of Algorithm 2.1 to occur. However, if no such replacement is found after going through all the k clusters, there will be no change to E, and the algorithm terminates with a local optimal. Since PAM attempts to replace each of the k cluster centers with one of the $(n - k)$ objects and each of these attempts results in $(n - k)$ operations for calculating E, the total complexity of PAM in one iteration is $O(k(n - k)^2)$. For large values of n, such computation becomes costly.

Because of its complexity, a partitioning algorithm like PAM works effectively for small data sets, but does not scale well for large data sets. To deal with larger data sets, a *sampling*-based method, called **CLARA** (Clustering LARge Applications) was developed by Kaufman and Rousseeuw in (Kaufman & Rousseeuw, 1990). Instead

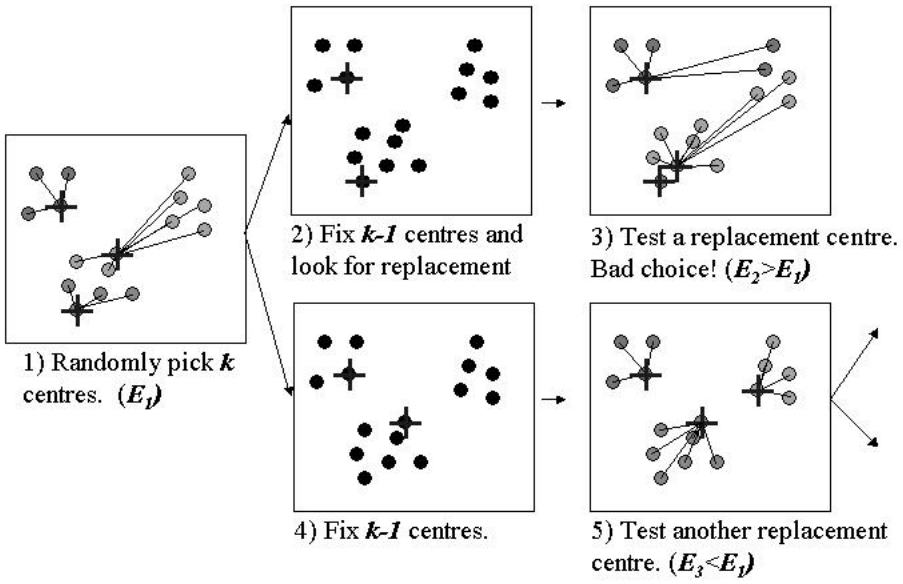


Figure 2: CLARANS searching for better solution.

of taking the whole set of data into consideration, a small portion of the actual data is chosen as a representative of the data. Medoids are chosen from this sample using PAM and the average dissimilarity is computed using the whole dataset. If a new set of medoids gives lower dissimilarity than a previous best solution, then the best solution is replaced with the new set of medoids. For better results, CLARA draws multiple samples of the data set, applies PAM on each sample, and returns its best clustering as the output. As expected, CLARA can deal with larger data sets than PAM. The complexity of each iteration now becomes $O(ks^2+k(n-k))$, where s is the size of the sample, k is the number of clusters, and n is the total number of objects.

The effectiveness of CLARA depends on the sample size. Notice that PAM searches for the best k medoids among a given data set, whereas CLARA searches for the best k medoids among the *selected* sample of the data set. CLARA cannot find the best clustering if any sampled medoid is not among the best k medoids. A good clustering based on samples will not necessarily represent a good clustering of the whole data set if the sample is biased. To improve the quality and scalability of CLARA, another clustering algorithm called **CLARANS** (Clustering Large Applications based upon RANdomized Search) was proposed in (Ng & Han, 1994). When searching for a better center in Step 4 of the iterative allocation algorithm, CLARANS tries to find a better solution by randomly picking one of the k centers and trying to replace it with another randomly chosen object from the other $(n-k)$ objects. If no better solution is found after a certain number of attempts, the local optimal is

assumed to be reached. In Figure 2, we show CLARANS fixing $k-1$ centers in Step 2 and testing out a new randomly selected center at Step 3. Having found that the new solution is not better than the original solution, CLARANS backtracks to Step 1 and repeats the process from there. Steps 4 and 5 show CLARANS being successful in searching for a better solution and it proceeds with the search from Step 5.

CLARANS has been experimentally shown to be more effective than both PAM and CLARA. However, its computational complexity is still about $O(n^2)$, where n is the number of objects. Furthermore, its clustering quality is dependent on the sampling method used. The performance of CLARANS can be further improved by exploring spatial data structures, such as R*-trees, and some focusing techniques (Ester *et al.*, 1995).

3 Hierarchical methods

A hierarchical method creates a hierarchical decomposition of the given set of data objects forming a *dendrogram* — a tree which splits the database recursively into smaller subsets. The dendrogram can be formed in two ways: "*bottom-up*" or "*top-down*". The "*bottom-up*" approach, also called the "*agglomerative*" approach, starts with each object forming a separate group. It successively merges the objects or groups according to some measures like the distance between the two centers of two groups and this is done until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds. The *top-down*, also called the "*divisive*" approach, starts with all the objects in the same cluster. In each successive iteration, a cluster is split into smaller clusters according to some measures until eventually each object is in one cluster, or until a termination condition holds.

Earlier hierarchical clustering methods like **AGNES** and **DIANA** (Kaufman & Rousseeuw, 1990) suffer from the use of over-simplified measures to split or merge the clusters. Coupled with the fact that steps (merge or split) taken are irreversible, such methods often result in erroneous clusters being found.

To enhance the effectiveness of hierarchical clustering, recent methods have adopted one of the following two approaches. The first approach represented by algorithms like **CURE** (Guha *et al.*, 1998) and **CHAMELEON** (Karypis *et al.*, 1999) utilizes a more complex principle when splitting or merging the clusters. Although the splitting and merging of clusters are still irreversible in this approach, less errors are made because a better method is used for merging or splitting. The second approach represented by algorithms like **BIRCH** (Zhang *et al.*, 1996) is to obtain an initial result by using a hierarchical agglomerative algorithm and then refining the result using iterative relocation.

In this section, we will have a look at these hierarchical clustering methods in details.

3.1 AGNES and DIANA

AGNES and DIANA are two earlier hierarchical clustering algorithms. AGNES (AGglomerative NESting) is a bottom-up algorithm which starts by placing each object in its own cluster and then merging these atomic clusters into larger and larger clusters,

until all of the objects are in a single cluster or until a certain termination condition is satisfied. DIANA (DIvisia ANAlysis), on the other hand, adopted a top-down strategy that does the reverse of AGNES by starting with all objects in one cluster. It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the distance between the two closest clusters is above a certain threshold distance.

In either AGNES or DIANA, one can specify the desired number of clusters as a termination condition. The widely used measures for distance between clusters are as follows, where m_i is the mean for cluster C_i, n_i is the number of objects in C_i, and $|p-p'|$ is the distance between two objects or points p and p'.

$$\begin{aligned} d_{min}(C_i, C_j) &= min_{p \in C_i, p' \in C_j} |p - p'| \\ d_{mean}(C_i, C_j) &= |m_i - m_j| \\ d_{avg}(C_i, C_j) &= 1/(n_i n_j) \Sigma_{p \in C_i} \Sigma_{p' \in C_j} |p - p'| \\ d_{max}(C_i, C_j) &= max_{p \in Ci, p' \in C_j} |p - p'| \end{aligned}$$

The two algorithms, AGNES and DIANA, though simple, often encounter difficulties regarding the selection of merge or split points. Such a decision is critical because once a group of objects are merged or split, the process at the next step will operate on the newly generated clusters. It will neither undo what was done previously, nor perform object swapping between clusters. Thus merge or split decisions, if not well chosen at some step, may lead to low quality clusters. Moreover, the method does not scale well since the decision of merge or split needs to examine and evaluate a good many number of objects or clusters.

3.2 BIRCH: Balanced Iterative Reducing and Clustering using Hierarchies

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is an integrated hierarchical clustering method. The main concept of BIRCH is to compress the data objects into many small subclusters and then perform clustering with these subclusters. Due to the compression, the number of subclusters is much less than the number of data objects and thus it allows the clustering to be performed in the main memory. This results in an algorithm which only needs to scan the database once.

In BIRCH, each small subcluster is represented by a *clustering feature* which is a triplet summarizing information about the group of data objects it represents. Given N d-dimensional points or objects $\{X_i\}$ in a cluster, the CF of a subcluster is defined as

$$CF = (N, \vec{LS}, SS), \tag{3.9}$$

where N is the number of points in the subcluster, $\vec{LS}$ is the linear sum on N points, i.e., $\sum_{i=1}^{N} \vec{X}_i$, and SS is the square sum of data points, i.e., $\sum_{i=1}^{N} \vec{X}_i^{\,2}$.

These CFs are stored in a *clustering feature tree* (**CF tree**), which are used to summarize cluster representations. A *CF-tree* is a height-balanced tree which stores

the CFs for a hierarchical clustering. The non-leaf nodes store sums of the CFs of their children, and thus, summarize clustering information about their children. A CF tree has two parameters: *branching factor*, B, and *threshold*, T. The branching factor specifies the maximum number of children per non-leaf node. The threshold parameter specifies the maximum diameter of subclusters stored at the leaf nodes of the tree. These two parameters influence the size of the resulting tree.

The CF tree is built dynamically as objects are inserted. An object is inserted to the closest leaf entry (subcluster). If the diameter of the subcluster stored in the leaf node after insertion is larger than the threshold value, then the leaf node and possibly other nodes are split. After the insertion of the new object, information about it is passed towards the root of the tree. The size of the CF tree can be changed by modifying the threshold. If a threhold value results in a CF-tree that could not fit into the main memory, the threhold is increased and the CF-tree is rebuilt starting from the root of the tree. Thus, the process of rebuilding the tree is done without the necessity of re-reading all of the objects or points. As such the database is only scan once when building the CF tree. After the CF tree is built, any clustering algorithm, such as a typical partitioning algorithm is then used to perform the clustering in main memory. To improve the clustering quality further, one or more additional scans can (optionally) be performed . The computation complexity of the algorithm is $O(N)$, where N is the number of objects to be clustered.

Experiments have shown the linear scalability of the algorithm with respect to the number of objects, and good quality of clustering of the data. However, since each node in a CF-tree can hold only a limited number of entries due to its size, a CF-tree node does not always correspond to what a user may consider a natural cluster. Moreover, if the clusters are not spherical in shape, BIRCH does not perform well because it uses the notion of radius or diameter to control the boundary of a cluster.

3.3 CURE: Clustering Using REpresentatives

Unlike traditional agglomerative methods like AGNES, CURE (Clustering Using REpresentatives) is an agglomerative method which uses a more sophisticated principle when merging clusters.

There are two main ideas which CURE employs to obtain high quality clusters. First, instead of using a single centroid or object to represent a cluster, a fixed number of well-scattered objects are selected to represent each cluster. Second, the selected representative objects are shrunk towards their cluster centers by a specified fraction *shrinking factor* α which ranges between 0 and 1.

At each step of the algorithm, the two clusters with the closest pair of representative points (where each point in the pair is from a different cluster) are merged. Having more than one representative point per cluster is an approach that adopts the middle ground between the use of all points or only the centroid to represent a cluster. This approach allows CURE to adjust well to the geometry of non-spherical shapes. The shrinking or condensing of clusters helps dampen the effects of outliers. Therefore, CURE is more robust to outliers and identifies clusters having non-spherical shapes and wide variance in size. It scales well for large databases without sacrificing clustering quality. Figure 3 shows the merging of two subclusters in CURE and the

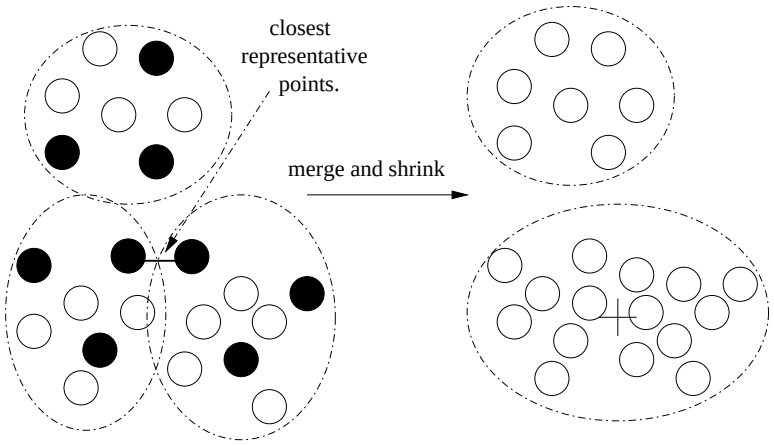


Figure 3: Merging two clusters in CURE.

subsequent shrinking of the data points towards the center of the merged cluster. The dark points in the figure are the representative points selected for each subcluster.

To handle large databases, CURE employs a combination of random sampling and partitioning: a random sample is first partitioned, and each partition is partially clustered. The partial clusters are then clustered in a second pass to yield the desired clusters. A sensitivity analysis shows that although some parameters can be varied without impacting the quality of clustering, the parameter setting in general does have a significant influence on the results.

3.4 CHAMELEON

Similar to CURE, CHAMELEON is a clustering algorithm which tries to improve the clustering quality by using a more elaborate criteria when merging two clusters. Two clusters will be merged if the inter-connectivity and closeness (proximity) of the merged cluster is very similar to the inter-connectivity and closeness of the two individual clusters before merging.

To form the initial subclusters, CHAMELEON first creates a graph $G = (V, E)$ where each node $v \in V$ represents a data object and a weighted edge (v_i, v_j) exists between two node v_i and v_j if v_j is one of the k-nearest neighbors of v_i. The weight of each edge in G represents the closeness between the two data objects it connects, i.e., an edge will weight more if the two data objects are close to each other. CHAMELEON then uses a graph partitioning algorithm to recursively partition G into many small unconnected subgraph by doing a *min-cut* on G at each level of the recursion. Here a *min-cut* on a graph G refers to a partitioning of G into roughtly two parts of equal size such that the total weight of the edges being cut is minimized. Each subgraph is then treated as an initial subcluster and an agglomerative hierarchical clustering algorithm will then repeatedly combine two most similar clusters until a certain criterion is reached. We illustrate this process in Figure 4.

CHAMELEON determines the similarity between each pair of clusters C_i and C_j according to their *relative inter-connectivity* $RI(C_i, C_j)$ and their *relative closeness*

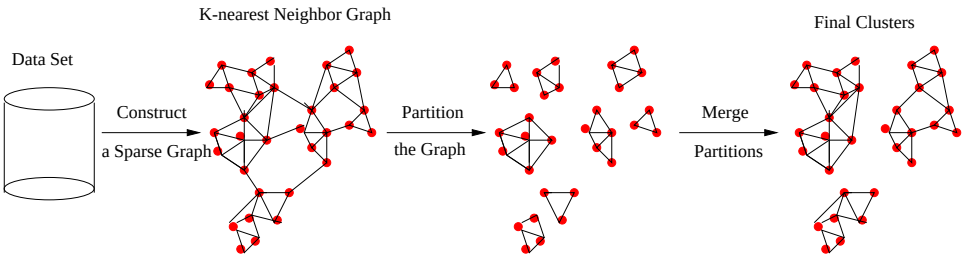


Figure 4: An overview of CHAMELEON.

$RC(C_i, C_j)$. Given that the *inter-connectivity* of a cluster is defined as the total weight of edges that are removed when a min-cut is performed on the cluster, we define the *relative inter-connectivity* $RI(C_i, C_j)$ between two clusters C_i and C_j as the ratio between the inter-connectivity of the merged cluster of C_i and C_j to the average inter-connectivity of C_i and C_i. Similarly, the *relative closeness* $RC(C_i, C_j)$ between two clusters C_i and C_j is defined as the ratio between the closeness of the merged cluster of C_i and C_j to the average internal closeness of C_i and C_j. Here the closeness of a cluster refers to the average weight of the edges that are removed when a min-cut is performed on the cluster. To specify the importance of the relative inter-connectivity and relative closeness in computing the similarity function, user has to specify a value α between 0 to 1. The similarity function is then computed as $RC(C_i, C_j) * RF(C_i, C_j)^{\alpha}$. A value of 1 for α will give equal weight to both the measure while decreasing α will give more emphasis on $RF(C_i, C_j)$.

Because CHAMELEON takes into account both the inter-connectivity as well as the closeness of the clusters, it does not depend on a static, user-supplied model (e.g. a log-linear model), and can automatically adapt to the internal characteristics of the clusters being merged. It has been shown that CHAMELEON is more effective than CURE in discovering arbitrarily-shaped clusters of varying density. However, the processing cost for high-dimensional data may require $O(n^2)$ time for n objects in the worst case.

4 Density-based methods

Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes. Other clustering methods have been developed based on the notion of *density*. These typically regard clusters as dense regions of objects in the data space which are separated by regions of low density (representing noise). Density-based methods can be used to filter out noise (outliers), and discover clusters of arbitrary shape.

The first density-based algorithm is the **DBSCAN** algorithm (Ester *et al.*, 1996) which judges the density around the neighborhood of an object to be sufficiently dense if the number of data points within a distance ϵ of an object is greater than $MinPts$ number of points. As the clusters discovered are dependent on the parameter ϵ and $MinPts$, DBSCAN relies on the user's ability to select a good set of parameters.

To help overcome this problem, a cluster ordering method called **OPTICS** (Ankerst *et al.*, 1999) was proposed. Rather than producing a data set clustering explicitly, OPTICS computes an augmented *cluster ordering* for automatic and interactive cluster analysis.

Because of the need to analyze the neighborhood of an object, both DBSCAN and OPTICS rely on a spatial index structure like R^*-tree (Beckmann *et al.*, 1990) or X-tree (Berchtold *et al.*, 1996) to process such neighborhood queries efficiently. However, the efficiency of answering such queries using these indexes drop increasingly with the number of dimensions. As such DBSCAN and OPTICS are not very efficient for high-dimensional data.

To handle high dimensional data efficiently, **DENCLUE** (Hinneburg & Keim, 1998) models the overall density of a point analytically as the sum of influence functions of data points around it. To compute the sum of influence functions efficiently, a grid structure is utilized. Experiments in (Hinneburg & Keim, 1998) show that DENCLUE outperforms DBSCAN by as much as 45 times. However, DENCLUE required careful selection of the clustering parameters which may significantly influence the quality of the clustering results.

In this section, we will have a detail look at the three density-based algorithms.

4.1 DBSCAN: A density-based clustering method based on connected regions with sufficiently high density

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) (Ester *et al.*, 1996) is a density-based clustering algorithm. The algorithm grows regions with sufficiently high density into clusters, and discovers clusters of arbitrary shape in spatial databases. The algorithm requires the input of two parameters ϵ and $Minpts$. The neighborhood within a radius ϵ of a given object is called the *ϵ-neighborhood* of the object and an object with at least $Minpts$ of objects within its ϵ-neighborhood is called a *core object*. The clustering formed from DBSCAN follows the rules below:

1. An object can only belong to a cluster if and only if it lies within the ϵ-neighborhood of some core object in the cluster.
2. A core object o within the ϵ-neighborhood of another core object p must belong to the same cluster as p.
3. A non-core object q within the ϵ-neighborhood of some core objects $p_1, ..., p_i$, $i > 0$, must belong to the same cluster to at least one of the core objects from $p_1, ..., p_i$.
4. A non-core object r which does not lie within the ϵ-neighborhood of any core object is considered to be noise.

Example 4.1 Consider Figure 5 for a given ϵ represented by the radius of the circles, and, say, let $MinPts = 3$. We represent core objects as solid points while the non-core objects are represented by hollow points. In the figure, two clusters, C1 and C2, are discovered by the DBSCAN algorithm. As can be seen data objects that belongs to C1 or C2 all lies within the ϵ-neighborhood at least one core object from C1 or

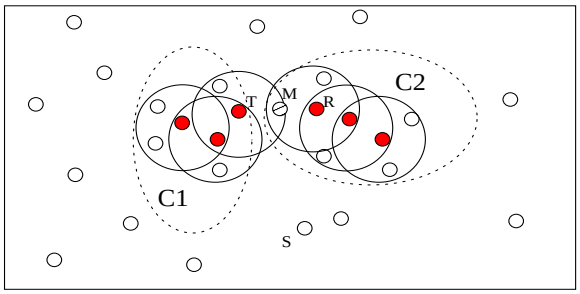


Figure 5: Two clusters discovered by DBSCAN.

C2 and there are no two core objects such that they lie within the ϵ-neighborhood of each other and yet belong to different clusters. A non-core object like M lies within the ϵ-neighborhood of T and R which are core-objects from C1 and C2 respectively. It thus can be assigned to either C1 or C2 since it is at the border of the two clusters. Finally, the object S is deemed to be noise because it is a non-core object and is not in the ϵ-neighborhood of any core object.

□

To discover clusters in the data, DBSCAN checks the ϵ-neighborhood of each point in the database. If the ϵ-neighborhood of a point p contains more than $MinPts$, a new cluster with p as a core object is created. Objects in the ϵ-neighborhood of p are then added to this new cluster. Core objects among these newly added members will then go through the same process as p to "grow" the cluster. When no more core object is found in this growing process, another core object will then be taken from the database for the growing of another new cluster. Note that during the growing process of DBSCAN, a core object which already belongs to another cluster may be encountered and this will result in the merge of the two clusters. The process terminates when no new point can be added to any cluster.

4.2 OPTICS: Ordering Points To Identify the Clustering Structure

Although DBSCAN (the density-based clustering algorithm described in Section 4.1) can discover clusters of arbitrary shapes in noisy data, it is however sensitive to the two input parameters, ϵ and $MinPts$. To discover clusters which are deemed acceptable to a user, he/she may need to run the algorithm many times while setting different parameter values for each run. Without any guidance in setting these parameters, a large amount of time can be wasted on such a trial-and-error approach.

To help overcome this difficulty, a cluster ordering method called OPTICS (Ordering Points To Identify the Clustering Structure) was proposed. Like DBSCAN, OPTICS requires the input of the two parameters, ϵ and $Minpts$. However, instead of producing the clustering result for one pair of parameter values, OPTICS produces an ordering of the data points such that clustering result for any lower value of ϵ and similar value of $Minpts$ can be visualized and computed easily.

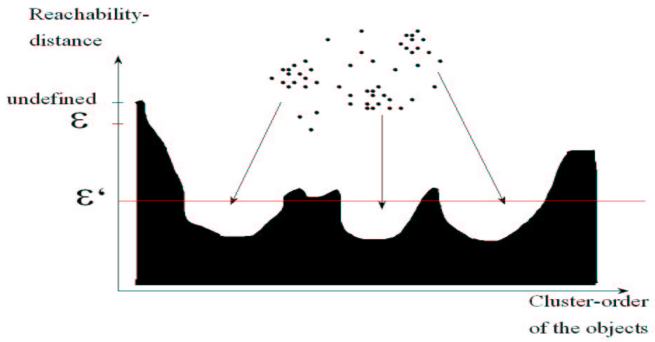


Figure 6: Cluster ordering in OPTICS.

By examining DBSCAN, one can easily see that for a constant $MinPts$-value, lowering ϵ to a new value ϵ' will bring about two effects:

1. Core objects might become non-core objects because it does not have at least $MinPts$ data objects in its ϵ'-neighborhood.
2. Non-core objects which are originally in the ϵ-neighborhood of some core objects may become noise either because they are not in the ϵ'-neighborhood of these core objects or because these core objects have become non-core.

As can be seen, both of these effects will result in a set of clusters which are *completely contained* in the earlier set of clusters found with the higher value of ϵ. Therefore, in order to produce a set of ordering of density-based clusters, it is sufficient for us to store some threshold values for each data objects such that the effect of a lower ϵ can be easily seen. These values that need to be stored are — *core-distance* and *reachability-distance*:

- The *core-distance* of an object p is, $core(p)$, the smallest distance such that its $core(p)$-neighborhood contains exactly $MinPts$ objects. If p is not a core object initially with respect to ϵ, this value is undefined for p.
- The *reachability-distance* of an object p with respect to another object o, $reach(o, p)$, is the smallest distance such that p is within the $reach(o, p)$-neighborhood of o and o remains a core object with respect to $reach(o, p)$. If o is not a core object initially with respect to ϵ, then reach(o, p) is undefined.

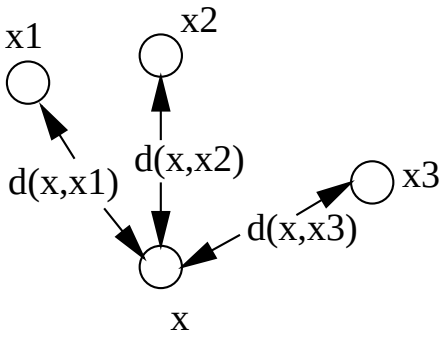
The OPTICS algorithm creates an ordering of the objects in a database, additionally storing the core-distance and a suitable reachability-distance for each object. Such information is sufficient for the extraction of all density-based clusterings with respect to any distance ϵ' that is smaller than the distance ϵ used in generating the order.

The cluster ordering of a data set can be represented graphically, which helps its understanding. For example, Figure 6 is the reachability plot for a simple 2-dimensional data set, which presents a general overview of how the data are structured

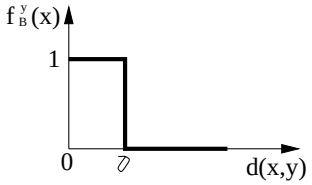
and clustered. Methods have also been developed for viewing clustering structures of high-dimensional data.

4.3 DENCLUE: Clustering based on density distribution functions

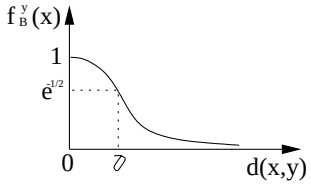
DENCLUE (DENsity-based CLUstEring) is a clustering method based on a set of density distribution functions. The method is built on the following ideas: (1) the influence of each data point can be formally modeled using a mathematical function, called an *influence function*, which describes the impact of a data point within its neighborhood; (2) the overall density of the data space can be modeled analytically as the sum of the influence function of all data points; and (3) clusters can then be determined mathematically by identifying *density attractors*, where density attractors are local maxima of the overall density function.



(a) $f_B^D(x) = f_B^{x_1}(x) + f_B^{x_2}(x) + f_B^{x_3}(x)$.



(b) Square-wave.



(c) Gaussian.

Figure 7: Computing density function from influence functions.

Let x and y be objects in F^d, a d-dimensional feature space. The *influence function* of data object y on x is a function $f_B^y : F^d \to R_0^+$, which is defined in terms of a basic influence function f_B,

$$f_B^y(x) = f_B(x, y). \tag{4.10}$$

In principle, the influence function can be an arbitrary function that takes in the distance between x and y, $d(x, y)$, as an input and computes the influence that y has

on x. The distance function, $d(x, y)$, should be reflexive and symmetric, such as the Euclidean distance function. Two examples of influence function are a *square-wave function*, and a *Gaussian function* which graphs with respect to $d(x, y)$ are depicted in Figure 7(b) and Figure 7(c) respectively. The definition of these two influence function are as follow:

The square-wave influence function

$$f_{Square}(x, y) = \begin{cases} 0 & if\ d(x, y) > \sigma \\ 1 & otherwise \end{cases} \tag{4.11}$$

The Gaussian influence function,

$$f_{Gauss}(x, y) = e^{-\frac{d(x,y)^2}{2\sigma^2}}. \tag{4.12}$$

The **density function** at an object $x \in F^d$ is defined as the sum of influence functions of all data points. $D = \{x_1, \ldots, x_n\} \subset F^d$, the density function at x is defined as,

$$f_B^D(x) = \sum_{i=1}^{n} f_B^{x_i}(x). \tag{4.13}$$

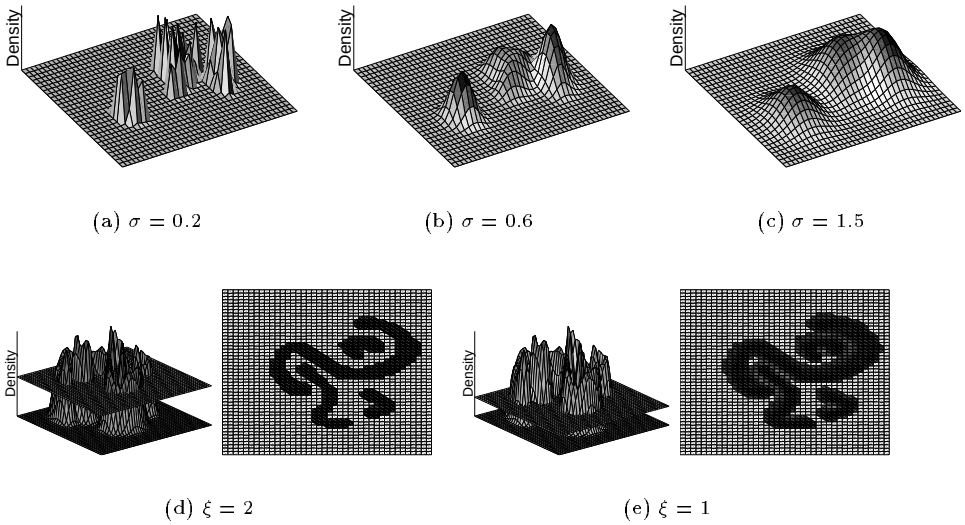
For example, the density function at point x in Figure 7(a) will be $f_B^D(x) = f_B^{x_i}(x) + f_B^{x_2}(x) + f_B^{x_3}(x)$. From the density function, one can define the *gradient* function at a point x,

$$\bigtriangledown f_B^D(x) = \sum_{i=1}^{n} (x_i - x) f_B^{x_i}(x). \tag{4.14}$$

which is in fact a vector that indicates the strength and direction where most of x's influence comes from. The density function is also used to locate *density attractors* which are the local maximas in the overall density function. A point x is said to be *density-attracted* to a density attractor x^* if there exist a set of points $x_0, x_1, ..., x_k, x_{k+1}$ such that $x_0 = x$, $x_{k+1} = x^*$ and the gradient of x_{i-1} is in the direction of x_i for $0 < i < k + 1$. For a continuous and differentiable influence function, a hill climbing algorithm guided by the gradient can be used to determine the density attractor of each data point.

Based on these notions, both *center-defined cluster* and *arbitrary-shape cluster* can be formally defined. A *center-defined cluster* for a density attractor x^* is a subset C that is *density-attracted* by x^*, and $f_B^D(x^*)$ is greater than a user-defined threshold ξ; otherwise (i.e., if its density function value is less than ξ), it is considered an outlier. An *arbitrary-shape cluster* is a set of C's, each being density-extracted, with the density function value no less than a threshold ξ, and where there exists a path P from each region to another, and the density function value for each point along the path is no less than ξ. Examples of center-defined and arbitrary-shape clusters are shown in Figure 4.3.

To form clusters based on these definitions, DENCLUE makes use of a grid-like structure to facilitate the calculation of the density function and the hill-climbing



(a) $\sigma = 0.2$ (b) $\sigma = 0.6$ (c) $\sigma = 1.5$

(d) $\xi = 2$ (e) $\xi = 1$

Figure 8: Examples of center-defined clusters (top row) and arbitrary-shaped clusters (bottom row). Figure is from (Hinneburg & Keim, 1998).

procedure. It should be noted that although DENCLUE utilizes a grid structure during clustering, it is different from grid-based clustering which we will discuss in the next section. In grid-based clustering, summarized information about the data points in each grid cell is stored. Such information is then used for clustering purpose while information about the individual data points is abandoned. In DENCLUE, however, the grid structure is used in efficient computation of the sum of influence functions at each data point and the information about each data point is utilized throughout the clustering process.

5 Grid-based methods

As mentioned in the previous section, density-based methods like DBSCAN and OPTICS are index-based methods which will face a breakdown in efficiency when the number of dimensions is high. To enhance the efficiency of clustering, a grid-based clustering approach uses a grid data structure. It quantizes the space into a finite number of cells which form a grid structure on which all of the operations for clustering are performed. The main advantage of the approach is its fast processing time which is typically independent of the number of data objects, yet dependent on only the number of cells in each dimension in the quantized space.

Some typical examples of the grid-based approach include **STING** (Wang *et al.*, 1997), which explores statistical information stored in the grid cells; **WaveCluster** (Sheikholeslami *et al.*, 1998), which clusters objects using a wavelet transform method; and **CLIQUE** (Agrawal *et al.*, 1998), which represents a grid- and density-

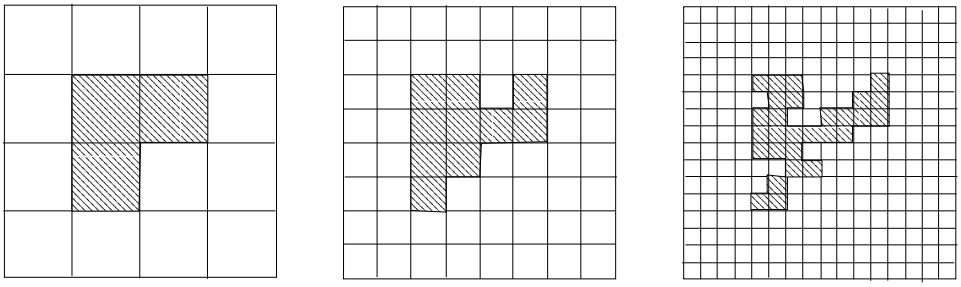


Figure 9: Three consecutive layers of a STING structure.

based approach for clustering in high-dimensional data space.

A grid-based approach is usually more efficient than a density-based approach especially in high-dimensional space. On the other hand, the use of summarized information causes it to increasingly lose effectiveness as the number of dimensions increases.

5.1 STING: A STatistical INformation Grid approach

STING (STatistical INformation Grid) is a grid-based multiresolution data structure in which the spatial area is divided into rectangular cells. There are usually several levels of such rectangular cells corresponding to different levels of resolution, and these cells form a hierarchical structure: each cell at a high level is partitioned to form a number of cells at the next lower level. Statistical information regarding the attributes in each grid cell (such as the mean, maximum, and minimum values) are precomputed and stored. These statistical parameters are useful for query processing, as described below.

In Figure 9, we show three consecutive layers of a STING structure where each parent cell is divided into four cells at the next immediate level. Statistical parameters of higher level cells can easily be computed from the parameters of the lower level cells. These parameters include the following: the attribute-independent parameter, *count*, and attribute-dependent parameters, m (mean), s (standard deviation), *min* (minimum), *max* (maximum), and the type of *distribution* that the attribute value in the cell follows, such as *normal, uniform*, *exponential*, or *none* (if the distribution is unknown). Data are loaded into the structure starting from the bottom-most level, i.e., the level with the highest resolution. The parameters, *count*, m, s, *min* and *max*, of the bottom level cells are calculated directly from the data. The value of *distribution* may either be assigned by the user if the distribution type is known beforehand, or obtained by hypothesis tests such as the χ^2-test. The type of distribution of a higher level cell can be computed based on the majority of distribution types of its corresponding lower level cells in conjunction with a threshold filtering process. If the distributions of the lower level cells disagree with each other and fail the threshold test, the distribution type of the high level cell is set to *none*.

To perform clustering on such a data structure, users must first supply the density

level as an input parameter. Using this parameter, a top-down, grid-based method is used to find regions with sufficient density by adopting the following procedure. First, a layer within the hierarchical structure is determined from which the query-answering process is to start. This layer typically contains a small number of cells. For each cell in the current layer, we compute the confidence interval (or estimated range of probability) that the cell will be relevant to the result of the clustering. Cells which does not meet the confident level are deemed irrelevant and will be removed from further consideration. The irrelevant cells are then refined to finer resolution by repeating the procedure at the next level of the structure. This process is repeated until the bottom layer is reached. At this time, if the query specification is met, the regions of relevant cells that satisfy the query are returned. Otherwise, the data that fall into the relevant cells are retrieved, and further processed until they meet the requirements of the query. Figure 9 shows how a cluster is refined at each level of a STING structure.

The multiresolution approach using STING offers several advantages over some other clustering methods: First, the grid-based computation is *query-independent* since the statistical information stored in each cell represents the summary information of the data in the grid cell, independent of the query. Second, the grid structure facilitates parallel processing and incremental updating, and third, the method is very efficient. STING goes through the database once to compute the statistical parameters of the cells, and hence the time complexity of generating clusters is $O(n)$, where n is the total number of objects. After generating the hierarchical structure, the query processing time is $O(g)$, where g is the total number of grid cells at the lowest level, which is usually much smaller than n.

Since STING uses a multiresolution approach to perform cluster analysis, the quality of STING clustering depends on the granularity of the lowest level of the grid structure. If the granularity is very fine, the cost of processing will increase substantially; however, if the bottom level of the grid structure is too coarse, it may reduce the quality of cluster analysis. Moreover, STING does not consider the spatial relationship between the children and their neighboring cells for construction of the parent cell. As a result, the shapes of the resulting clusters are isothetic, that is, all of the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected. This may lower the quality and accuracy of the clusters despite the fast processing time of the technique.

5.2 WaveCluster: Clustering using wavelet transformation

WaveCluster is a multiresolution clustering algorithm which first summarizes the data by imposing a multidimensional grid structure onto the data space. It then uses a *wavelet transformation* to transform the original feature space, finding dense regions in the transformed space.

In this approach, each grid cell summarizes the information of a group of points which map into the cell. This summary information typically fits into main memory for use by the multiresolution wavelet transform and the subsequent cluster analysis.

A *wavelet transform* is a signal processing technique that decomposes a signal into different frequency sub-bands. The wavelet model can be applied to n-dimensional signals by applying a one-dimensional wavelet transform n times. Convolution with an

appropriate kernel function results in a transformed space where the natural clusters in the data become more distinguishable. Clusters can then be identified by searching for dense regions in the new domain.

Wavelet transformation is useful for clustering because it offers the following advantages.

- First, it provides unsupervised clustering. It uses hat-shape filters which emphasize regions where the points cluster, while at the same time, suppressing weaker information outside of the cluster boundaries. Thus, dense regions in the original feature space act as attractors for nearby points, and as inhibitors for points that are further away. This means that the clusters in the data automatically stand out and "clear" the regions around them. Thus, another advantage is that wavelet transformation can automatically result in the removal of outliers.

Figure 10: A sample of 2-dimensional feature space. Figure is from (Sheikholeslami *et al.*, 1998), reprint with permission from the VLDB Endowment

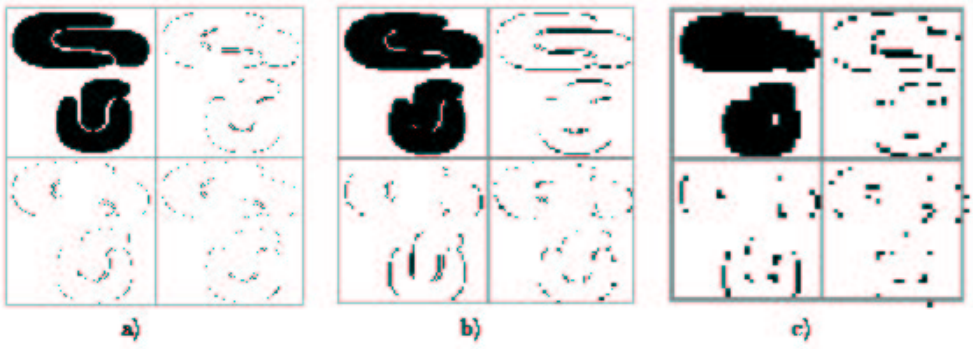


Figure 11: Multiresolution of the feature space in Figure 10 at a) scale 1; b) scale 2; c) scale 3. Figure is from (Sheikholeslami *et al.*, 1998), reprint with permission from the VLDB Endowment

- The multiresolution property of wavelet transformations can help the detection of clusters at varying levels of accuracy. For example, Figure 10 shows a sample

of 2-dimensional feature space, where each point in the image represents the attribute or feature values of one object in the spatial data set. Figure 11 shows the resulting wavelet transformation at different resolutions, from a fine scale (scale 1) to a coarse scale (scale 3). At each level, the four sub-bands into which the original data are decomposed are shown. The sub-band shown at the upper-left quadrant emphasizes the average neighborhood around each data point The sub-band at the upper-right quadrant emphasizes the horizontal edges of the data. The sub-band at the lower-left quadrant emphasizes the vertical edges, while the sub-band at the lower-right quadrant emphasizes the corners.

- Wavelet-based clustering is very fast, with a computational complexity of $O(n)$ where n is the number of objects in the database. The algorithm implementation can be made parallel.

WaveCluster is a grid-based and density-based algorithm. It conforms with all of the requirements of a good clustering algorithm: It handles large data sets efficiently, discovers clusters with arbitrary shape, successfully handles outliers, and is insensitive to the order of input. In experimental studies, WaveCluster was found to outperform BIRCH, CLARANS and DBSCAN in terms of both efficiency and clustering quality. Experiments in (Yu *et al.*, 1998) show that WaveCluster is able to handle data with up to 20 dimensions.

5.3 CLIQUE: Clustering high-dimensional space

The CLIQUE algorithm integrates density-based and grid-based clustering. Unlike other clustering algorithms described earlier, CLIQUE is able to discover clusters in the subspaces of the data. It is useful for clustering high dimensional data which are usually very sparse and do not form clusters in the full dimensional space.

In CLIQUE, the data space is partitioned into non-overlapping rectangular unit by equal space partition along each dimension. A unit is **dense** if the fraction of total data points contained in it exceeds an input model parameter. A cluster is defined as a maximal set of *connected dense units*.

CLIQUE performs multidimensional clustering by moving from the lower dimensional space to the higher. When seaching for dense units at the k-dimensional space, CLIQUE makes use of information that is obtained from clustering at the $(k-1)$-dimensional space to prune off unnecessary search. This is done by observing the *Apriori property* used in association rule mining (Agrawal & Srikant, 1994). In general, the property employs prior knowledge of items in the search space so that portions of the space can be pruned. The property, adapted for CLIQUE, states the following: *If a k-dimensional unit is dense, then so are its projections in $(k-1)$-dimensional space.* That is, given a k-dimensional candidate dense unit, if we check its $(k-1)$-th projection units and find any that is not dense, then we know that the k-th dimensional unit cannot be dense either. Therefore, we can generate potential candidate dense units in the k-dimensional space from the dense units found in the $(k-1)$-dimensional space. We illustrates this principle in Figure 12. In general, the resulting space searched is much smaller than the original one. The dense units that are then examined to determine the clusters.

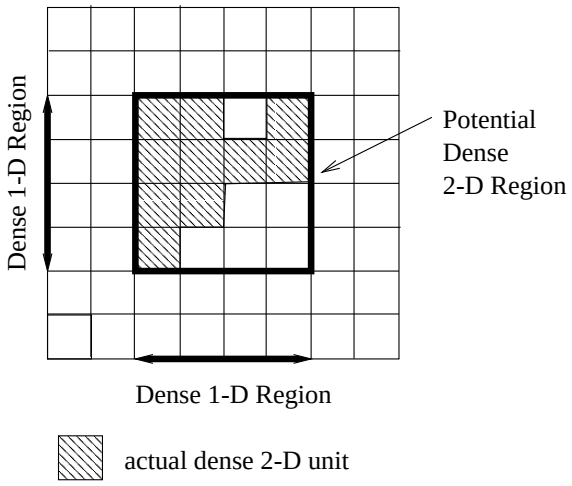


Figure 12: CLIQUE: Determining potential region with dense units.

Having found the clusters, CLIQUE generates a minimal description for each cluster as follows. For each cluster, it determines the maximal region that covers the cluster of connected dense units. It then determines a minimal cover for each cluster.

CLIQUE automatically finds subspaces of the highest dimensionality such that high density clusters exist in those subspaces. It is insensitive to the order of input tuples and does not presume any canonical data distribution. It scales linearly with the size of input and has good scalability as the number of dimensions in the data is increased. However, the accuracy of the clustering results may be degraded at the expense of the simplicity of the method.

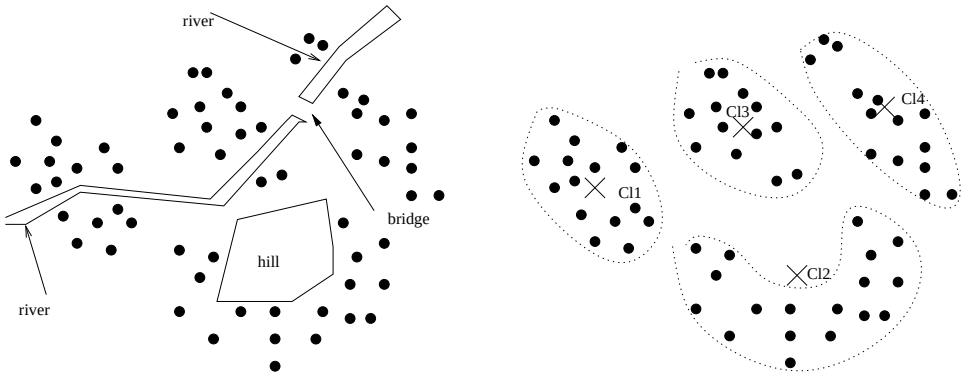
6 Constraint-Based Cluster Analysis

The development of spatial clustering on large database has provided many useful tools for the analysis of geographical data. However, most of these algorithms provide very few avenues for users to specify real life constraints which must be satisfied with the clustering. In order for spatial clustering to be more useful, additional research must be done on providing users with the ability to incorporate real life constraints into the clustering algorithm. In this section, we introduce two pieces of work which are of such nature.

6.1 COD: Clustering with obstructed distance

In many applications, physical obstacles like mountains and rivers could affect the result of a clustering algorithm substantially. For example, consider a bank planner who wishes to locate 4 ATMs in the area shown in Figure 13(a) to serve the customers who are represented by points in the figure. In such a situation, however, natural obstacles exist in the area and they should not be ignored. This is because ignoring these obstacles will result in clusters like those in Figure 13(b) which are obviously

inappropriate. For example, Cluster Cl_1 is, as a result of clustering, split by a river, and some customers on one side of the river will have to travel a long way to the ATM located at the other side. Thus the ability to handle such real life constraints in a clustering algorithm is a very important one.



(a) Customers' location and obstacles.

(b) Clusters formed when ignoring obstacles.

Figure 13: Planning the location of ATMs

In order to handle such constraints, a new clustering problem called *Clustering with Obstructed Distance(COD)* [1] is introduced in (Tung *et al.*, 2001b). The COD problem is defined as follows.

A set P of n points $\{p_1, p_2, ..., p_n\}$ and a set O of m *non-intersecting* obstacles $\{o_1, ..., o_m\}$ are given in a two-dimensional region, R, with each obstacle o_i represented by a simple polygon. The distance, $d(p, q)$, between any two points, p and q, is defined as the length of the shortest Euclidean path from p to q without cutting through any obstacles. To distinguish this distance from the direct Euclidean distance, this distance is referred to as *obstructed distance*. The objective in solving the COD problem is to partition P into k clusters $C_1, ..., C_k$ such that the following square-error function, E, is minimized:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} d^2(p, m_i)$$

where m_i is the center of cluster C_i that is determined also by the clustering.

Note that the use of simple polygons to represent obstacles is a generic one, as a line obstacle can be represented by a very thin and long polygon while a planar network is basically made up of the edges of polygons that are located side by side.

In order to solve the COD problem, an algorithm *COD-CLARANS (Clustering with Obstacle Entities based on CLARANS)* is proposed in (Tung *et al.*, 2001b) which makes use of two techniques to improve the efficiency of clustering.

First, a pre-clustering step similar to those in BIRCH (Zhang *et al.*, 1996) and CHAMELEON (Karypis *et al.*, 1999) is taken to group the objects into a set of *micro-clusters*. A micro-cluster contains summarized information about objects which are so

[1] The COD problem was formally called the COE (Clustering with Obstacle Entities) problem.

close together that they are most likely to be in the same cluster. By performing the clustering on these compressed micro-clusters instead of the actual data, processing can be achieved in the main memory and the cost of computing the distance between objects and the cluster centers can be reduced.

The second technique COD-CLARANS is to use a lower bound of the square-error function E to prune search space. As mentioned in Section 2, CLARANS is a generate-and-test algorithm which randomly picks a cluster center, the "goodness" of o_{random} over o_i, E, must be computed with o_{random} as a cluster center and if o_{random} is a bad choice, the expensive computation of E will be wasted. To avoid this unnecessary computation of E, a more easily computed lower bound of E, E', is first computed. In order to compute E', the distance between o_{random} and the micro-clusters E' is first underestimated by using direct Euclidean distance instead of obstructed distance. Thus, if the direct Euclidean distance between a micro-cluster p and o_{random} is shorter than the obstructed distance between p and the other $k-1$ unchanged cluster centers, then p is assigned to o_{random} and the direct Euclidean distance between them will be used when computing E'. This makes E' a lower bound for the actual square-error function E. Since E' is a lower bound of E, o_{random} can be abandoned without computing E if E' is already higher than the square-error function of the best solution so far.

Experiments in (Tung *et al.*, 2001b) show that these two techniques help reduce processing time significantly with no or little degrade in cluster quality.

6.2 Handling operational constraints in clustering

While physical obstacles provide one type of constraints, operational requirements provide another. For example, a manager who is handling facility allocation may want to ensure that all the facilities are fully utilized and thus want each cluster to contain at least a certain number of clients. Similarly, when a manager is planning discount packages to cater to different groups of customers, he/she may want to partition his/her customers into k groups such that each group contains a certain number of customers while ensuring that customers in each group share similar characteristics. We refer to such constraints as operational constraints.

In (Tung *et al.*, 2001), a study is done on a type of constraint called *existential constraint* which requires that the data objects be grouped into k clusters containing at least c *pivot objects* while minimizing the squared-error function [2] to the cluster centers. *Pivot objects* are objects which satisfy certain criteria. An example of an existential constraint could be "each cluster must contain more than 10 customers whose asset is greater than 1 million dollars". Here the pivot objects are those customers who own more than 1 million dollars in asset and each cluster must contain at least 10 of them. Each cluster however could contain an arbitrary number of non-pivot objects.

One important complication when an existential constraint is introduced into clustering is that a property called the **Nearest Representative Property (NRP)** does not hold anymore. Briefly, the NRP states that data objects must be assigned to their nearest center in order to minimize the squared-error function. Because of this property, existing partitioning algorithms like k-means or k-medoids focus on

[2] Definition of squared-error function is in Section 2.1

searching for the k centres that will give a good solution. Once these k centers are found, the membership of the data objects fall automatically into place. However, since NRP does not hold in constrained clustering, an algorithm which attempts to solve this problem will need to test the large number of combinations of separating the data objects into k clusters and find a valid and good solution. To solve this problem, the algorithm in (Tung *et al.*, 2001) adopt a strictly descending stragety when searching for a good clustering solution. The algorithm is an iterative refinement algorithm involving two phases: *pivot movement* and *deadlock resolution*. Both phases are shown to be NP-hard when optimal solution is required. As such several heuristics are used in the algorithm. To scale up the algorithm for large data sets, a heuristic called *micro-cluster sharing* is used. Experiments show that the algorithm is both effective and efficient.

Besides existential constraints, other types of constraints are also discussed in (Tung *et al.*, 2001). These include *universal constraints*, *existential-like constraints*, *summation constraints* and *averaging constraints*. *Universal constraints* are constraints in which a specific condition must be satisfied by *every* object in a cluster. This can be simply reduced to the unconstrained clustering problem by running the clustering algorithm on only those objects that satisfy the condition. *Existential-like constraints* are constraints which are similar in nature to existential constraints. An example of an existential-like constraint is "each cluster must contain less than 1000 customers". Instead of ensuring that each cluster contain more than a certain number of pivot objects, we have to ensure that each cluster contain more than one "hole". The number of "holes" in the cluster represents the number of pivot objects which must be added to it to invalidate the constraint. In the previous example, a cluster which contain 995 customers will contain 6 holes. As can be seen, existential-like constraints are very similar in nature to existential constraints and can be easily handled by simple modification to an algorithm that handle existential constraint. *Summation* and *averaging* constraints are constraints which involve summation and averaging of some numerical attributes of the data objects. For example, a summation constraint could be "each cluster must have customers whose sum of asset is greater than 1 million dollars". Even computing an initial solution for summation and averaging constraint is an NP-hard problem similar to a bin-packing or knapsack problem (Garey & Johnson, 1979).

Although introducing operational constraints into clustering generally results in higher processing time, it is nonetheless necessary for clustering to be more useful in real-life applications.

7 Conclusion

We have presented an overview of clustering algorithms that are useful to the geographical community. We categorize them into four categories: partitioning-based, hierarchical-based, density-based and grid-based .

Partitioning methods like k-means, k-medoids and EM clustering are methods which make uses of a technique called iterative reallocation to improve the clustering quality from an initial solution. As such methods tend to find clusters that are of spherical shape and similar in size, they are more useful for applications like facility

allocation where the objective is not to find natural cluster but to minimize the sum of distances from the data objects to their cluster centers.

Unlike partitioning-based clustering algorithms which reallocate data objects from one cluster to another in order to improve the clustering quality, hierarchical clustering algorithms fixed the membership of a data object once it has been allocated to a cluster. Coupled with an over-simplistic method for merging or splitting clusters, earlier hierarchical clustering algorithms like AGNES and DIANA tend to produce the erroneous clusters. To find better quality clusters, BIRCH first performs a hierarchical clustering on the data to compress them into clustering features before using iterative relocation to improve the clustering quality in the main memory. A different approach adopted by CURE and CHAMELEON uses more complex criteria when merging clusters. Both approaches are successful in improving the quality of clustering.

Instead of using distance to judge the membership of a data objects, density-based clustering algorithms like DBSCAN make use of the density of data points within a region to discover clusters. To define the density around the region of a data object, DBSCAN requires the input of two parameters, Eps and $MinPts$. A region around an object, o, is considered dense only if $MinPts$ objects are within a radius Eps of o and only points within a dense region will form clusters. Due to this definition, DBSCAN is very sensitive to the two input parameters. To help overcome this problem, an algorithm OPTICS is derived which computes an augmented cluster ordering for a wide range of parameter settings, which gives users the flexibility of interactive cluster analysis. However, the requirement to do spatial queries for both DBSCAN and OPTICS results in a loss of efficiency for high dimensional clustering. This problem is addressed by DENCLUE which models the overall density of a point as the sum of influence functions of data points around it and utilizes a grid structure to handle the computation efficiently. However, DENCLUE requires a careful choice of clustering parameters.

To increase the efficiency of clustering, grid-based clustering methods approximate the dense regions of the clustering space by quantizing it into a finite number of cells and identifying cells that contain more than a number of points as dense. Clusters are then formed by connecting these dense cells. Grid-based clustering methods include STING, which explores statistical information stored in the grid cells; WaveCluster, which clusters objects using a wavelet transform method; and CLIQUE, which discovers clusters in subspaces using the Apriori principle. Although a grid-based approach is usually more efficient than a density-based approach, the use of summarized information causes it to lose effectiveness as the number of dimensions increases.

Towards the end of this survey, we introduce the concept of constraint-based clustering. The two types of constraints that we introduced are physical constraints and operational constraints. Physical constraints are constraints imposed by physical obstacles like mountains and rivers. A clustering algorithm called COD-CLARANS is introduced to perform efficient clustering with physical constraints. Operational constraints are constraints which are imposed by the operational consideration of the users. Operational constraints like existential constraints and summation constraints can result in NP-hard problems and we are now looking at possible heuristics and approximation algorithms for solving them.

Finally, we would like to add that the clustering algorithms we introduce here are

those which we believe are useful for the geographical community. Clustering has also been investigated by statisticans, machine learning experts and people in other fields. Due to the limited scope of this paper, algorithms from these fields have not been covered in the discussion. We hope that readers will look into them if such situation arises.

References

Agrawal, R., & Srikant, R. 1994 (September). Fast Algorithms for Mining Association Rules. *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94), Pages 487–499.*

Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. 1998 (June). Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), Pages 94–105.*

Ankerst, M., Breunig, M., Kriegel, H.-P., & Sander, J. 1999 (June). OPTICS: Ordering Points To Identify the Clustering Structure. *Proc. 1999 ACM-SIGMOD Conf. on Management of Data (SIGMOD'99), Pages 49–60.*

Beckmann, N., Kriegel, H.-P., Schneider, R., & Seeger, B. 1990 (June). The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. *Proc. 1990 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'90), Pages 322–331.*

Berchtold, S., Keim, D., & Kriegel, H.-P. 1996 (Sept.). The X-tree: An Efficient and Robust Access Method for Points and Rectangles. *Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96)Pages 28–39* .

Bradley, P. S., Fayyad, U. M., & Reina, C. A. 1998 (Nov.). Scaling EM(Expectation-Maximization) Clustering to Large Databases. *Microsoft Research Technical Report 98-35.*

Dempster, A. P., Laird, N. M., & Rubin, D. B. 1977. Maximum Likelihood from Incomplete Data via the EM algorithm. *Journal of the Royal Statistical Societyi, Series B*, **39**, 1–38.

Ester, M., Kriegel, H.-P., & Xu, X. 1995 (August). Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification. *Proc. 4th Int. Symp. Large Spatial Databases (SSD'95), Pages 67–82.*

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. 1996 (August). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases. *Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD'96), Pages 226–231.*

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., & (eds.), R. Uthurusamy. 1996. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press.

Garey, M., & Johnson, D. 1979. *Computers and Intractability: a Guide to The Theory of NP-Completeness.* New York: Freeman and Company.

Guha, S., Rastogi, R., & Shim, K. 1998 (June). CURE: An Efficient Clustering Algorithm for Large Databases. *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), Pages 73–84.*

Han, J., & Kamber, M. 2000. *Data Mining: Concepts and Techniques.* Morgan Kaufmann.

Han, J., Ng, R., Lakshmanan, L., & Tung, A. K. H. 2000 (Jan.). On the Problem of Constrained Clustering, *Simon Fraser University, Computing Science Technical Report 4* .

Hinneburg, A., & Keim, D. A. 1998 (August). An Efficient Approach to Clustering in Large Multimedia Databases with Noise. *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)Pages 58–65* .

Hou, J. 1999. *Clustering with Obstacle Entities*. School of Computing Science, Simon Fraser University, Canada: M.Sc. Thesis.

Karypis, G., Han, E.-H., & Kumar, V. 1999. CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling. *COMPUTER*, **32**, *Pages 68–75.*

Kaufman, L., & Rousseeuw, P. J. 1990. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons.

MacQueen, J. 1967. Some Methods for Classification and Analysis of Multivariate Observations. *Proc. 5th Berkeley Symp. Math. Statist, Prob.*, **1**, *Pages 281–297.*

Ng, R., & Han, J. 1994 (September). Efficient and Effective Clustering Method for Spatial Data Mining. *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94), Pages 144–155.*

Scott, D. W. 1992. *Multivariate Density Estimation*. New York: John Wiley & Sons.

Sheikholeslami, G., Chatterjee, S., & Zhang, A. 1998 (August). WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98), Pages 428–439* .

Silverman, B. W. 1986. *Density Estimation for Statistics and Data Analysis*. London: Chapman & Hill.

Tung, A. K. H., Han, J., Ng, R., & Lakshmanan, L. Jan 2001 . Constrained Clustering on Large Database, *To Appear in Proc. of the 8th International Conference on Database Theory (ICDT'01).*

Tung, A. K. H., Hou, J., & Han, J. Apr. 2001b. Spatial Clustering in the Presence of Obstacles. *To Appear in Proc. The 17th International Conference on Data Engineering (ICDE'01).*

Wang, W., Yang, J., & Muntz, R. 1997 (Aug.). STING: A Statistical Information Grid Approach to Spatial Data Mining. *Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97), Pages 186–195.*

Yu, D., Chatterjee, S., Sheikholeslami, G., & Zhang, A. 1998 (Nov.). Efficiently Detecting Arbitrary Shaped Clusters in Very Large Datasets with High Dimensions. *SUNY Buffalo, Computer Science Technical Report 98-08.*

Zhang, T., Ramakrishnan, R., & Livny, M. 1996 (June). BIRCH: an efficient data clustering method for very large databases. *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96), Pages 103–114.*