# Abstract

Clustering means grouping similar objects into classes. In the result, objects within a same group should bear similarity to each other while objects in different groups are dissimilar to each other. As an important component of data mining, much research on clustering has been conducted in different disciplines. Clustering is a very popular practical problem in many areas.

This essay presents a taxonomy of the major and recently devised clustering techniques, and identifies recent advances in the field. Detailed analysis, implementation and experiments for major clustering methods are given in the essay. We also addressed the problem of applying clustering techniques in the context of web mining in particular for grouping web sessions using algorithms that can handle categorical data efficiently.

# Acknowledgements

First of all, my thanks go to my supervisor Dr. Osmar R. Zaïane for his guidance and funding.

Thanks to Jörg Sander who reviewed this work and offered constructive criticisms.

To Jun Luo and Andrew Foss, thanks for sharing part of your work.

I owe special thanks to Jie Sheng who gave me great support in my study.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Clustering is a process of grouping objects into groups (i.e. clusters), so that objects within one cluster are *similar* to each other and objects in different clusters are *dissimilar* to each other [HK00]. The keywords in this widely accepted definition of clustering are *similar* and *dissimilar*: What is "*similar/dissimilar*"? How can people tell whether two data objects are *similar* or *dissimilar* to each other? Much of the research in clustering is actually around these two keywords, i.e. the definition of *similarity/dissimilarity*. In clustering analysis, a similarity function or dissimilarity function is used to measure the *similarity/dissimilarity* between data objects. In some cases, definition of such a function is obvious. For example, in two dimensional spatial data, dissimilarity between two data objects can simply be defined as the distance between them. However, in most of the real application problems it is usually not easy to properly define similarity function.

Cluster analysis has been widely used in many application areas which including data analysis, image processing, market analysis, pattern recognition, etc.

In this essay project, we conduct a survey and extensive experiments on the data clustering techniques, and then concentrate on clustering web transactions to investigate possible similarity measure for web sessions. Possible applications for this part of the work include web session analysis, clustering on time series, clustering on DNA sequences, etc.

# Chapter 2

# Comparative Study of Clustering Algorithms

Due to the huge amount of data stored in databases, cluster analysis has become a very active topic in data mining research. There exist some very good survey papers [AMP99] [HKT01], but either they are out of date or only cover part of clustering techniques. Several research papers have been published in the past few years with different emphasis and ideas. Here we try to produce an up-to-date summarization and analysis of some known clustering methods.

## 2.1 Requirements for Data Clustering

In data mining, people have been seeking for effective and efficient clustering techniques in the analysis of large databases. These are the typical requirements for a good clustering technique in data mining [HK00]:

- **Scalability:** The cluster method should be applicable to huge databases. Techniques which can only be applied to small datasets are practically useless.
- **Ability to cluster different types of attributes:** Clustering objects could be of different types : numerical data, boolean data or categorical data. Ideally a clustering method should be suitable for all different types of data objects.

- **Ability to discover clusters with different shapes:** This is an important requirement for spatial data clustering. Many earlier clustering algorithms can only discover spherical shaped clusters. This ability is necessary for all the recent clustering techniques.

- **Minimal input parameter:** This means the method requires minimum domain knowledge for correct clustering. It is one of the main reasons why most of the clustering algorithms are not practical in real applications. Some very recent works try to overcome this problem.

- **Not sensitive to noise:** It is a typical requirement for clustering ability because noise exists everywhere in practical problems. A good clustering algorithm should be able to successfully perform clustering upon heavily noised data.

- **Insensitive to the order of input records:** The clustering method should give consistent results. In case the order of input data is changed, a good method should give the same clustering result.

- **Ability to handle high dimensionality:** A simple numerical clustering problem is often in 2-dimensions or 3-dimensions, but a real database could be in several dimensions. It is very challenging and a practical requirement that a clustering algorithm should be able to handle high dimensional data.

So far, there is no single algorithm that can fully satisfy all the above requirements (clustering properties of all the studied algorithms are summarized in Table 2.1 - 2.3 at the end of the chapter), but it is important to understand characteristics of each algorithm, so that the user is able to select the proper algorithm depending on different problem. Also recently, there are several new clustering techniques, some of which have made breakthroughs in their clustering abilities, thus it is necessary to put all these different clustering methods together in perspective. This is one of the motivations of this essay.

## 2.2 Taxonomy Study on Clustering Techniques

There exist a large number of clustering algorithms. Generally speaking, these clustering algorithms can be clustered into 4 groups: partitioning methods, hierarchical methods, density-based methods and grid-based methods. This section gives a taxonomy analysis and experimental study of representative methods in each group.

In order to examine the clustering ability of clustering algorithms, we performed experimental evaluation upon K-means [J.M67], CURE [SG98], ROCK [GRS99], DBSCAN [EKSX96], CHAMELEON [KHK99], WaveCluster [SCZ98] and CLIQUE [AGGR98]. The DBSCAN program is from its authors. CURE and ROCK codes are from the Department of Computer Science and Engineering, University of Minnesota. K-means, CHAMELEON, WaveCluster, and CLIQUE programs were locally implemented. We evaluate these algorithms by using two dimensional spatial data sets referenced and used in the CHAMELEON paper [KHK99] and data sets referenced and used in the WaveCluster paper [SCZ98]. The reason for using two dimensional spatial data is because it is easier to evaluate the quality of clustering result. Often people can intuitively identify clusters on two dimensional spatial data, while this is usually very difficult for high dimensional data sets.

We show the experimental results of each algorithm on the t7 data set from CHAMELEON paper. The data set is shown in Figure 2.1. This data set has 9 heavily noised clusters of different shapes, and it has clusters inside a cluster. In all the following clustering result figures, black points mean noise.

### 2.2.1 Partitioning methods

Suppose there are $n$ objects in the original data set, partitioning methods break the original data set into $k$ partitions ($k$ is the number of clusters required in output). The basic idea of partitioning is very intuitive, and the process of partitioning is typically to achieve certain optimal criterion iteratively.

The most classical and popular partitioning methods are k-means [J.M67] and k-medoid [LP90], where each cluster is represented by the gravity centre of

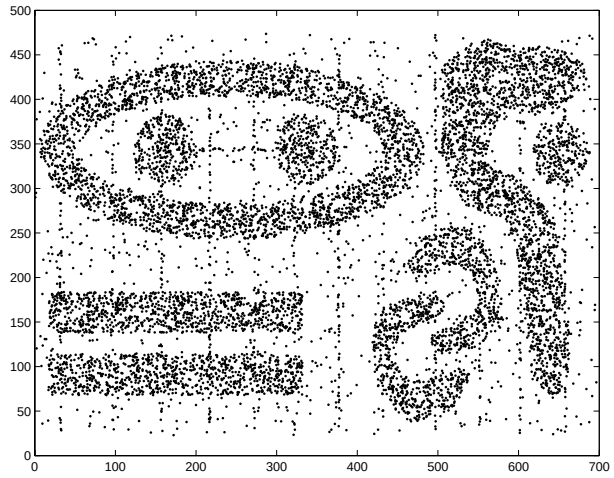


Figure 2.1: The test data set t7 which has 10000 data points.

the cluster in k-means method or by one of the "centre" objects of the cluster in k-medoid method. CLARANS [RJ94] is an improved k-medoid algorithms. It is more effective and more efficient than other k-medoid methods, for example PAM [LP90] and CLARA [LP90], but CLARANS assumes that all the original data set can be held in the main memory which may not be true for huge data sets. Another extension of k-means is k-modes method [Hua98], which is specially designed for clustering categorical data. Instead of "*mean*" in k-means method, k-modes method defined "*mode*" for each cluster. It defined dissimilarity measures of two categorical objects as the total mismatches of the corresponding attribute categories of the two objects. The modes of clusters are updated using a frequency-based method.

All the partitioning methods have similar clustering quality. The reason for this phenomenon is because all the partitioning methods use only one centre object or gravity centre to represent a cluster, and cluster quality is measured by the distances of all the other points of the same cluster from their centre point. This is often not enough for representing all the information of a cluster and not able to truly reflect cluster quality for non-spherical clusters or large clusters. Difficulties with partitioning methods include: (1) The choice of the output cluster number $k$ requires some domain knowledge which maybe not available in many circumstances, (2) difficulty in identifying clusters with large variation in sizes, (3) cannot identify clusters with non-convex shape, (4)

cannot identify clusters with elongated shapes.

Because this group of methods is relatively earlier than the other groups of methods, and they usually have similar clustering results, we implemented only k-means method in this group of methods as representative.

**k-means**

K-means needs a input parameter $k$ which specifies the number of output clusters; the parameter $k$ is common for all the partitioning methods. K-means uses the mean value of the objects in a cluster as the cluster centre. The method partitions the input set of $n$ data objects into $k$ clusters, so that the square-error function following reaches *minimal*:

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

where $x$ is the data object, $m_i$ is the mean of cluster $C_i$. However, depends on the starting mean of clusters, it is also possible that k-means can only reaches a local *minimal*.

K-means method works well if the clusters are spherical, well-separated, and if $k$ is known in advance. K-means' result on a simple testing data set is shown in Figure 2.2. This data set has five simple sphere well-separated clusters, and we see K-means successes on this simple testing data.

However, the method cannot cluster more complex data. K_mean's result on t7 is shown in Figure 2.3. From here we see K-means still tends to find sphere clusters, so that it is not able to find arbitrary shaped clusters. This is actually a general problem for all the partition methods because they use only one gravity centre to represent a cluster, and clustering of all the other points are decided by their relative closeness to the gravity centres of clusters. Only one point, gravity centre's information is considered. Partition methods do not consider relative closeness and connection among all the points in one cluster.

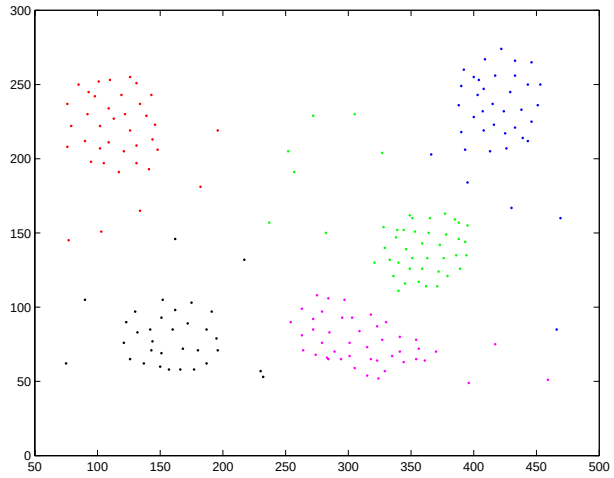


Figure 2.2: K_means' clustering result on C3.dat. number_of_clusters = 5

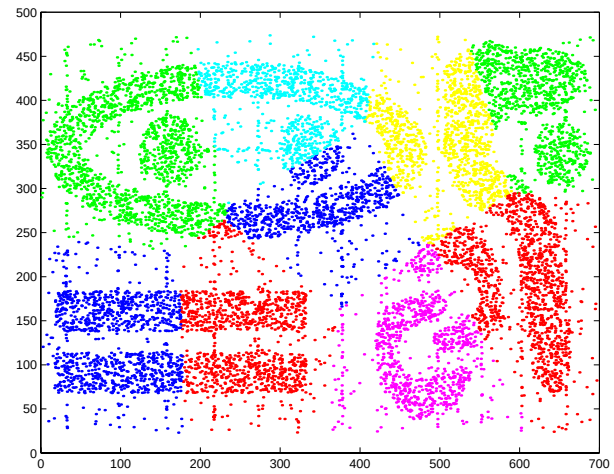


Figure 2.3: K_means' clustering result on t7.10k.dat. number_of_clusters = 9

**K-modes**

K-modes [Hua98] is the newest reported method in the group of partition methods. It is designed for clustering categorical data. K-modes algorithm first uses a simple matching dissimilarity measure for categorical objects; Dissimilarity between $X$ and $Y$ is defined as the total mismatches of the corresponding attribute of the two objects. Then it defines a *mode* of a set of categorical objects $X = x_1, x_2, \cdots, x_n$ as a vector $Q = [q_1, q_2, \cdots, q_m]$ that minimizes

$$D(X, Q) = \Sigma_{i=1}^{n} d(x_i, Q)$$

where $d(x_i, Q)$ is the dissimilarity between object $x_i$ and the mode $Q$. Here the mode $Q$ corresponds to *mean* in the k-means method. The k-modes method then uses a frequency-based algorithm to update modes in the clustering process to minimize the clustering cost function. In doing this, k-modes algorithm extends k-means algorithm to categorical domain. This algorithm also has the similar difficulty in clustering as k-means has: favouring spherical clusters. Due to the time limitation, we did not implement k-modes in this work.

### 2.2.2 Hierarchical Methods

A hierarchical clustering algorithm produces a dendrogram representing the nested grouping relationship among objects. If the clustering hierarchy is formed from bottom up, at beginning each data object is a cluster by itself, then small clusters are merged into bigger clusters at each level of the hierarchy until at the top of the hierarchy all the data objects are in one cluster. This kind of hierarchical methods are called agglomerative hierarchical methods. On the other hand, if the clustering hierarchy is formed from top down, at beginning all the data objects are in one cluster, then big cluster is cut into smaller clusters in each level of the hierarchy, until at the bottom level each data object is a cluster by itself. This kind of hierarchical methods is called divisive hierarchical clustering methods.

There are many new hierarchical algorithms that have appeared in the past few years. The major difference between all these hierarchical algorithms is how to measure the similarity between each pair of clusters.

### BIRCH

BIRCH [ZRL96] introduced the concept of clustering features and the CF-tree. It first partitions objects hierarchically using CF-tree structure. This CF-tree is used as a summarized hierarchical data structure which contains compression of the data that tries to preserve the inherent clustering structure of the data. After the building of the CF-tree, any clustering algorithm can be applied to the leaf nodes of the CF-tree. BIRCH is particularly suitable for large data set, however BIRCH does not perform well if the clusters are not spherical in shape or there are big differences among cluster sizes. In other words, BIRCH made breakthroughs on the efficiency issue, but not on the effectiveness issue of clustering. It has similar clustering results with partitioning methods. For this reason, and also because of the time limitation, we did not implement BIRCH in this work, but it is worthwhile to mention BIRCH here as one important algorithm for clustering large data sets.

### CURE

CURE [SG98] tends to solve two problems of partitioning clustering algorithms: (1) Favour clusters with spherical shapes and similar sizes. (2) Fragile in the presence of outliers. To achieve this, instead of using a single point to represent a cluster in centroid/medoid based methods, CURE uses a set of constant number of representative points to represent a cluster. The constant number of representative points of each cluster are selected so that they are well scattered and then shrunk towards the centroid of the cluster according to a shrinking factor. The CURE authors argued that having more than one representative point per cluster allows CURE to adjust well to the geometry of non-spherical shapes and the shrinking helps to dampen the effects of outliers, thus this set of representative points keeps shape and size information of the cluster. The similarity between two clusters is measured by the similarity of the closest pair of the representative points belonging to different clusters.

Figure 2.4 shows the merging of two clusters in CURE. Representing points of each cluster are shrunk. The distance between each pair of clusters is mea-

sured by the closest distance between representing points from each individual cluster. Clusters with the closest distance are merged into a new cluster. A new set of representing point for this new cluster is then selected to represent the new cluster.

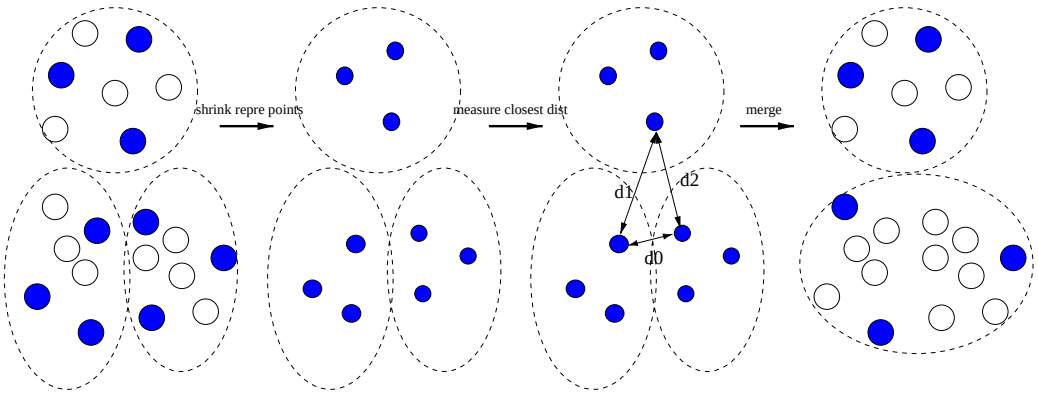


Figure 2.4: CURE's process of merging clusters

With proper parameter selection, CURE partly remedies the problem of favouring clusters with spherical shape and similar sizes, and it is not sensitive to outliers. CURE's using of a set of representative points for each cluster can better represent cluster shape information. However the effectiveness of these representative points highly depends on the the way points are selected, the number of representative points and the shrinking factor. For example, when the shrinking factor $\alpha$ is big, the set of representative points are shrunk too much toward the cluster centre. In this case, the shrunk point set cannot effectively represent the original shape and size of the cluster, and CURE's cluster effectiveness is close to K-means. On the other hand, CURE becomes sensitive to outliers when the shrinking factor becomes small. Also, notice that similar to k-means, CURE also does not consider all the points of a cluster in its clustering process. Compared with partitioning methods, it just improved the way of representing clusters. Through experiments we found that CURE has similar problem as k-means: it still tends to find sphere clusters, although now the problem is not as serious as that of k-means'. It is difficult for CURE to find an elongated cluster. Figure 2.5 shows CURE's clustering result on t7.

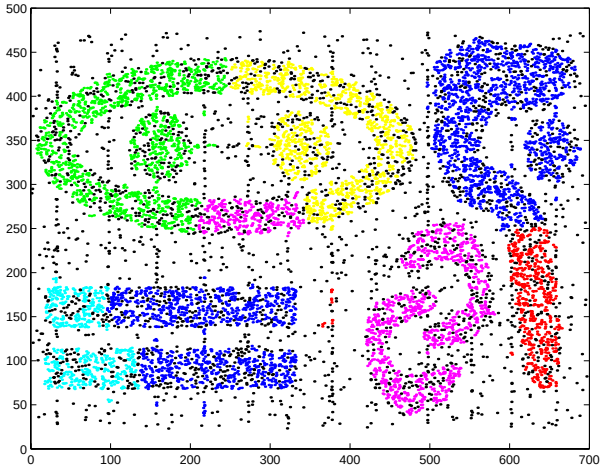


Figure 2.5: CURE's clustering result on t7.10k.dat. cluster_number $= 9, \alpha = 0.3$, numerber_of_representative_points $= 10$

### ROCK

Another interesting method by the same group of CURE authors is ROCK [GRS99]. ROCK operates on a derived similarity graph, so that ROCK is not only suitable for numerical data, but also applicable for categorical data.

ROCK's authors intended to make their method applicable for categorical data. Instead of using *distance* to measure similarity between data points (actually there is no proper *distance* definition for categorical data), ROCK proposed a novel concept of *links* to measure the similarity between a pair of data points. The basic idea of *links* is shown in Figure 2.6.

In the upper figure of Figure 2.6, if we want to decide whether the green point should be in the same cluster with the red point or the blue point just based on the inter-distances of these three points, it is obvious that the green one is closer to the blue one, thus it seems the green point should be in the same cluster with the blue one.

However, if we consider all the other points in the cluster space, the lower picture tells us that globally it is more reasonable that the green point should be with the red point. When we look at the difference between the two pictures, we notice that it is because of the existence of the common neighbour points between the green point and the red point that makes things different. Those common neighbour points are the *links* between the red one and the green one.

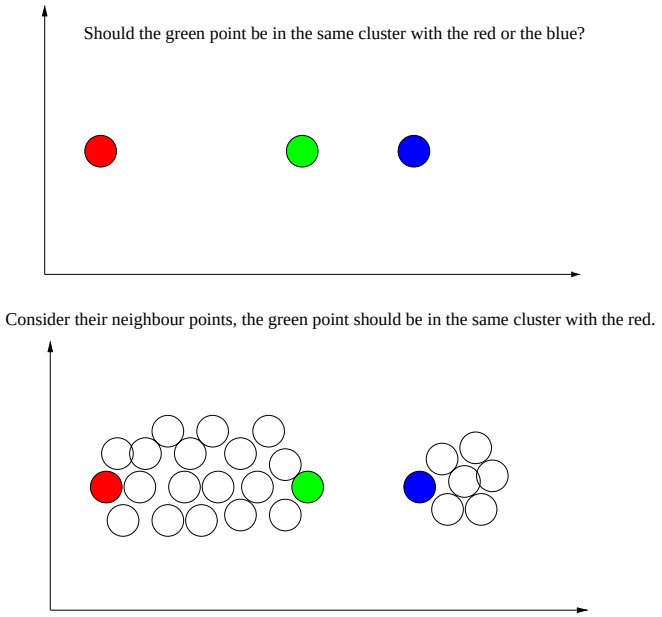


Figure 2.6: ROCK's idea of *links*.

Because the green point has more *links* to the red point than to the blue point, so it should be in the same cluster with the red point. This idea of *links* is the basic idea of ROCK. The concept of *links* uses more global information of cluster space compared with the distance similarity measurement which only consider local distance between two points.

In order to overcome the problem of favouring bigger clusters in the process of clustering, ROCK argues that the similarity between each pair of clusters can be measured by the normalized number of total links between two clusters.

The problem of ROCK is that it is not successful in normalizing cluster links: it uses a fixed global parameter to normalize the total number of links. This fixed parameter actually reflects a fixed modeling of clusters, and it is not suitable for clusters with various densities. ROCK's clustering result is not good for complex clusters with various data density. Also, ROCK is very sensitive to the selection of parameters and sensitive to noise.

ROCK is designed for clustering categorical data, so that its result for clustering this spatial data set is not good. After adjusting parameters for a long time, the best clustering result on t7 we can find is illustrated in Figure 2.7. Notice that we set the number of clusters to be 1000, then among the

resulting 1000 clusters, we got 5 big clusters, all the other 995 are just noise. This is because ROCK does not collect noise in its clustering process. The problem causes difficulty in application: the user cannot know what cluster number he should give ROCK although he may know that there should be 9 clusters in the data set. For this data set, if we set cluster number to be 9, then ROCK's result is as shown in Figure 2.8, where most of the points, 9985 points, are in one cluster, and the other 15 points exists in the 8 noise clusters.

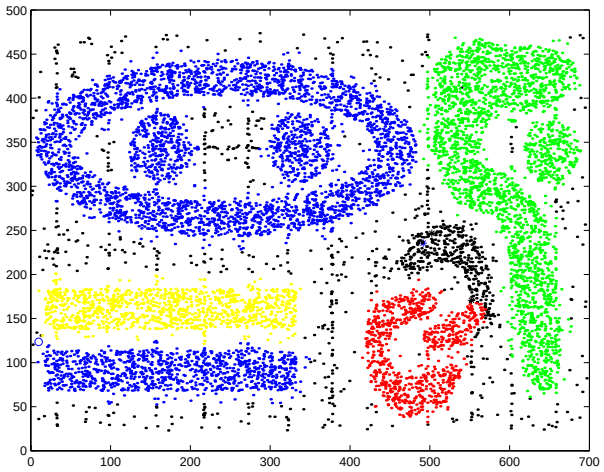


Figure 2.7: ROCK's clustering result on t7.10k.dat. $\theta = 0.975, \text{number\_of\_clusters} = 1000$

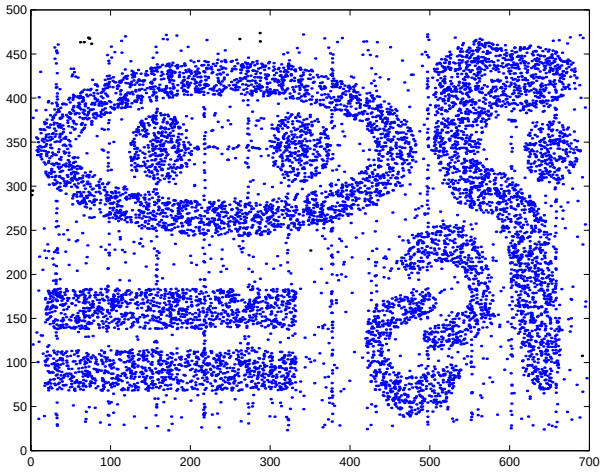


Figure 2.8: ROCK's clustering result on t7.10k.dat. $\theta = 0.975, \text{number\_of\_clusters} = 9$

**CHAMELEON**

A very recent method on clustering is CHAMELEON [KHK99]. The authors of CHAMELEON claimed that the primary reason why all the previous methods failed in successfully finding clusters with diverse shapes, densities or sizes is because they all use some static models. For example, DBSCAN assumes that all points within a cluster are density reachable with pre-set density parameters. ROCK measures the similarity of two clusters by using normalized interconnectivity which is aggregate inter-connectivity against a static inter-connectivity model. CHAMELEON performs clustering through dynamic modeling: two clusters are merged only if the inter-connectivity and closeness between two clusters are comparable to the internal inter-connectivity and closeness within the clusters. CHAMELEON also operates on a derived similarity graph, so that this algorithm can be applied to both numerical data and categorical data. Figure 2.9 shows an overview of the CHAMELEON approach.

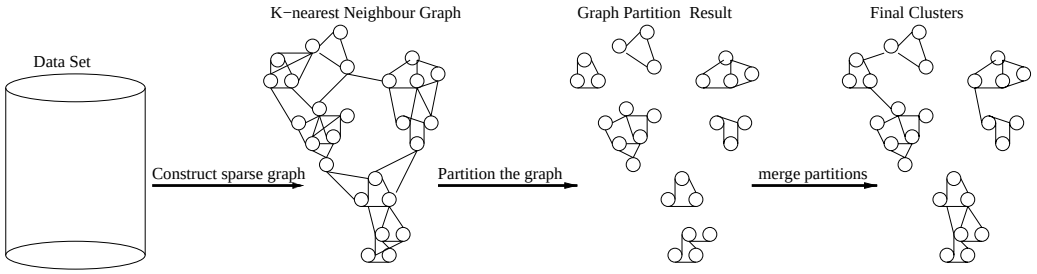


Figure 2.9: An overview of CHAMELEON

CHAMELEON operates on a sparse graph in which nodes represent data items, and weighted edges represent similarities among the data items. The sparse graph is formed by keeping *k-nearest neighbour* of each node. The two-phase clustering process is based on this k-nearest neighbour graph. In phase 1 of the algorithm, it uses the graph partition method to pre-cluster objects into a set of small clusters. Phase 2 of the algorithm merges these small clusters based on their relative interconnectivity and relative closeness.

CHAMELEON has been found to be very effective in clustering. Major

shortcomings of this algorithm are: It cannot handle outliers. Another problem in implementation is that it needs proper setting of several parameters in order to work effectively. The major parameters and their effect on clustering follows:

- **k** is the number of nearest neighbours recorded for each point. When k is bigger, the memory and computational complexity of the program will rise; when k is too small, the k-nearest neighbour graph becomes too sparse, and the process of merging may stop in middle because the relative connectivity becomes infinity.

- **MINSIZE** is used in graph partition to decide the stopping size of graph partition. When MINSIZE is small, phase 1 will produce a set of large number of small partitions. This will cause the merging phase start with large number of partitions, and thus the merging process will be longer. Since computational complexity of phase 2 is $O(n^2)$, while computational complexity of phase 1 is only $O(nlog(n))$, for speed consideration, MINSIZE should not be too small. However, MINSIZE cannot be too large; when MINSIZE is too large, the clustering precision in phase 2 will decrease.

- $\alpha$ for adjusting relative closeness and relative connection's weight in merging goodness function. In merging clusters, CHAMELEON considers both relative connectivity and relative closeness. The merging goodness function of CHAMELEON is of the form:

$$RI(C_i, C_j) * RC(C_i, C_j)^{\alpha},$$

  where $RI$ is the relative interconnection, and $RC$ is relative closeness. $\alpha$ is used to decide how much weight to give to relative closeness and how much to give to relative connection. In clustering 2 dimensional spatial data, we feel that relative closeness should be given more weight.

CHAMELEON's result on test data set t7 is shown in Figure 2.10. This result is very close to the result in the CHAMELEON paper. We see that

CHAMELEON does not collect noises: all the noise points are included inside neighbour clusters.

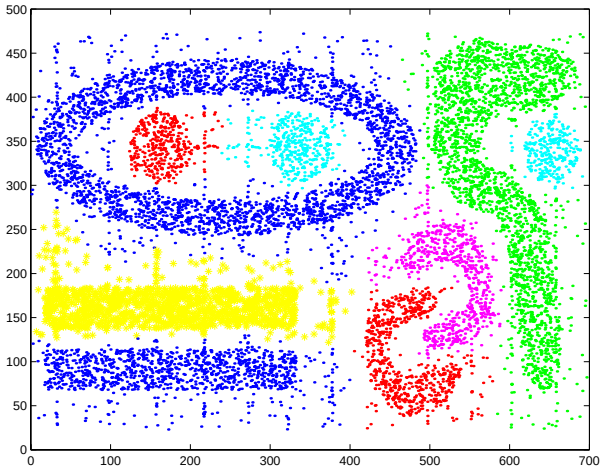


Figure 2.10: CHAMELEON's clustering result on t7.10k.dat. $k = 10$, MinSize $= 2.5\%$, cluster_number $= 9$

The group of hierarchical clustering algorithms is relatively new. The common disadvantage of hierarchical clustering algorithms is setting a termination condition which requires some domain knowledge for parameter setting. The problem of parameter setting prevents clustering algorithms from real application. Also typically, hierarchical clustering algorithms have high computational complexity.

### 2.2.3 Density-based Methods

The advantages of density-based methods are that they can discover clusters with arbitrary shapes and they do not need to pre-set the number of clusters.

#### DBSCAN

DBSCAN [EKSX96] is the most famous density-based method which connects regions with sufficiently high density into clusters. Each cluster is a maximum set of density-connected points.

DBSCAN's density definition is like this: For each object of a cluster, the neighbourhood of a given radius ($\epsilon$) has to contain at least a minimum

number of points ($MinPts$). The density definition of DBSCAN is intuitive. In practice it works very well in spatial clustering. The problem of DBSCAN is that it is very sensitive to the selection of $\epsilon$ and $MinPts$, also it cannot identify clusters with different densities. A simple example of DBSCAN is shown in Figure 2.11. Notice in this example the discovered cluster is of a non-spherical shape, and DBSCAN has no difficulty in finding this cluster and identifying other points as outliers.

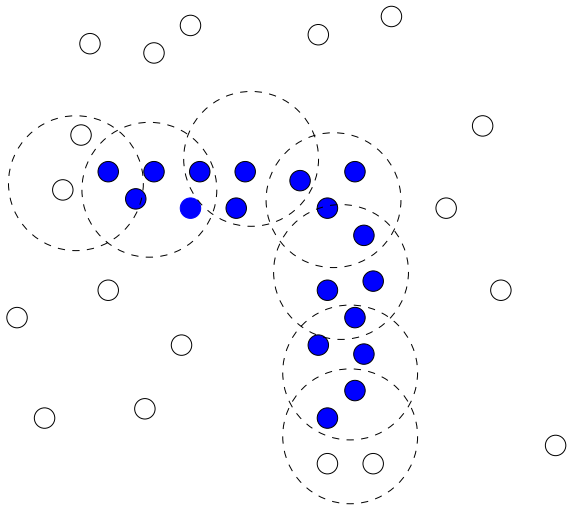

Figure 2.11: A enlonged cluster discovered by DBSCAN.

When we apply DBSCAN to the testing data set t7, it gives very good results as illustrated in Figure 2.12. We feel that the only problem of DBSCAN is that it is very sensitive to the two parameters $\epsilon$ and $MinPts$. For example, if we change $\epsilon$ a little bit from 5.9 to 5.5, then it gives a bad result (see Figure 2.13). If we increase $\epsilon$, the noise will create bridges that cause genuine clusters to merge.

By the same authors, OPTICS [MMKJ99] is an extension to DBSCAN. Instead of producing one set of clustering results with one pre-setting radius ($\epsilon$), OPTICS produces an augmented ordering of the database representing its density-based clustering structure. This cluster-ordering actually contains the information about every clustering level of the data set (up to a "generating distance"), and is very clear for further analysis. Restrictions of OPTICS are

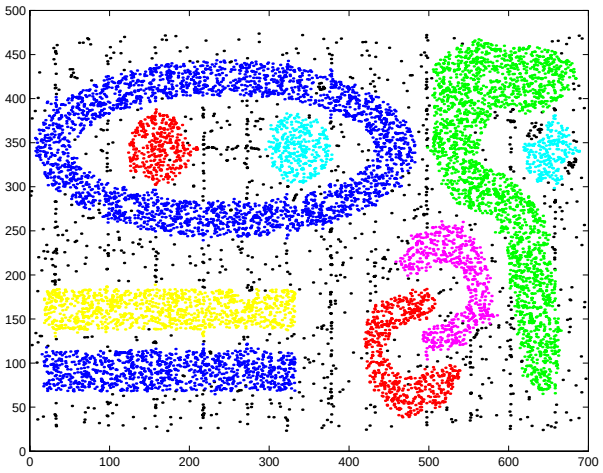


Figure 2.12: DBscan's clustering result on t7.10k.dat. $\epsilon = 5.9, \text{MinPts} = 4$

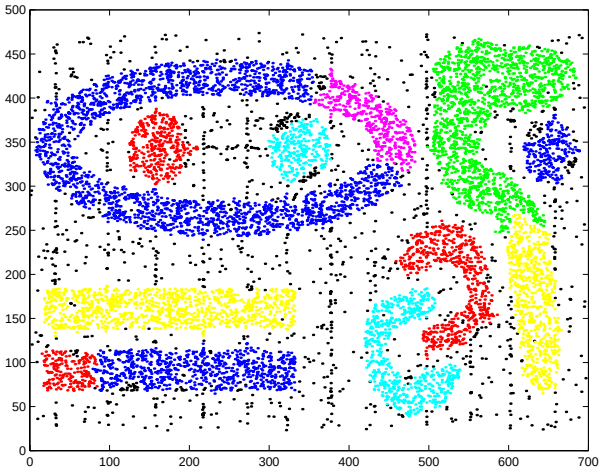


Figure 2.13: DBscan's clustering result on t7.10k.dat. $\epsilon = 5.5, \text{MinPts} = 4$

that it is still more suitable for numerical data, and also the user still needs to set one parameter, $MinPts$.

With proper parameter setting, DBSCAN and OPTICS are experimentally very effective for spatial data clustering, and they also have very good computation and memory efficiency. It is safe to say that DBSCAN/OPTICS is one of the best algorithms for spatial data clustering. Other density-based clustering algorithms include DENCLUE, which clusters based on density distribution functions.

### 2.2.4 Grid-based Methods

Grid-based methods first quantize the clustering space into a finite number of cells, and then perform clustering on the gridded cells. The main advantage of grid-based methods is that their speed only depends on the resolution of griding, but not on the size of the data set. Grid-based methods are more suitable for high density data sets with a huge number of data objects in limited space.

Representative grid-based algorithms include STING [WJR97], CLIQUE [AGGR98] and WaveCluster [SCZ98]. This group of algorithms are mostly new. WaveCluster is very important representative in this group of algorithms. It seems that it has most of the good properties for a clustering algorithm. For this group of clustering methods, we implemented WaveCluster and CLIQUE.

#### WaveCluster

WaveCluster is a novel clustering approach based on wavelet transforms. It is a data clustering method in the spatial data mining problem. WaveCluster first summarizes the data by applying a multi-resolution grid structure on the data space, then the original multi-dimensional data space is considered as multidimensional signals and signal processing techniques - wavelet transform is applied to convert the spatial data into the frequency domain. After wavelet transform, the natural clusters in the data become distinguishable in a transformed frequency space. Dense regions, i.e. clusters, are easily captured in the frequency domain.

WaveCluster's wavelet transform process for one resolution is shown in Figure 2.14, where $HP$ is high pass digital filter and $LP$ is low pass digital filter. The 2-dimensional space is first convolved along the horizontal dimension and down-sampled by 2, resulting in two images $L$ and $H$. Both $L$ and $H$ are then convolved along the vertical dimension and down-sampled by 2, resulting in four sub-images: $LL$, $LH$, $HL$, $HH$. After this process, $LL$ has an average signal for clustering in 2 dimensional signal space. Clustering in $LL$ is simply connecting significant signal cells in $LL$ to form a cluster. From the point of clustering, in this signal processing process we only care the $LL$ signal. However from the point of digital signal processing, all the four signals form a signal set which lose no information of original signal, and this set of four signals can be used for recovery of original signal.

The process of cluster is finding connected strong signal components in $LL$, and because Wavelet transform can filter out noise points, clustering in $LL$ is usually much simpler than clustering in original 2-dimensional space.

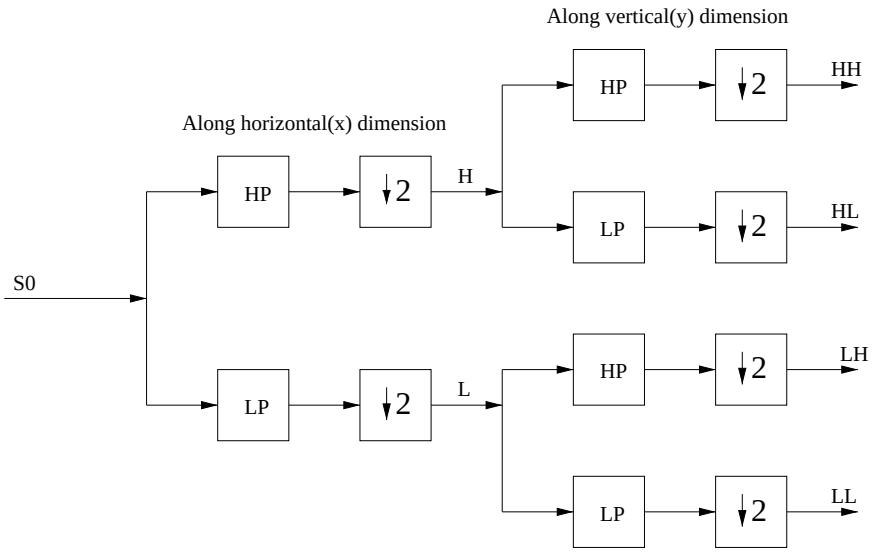


Figure 2.14: WaveCluster Transform Process

In the process of WaveCluster, there are two main parameters to be selected: one is the grid resolution; the other one is the signal threshold $\tau$ for deciding whether a cell is a significant cell in $LL$. This parameter is used for clustering on $LL$. Because of these two parameters, WaveCluster is still not

really an unsupervised clustering method. However, it is worthwhile to point out that usually the signal threshold $\tau$ on the transformed frequency domain is easy to select because clusters become distinguished on frequency space. Often, the wide range of $\tau$ values result in same cluster result. For the other parameter of resolution, the WaveCluster's authors argued that we should try different resolutions, and cluster results on different resolutions gives different views of the clustering space.

Experimental results on Wavelet transform are shown by a test on one data set from WaveCluster's authors. The testing data set DS4 is shown in Figure 2.15. This is one of the data sets used in the original WaveCluster paper. After wavelet transform, the original data space was decomposed into four signals: an average signal ($LL$) Figure 2.16 and three detail signals ($LH$ Figure 2.18 , $HL$ Figure 2.19, and $HH$ Figure 2.17). Here $LH$ emphasizes the horizontal image features, $HL$ the vertical features, and $HH$ the diagonal features. When we apply wavelet transform on t7, the result $LL$ signal is shown in Figure 2.20 and Figure 2.21. Notice that the clusters become distinctive on the $LL$ signal. Thus we see wavelet transform can filter out the noise in the original data. It becomes much easier for clustering upon the transformed $LL$ signal.

WaveCluster's clustering result on t7 is shown in Figure 2.22. Notice that WaveCluster can not separate the two clusters connected by a "bridge". This is because in $LL$, the bridge connecting the two clusters are still very strong signal, thus if we want to separate the two clusters, we also should cut other genuine clusters. Figure 2.23 shows another cluster result of WaveCluster by adjusting signal threshold $\tau$. Now it can separate the bridge-connected two clusters, but meanwhile it breaks other genuine clusters.

### CLIQUE

CLIQUE is specifically designed for finding subspace clusters in high dimensional data. This method is suitable for finding clusters in very sparse high dimensional space. Because of time limitation, we did not test CLIQUE with a high dimensional data. The experiment we did applied 2 dimensional data on CLIQUE.

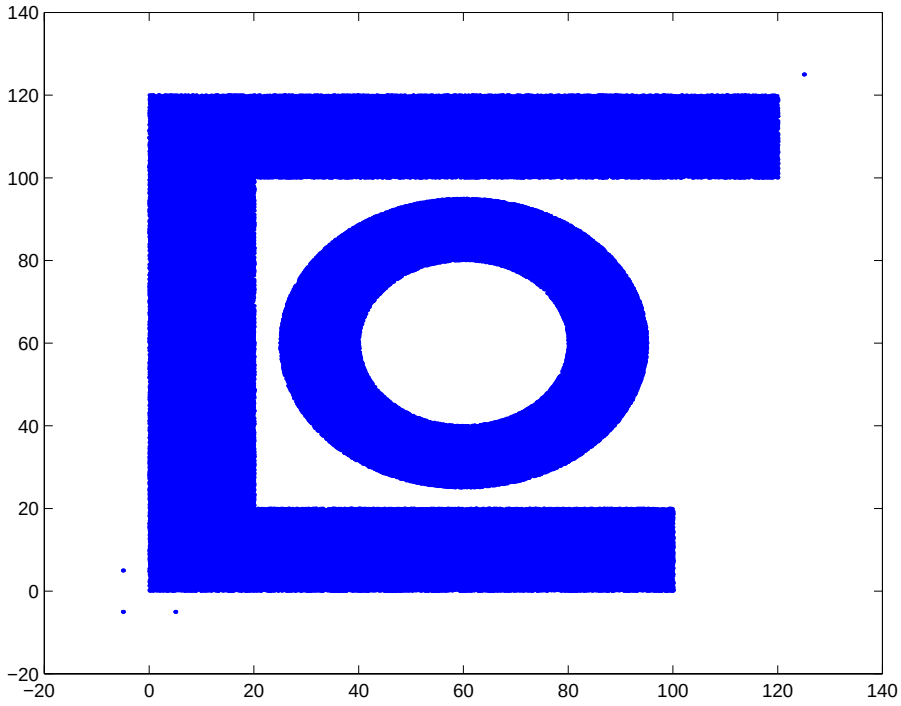


Figure 2.15: testing data set DS4

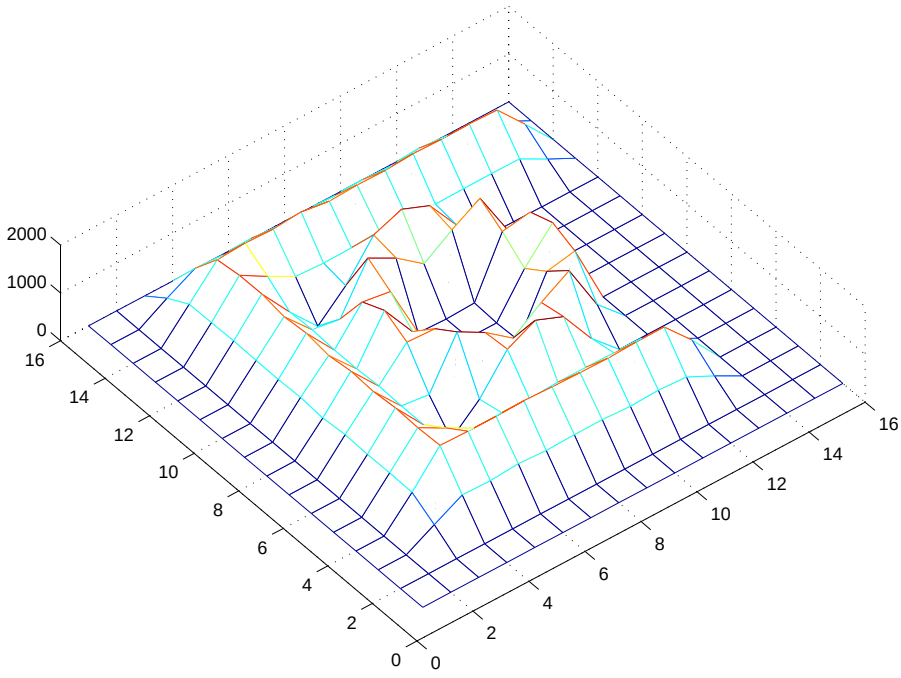


Figure 2.16: LL of DS4 after wavelet transform

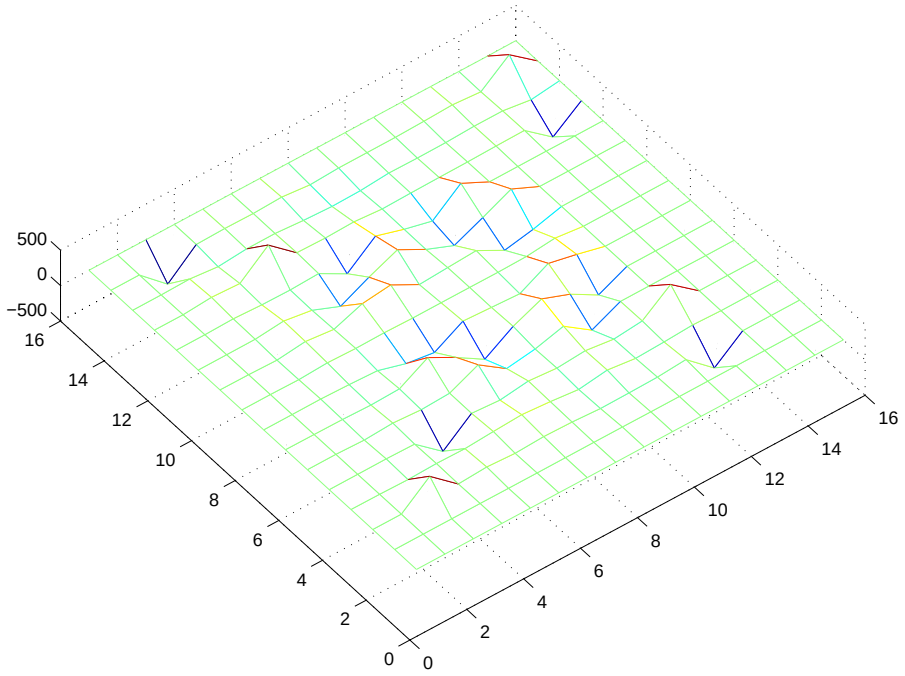


Figure 2.17: HH of DS4 after wavelet transform

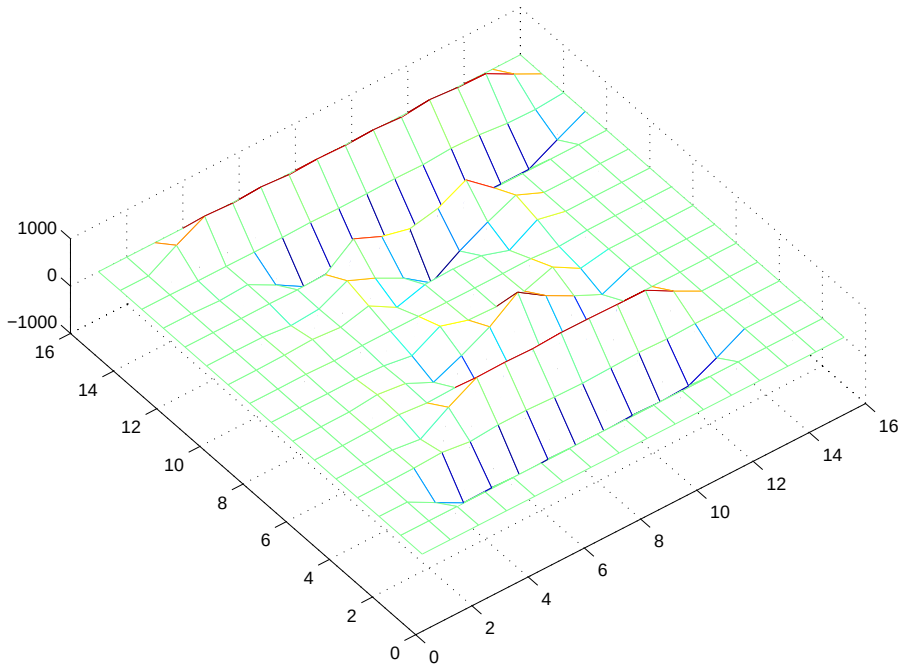


Figure 2.18: LH of DS4 after wavelet transform

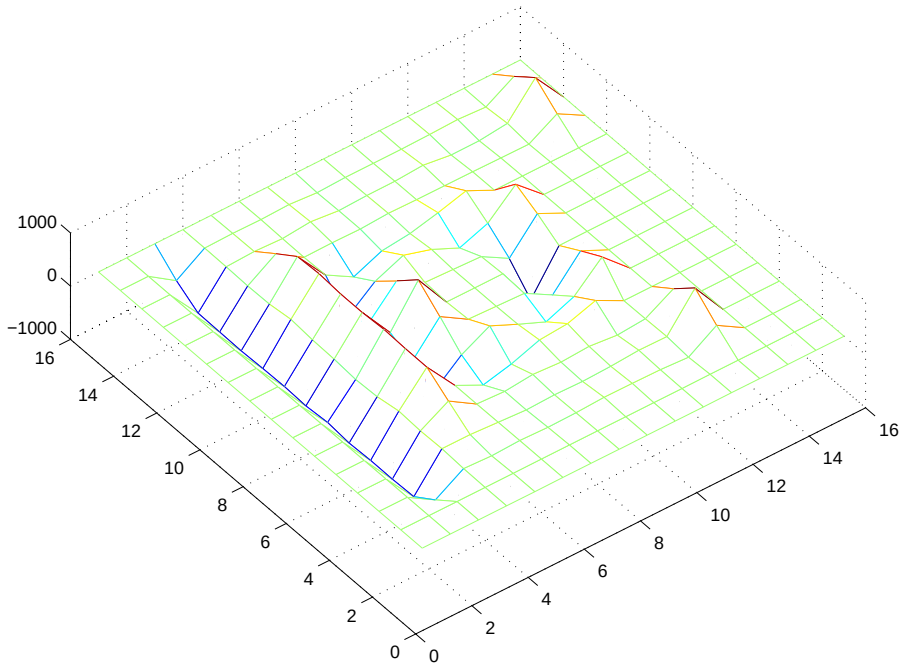


Figure 2.19: HL of DS4 after wavelet transform

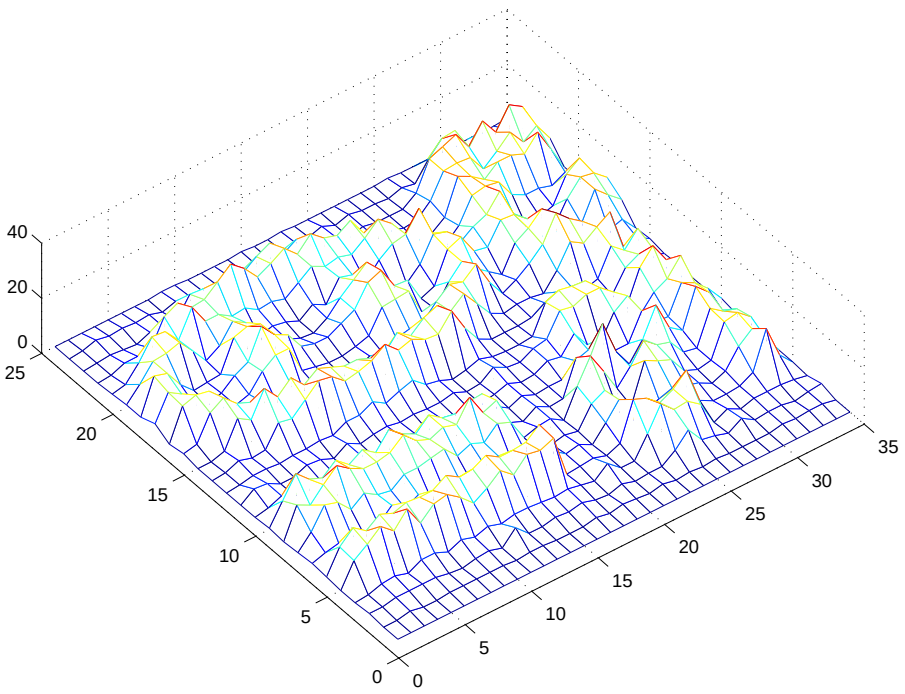


Figure 2.20: LL of t7 after wavelet transform (1)

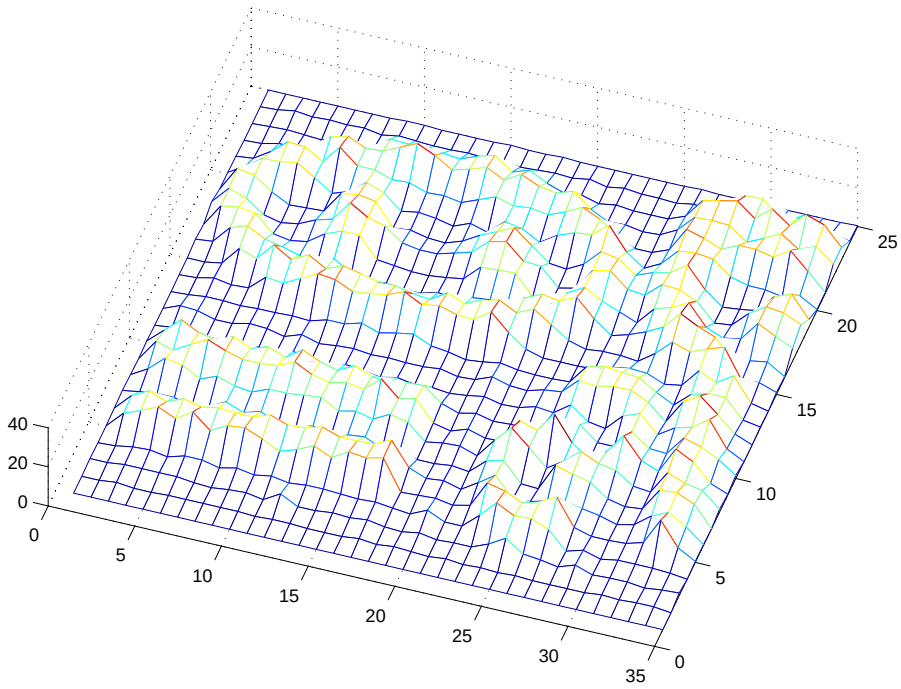


Figure 2.21: LL of t7 after wavelet transform (2)

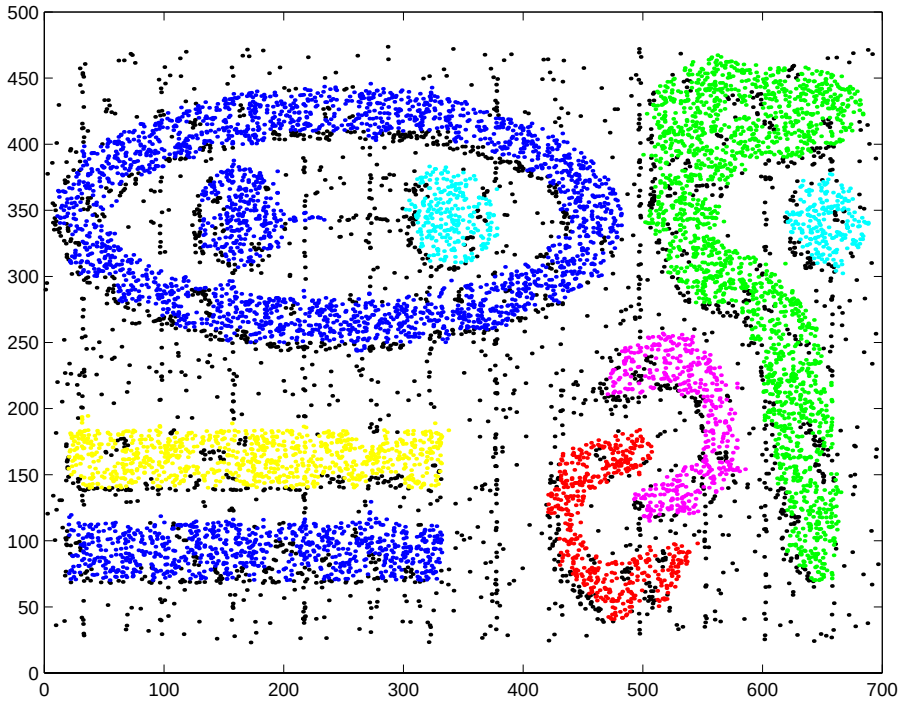


Figure 2.22: WaveCluster's result on t7 when $resolution = 5$ and $\tau = 1.5$

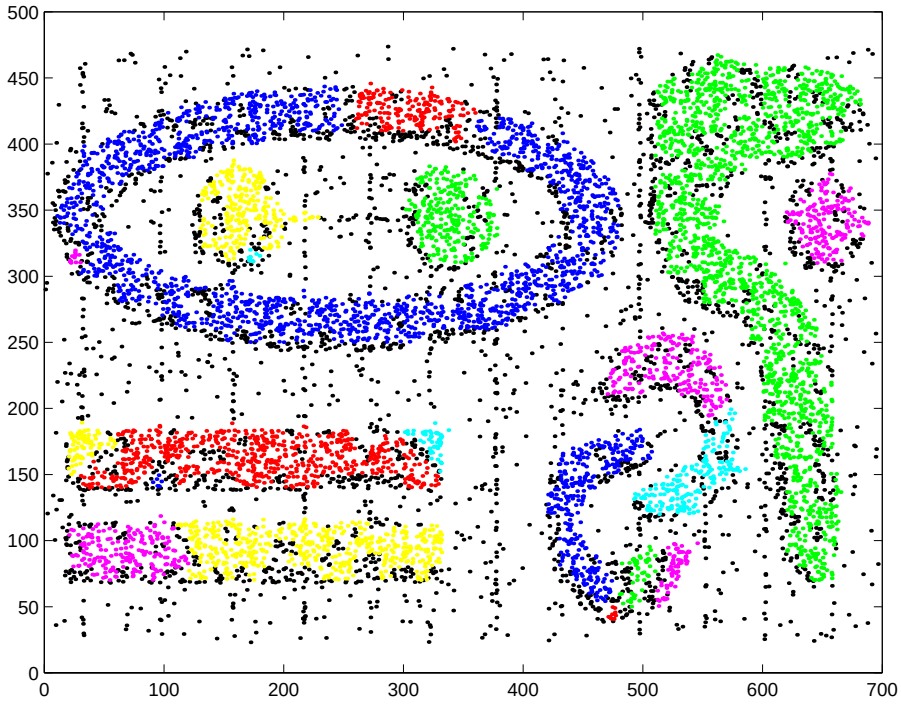


Figure 2.23: Cluster result of WaveCluster on t7 when $resolution = 5$ and $\tau = 1.999397$ signal threshold

CLIQUE's clustering process starts with the lower dimensional space. When clustering for k-dimensional space, CLIQUE makes use of information of (k-1)-dimension which is already available. CLIQUE's idea is shown in Figure 2.24: if project the 2-dimensional data to 1-dimensional axis, there are only 2 cells in horizontal axis and 3 cells in vertical axis are dense. So, when we check potential dense 2-dimensional data cells, we just need to check the intersection of the horizontal dense area and vertical dense area, instead of checking all the 2-dimensional space. From this example we see that using information of 1-dimensional dense cells, we do not need to study most of the cells in clustering of 2-dimensional space. This idea saves a lot effort in high dimensional data space.

However, it is worthwhile to point out that CLIQUE is only suitable for very sparse clustering space. In those cases, CLIQUE saved significant searching time. But, CLIQUE is not suitable for heavy noised clustering space or dense clustering space. Figure 2.25 gives a counter example where we see CLIQUE saves nothing in identifying potential dense cells in 2-dimensional

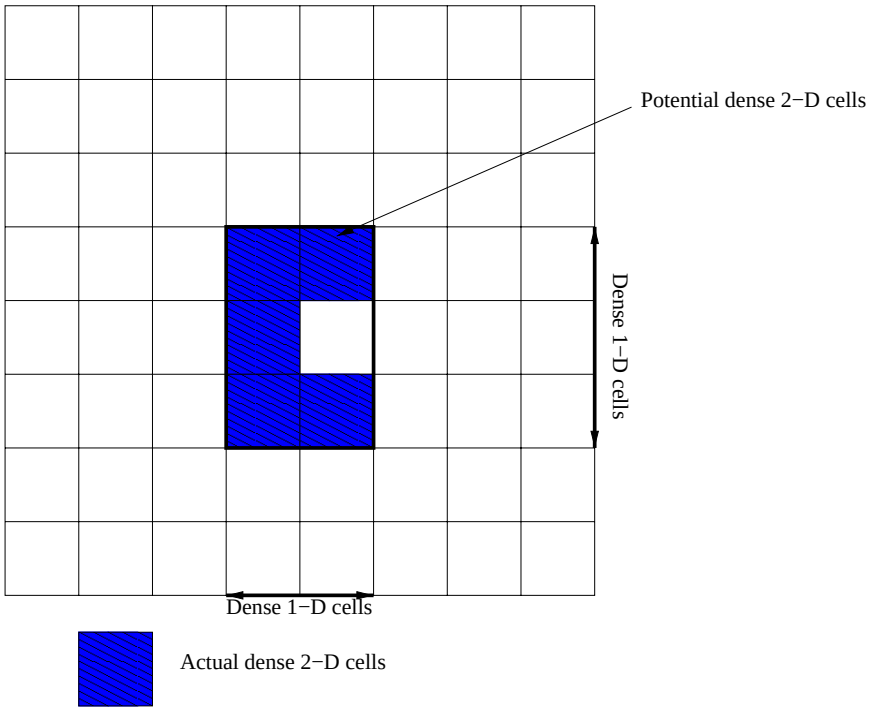


Figure 2.24: Determine potential 2-dimensional dense cells using 1-dimensional dense region information

space: in this figure, when 2-dimensional cells are projected to 1-dimensional axis, all the 1-dimensional cells are dense. If we use information of dense 1-dimensional cells to decide potential dense cells in 2-dimensional space, all the 2-dimensional cells are potentially dense, and CLIQUE's process saved nothing in finding dense 2-dimensional cells.

CLIQUE's clustering result on t7 is shown in Figure 2.26. From experiments we feel that CLIQUE is not very good in clustering 2 dimensional data. Its difficulties include: (1). Sensitive to noise. In heavily noised data sets, it cannot use high resolution, otherwise it tends to merge genuine clusters together. (2). If the resolution is not high enough, some "edge" points of clusters are deemed as outliers.

### 2.2.5 Turn, Turn* and Other

Besides these four groups of clustering methods, it is worthwhile to mention some other very recently developed methods.

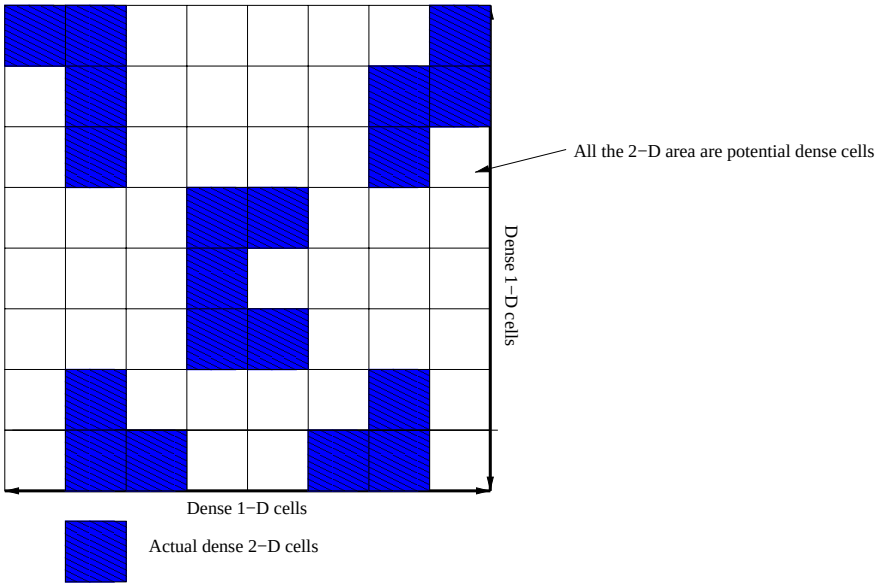


Figure 2.25: A counter example for CLIQUE

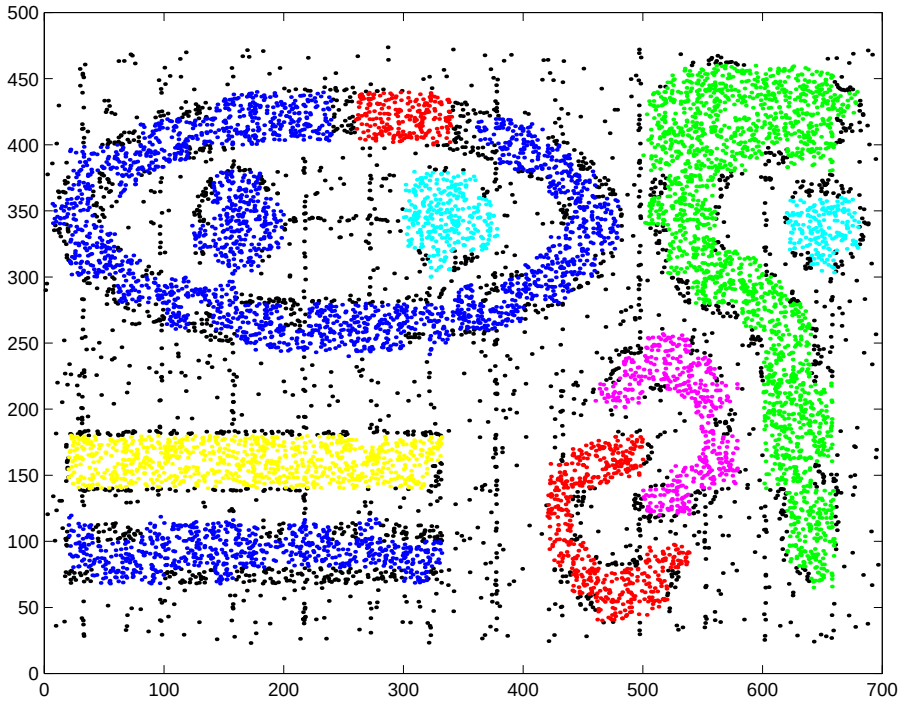


Figure 2.26: CLIQUE result on t7.10k.dat when $threshold = 0.18$ and $resolution = 20$

TURN [FWZ01] is a non-parametric clustering approach which is free of parameters in application. One common problem of all the above algorithms is that: they require some input parameters or heuristics that are usually unknown at the time they are needed. This is a key problem which prevents clustering techniques from many real applications. Aimed at this point, TURN [FWZ01] is a new algorithm without any parameters. The basic idea of TURN is to find the "*turning*" point between clusters which represent natural boundary between clusters. If that is found, then the data points can be easily assigned to the appropriate clusters. If X and Y are areas of high density of data points, then somewhere between them the data point density will decline to a minimum and start rising again. This is the natural cluster boundary and is an example of a "*turning*" point in the distribution.

TURN proceeds by iteratively selecting a non-assigned object $p$ and computes the similarity between $p$ and all other non-assigned objects. After sorting the similarities and differentiating them 3 times, the algorithm looks for a change of sign (i.e. turning point) in the differentiated numbers. All objects appear before the turning point are assigned to the same cluster as $p$ and their direct neighbours are also pulled in. The process repeats until all objects are assigned. TURN is suitable for clustering categorical data, and experimentally it outperformed ROCK in clustering web usage data [FWZ01].

Another improved new algorithm TURN* [ZFW02] is designed for spatial data clustering. TURN* first clusters spatial data at multi-resolution. In each resolution, it classifies the data points into "*internal*" points and "*border*" points. The process of clustering is simply put all the neighbouring internal points together into one cluster. After this, TURN* uses TURN for the selection of optimal clustering resolution based on the clustering statistics on each resolution.

So far some very impressive results show that it is better in speed, scalability and output quality than most available spatial clustering algorithm.

Another very recent method [HK01] developed by Harel and Koren is based on the deterministic analysis of random walks on a weighted similarity graph generated from the data. The main concept presented in the work is "*separat-*

*ing operator*", which is applied to the graph iteratively to decrease the weights of inter-cluster edges and to increase the weights of intra-cluster edges, thus "*sharpen*" the distinction between the two kinds of edges and finally find the natural boundaries between the clusters. This method seems to own most of the characteristics of a good clustering algorithm: unsupervised clustering process, applicable to both numerical data and categorical data, not sensitive to noise and can find clusters of arbitrary shape.

## 2.3 Clustering Effectiveness Comparison

This section summarizes the clustering effectiveness properties of each algorithm implemented. The properties of all the above discussed algorithms are listed in Table 2.1 - 2.3.

| **property** | **K-means** | **K-modes** | **DBScan** |
|---|---|---|---|
| suitable for Huge data set | Yes | Yes | Yes |
| attributes | metric | non-metric | metric/non-metric |
| can cluster arbitrary shape | No | No | Yes |
| Parameters | cluster NO | cluster NO | $\epsilon$, $MinPts$ |
| Not sensitive to noise | Yes | Yes | Yes |
| Insensitive to order of input records | Yes | Yes | Yes |
| handle high dimensionality | Yes | Yes | Yes |

Table 2.1: Clustering Properties of Partition Algorithms and Density-Based Algorithms

Here is the explanation on each clustering property:

- **Suitable for Huge data set.** We think that ROCK is not suitable for huge data set. This is because ROCK's clustering memory complexity

| property | CURE | ROCK | CHAMELEON | TURN |
|---|---|---|---|---|
| suitable for Huge data set | Yes | No | Yes | Yes |
| attributes | metric | non-metric | metric/non-metric | non-metric |
| can cluster arbitrary shape | Not good | Yes | Yes | Yes |
| Parameters | cluster NO, shrinking factor, number of representative points. | cluster NO, similarity threshold. | cluster NO, MINSIZE, $\alpha$, k_nearest. | |
| Not sensitive to noise | Yes | No | Yes | Yes |
| Insensitive to order of input records | Yes | Yes | Yes | Yes |
| handle high dimensionality | No | Yes | Yes | Yes |

Table 2.2: Clustering Properties of Hierarchical Algorithms and TURN

is very high (almost $O(n^3)$). This prevents ROCK from being directly applied onto huge data set. ROCK's authors suggest to cluster on a sample set, and then use the clustering result to label all the other data points.

- **cluster attribute.** ROCK, CHAMELEON and TURN are suitable for non-metric data. The reason behind this is that they are based on a similarity graph of original data, but not on the spatial property and the clustering space. K-modes is also suitable for categorical data because its similarity measurement is designed for categorical vector space. Although DBSCAN and WaveCluster can also be applied to some cases of categorical data, that is only in case those categorical attributes can be represented using numerical data.

- **can cluster arbitrary shape.** This is a difficulty of partitioning algorithm, and CURE is also not good on this property. Most of the recent algorithms can handle this problem.

| **property** | **WaveCluster** | **Clique** | **TURN*** |
|---|---|---|---|
| suitable for Huge data set | Yes | Yes | Yes |
| attributes | metric | metric | metric |
| can cluster arbitrary shape | Yes | Yes | Yes |
| Parameters | resolution $\tau$ | resolution threshold | |
| Not sensitive to noise | Yes | No | Yes |
| Insensitive to order of input records | Yes | Yes | Yes |
| handle high dimensionality | No | Yes | Yes |

Table 2.3: Clustering Properties of Grid-Based Algorithms

- **parameters.** We see that all the algorithms except TURN and TURN* have certain input parameters. It is worthwhile to say that WaveCluster is close to non-parameter clustering: although it need input parameter for density threshold $\tau$, in the wavelet transformed space clusters are distinctive, selecting of $\tau$ is easy, and WaveCluster is not sensitive to this parameter. As to the other parameter for resolution, the original WaveCluster paper claims that this method is for multi-resolution clustering, so resolution is not a parameter for the method.

- **Not sensitive to noise.** ROCK is very sensitive to noise. CLIQUE also cannot handle noise very well: in heavily noised space, it cannot use high resolution, otherwise noise will form bridges to connect genuine clusters together. CHAMELEON is not sensitive to noise, but it also cannot collect noise points; noise points in CHAMELEON are merged into neighbour clusters.

- **Insensitive to the order of input records.** All the tested algorithms are good on this property.

- **Ability to handle high dimensionality.** For K_means, as long as the distance between two data points can be defined, it can be extended to high dimensional data. For ROCK, CHAMELEON and TURN, the clustering algorithms do not care whether it is high dimensional data. All they need is just a similarity graph. So once proper similarity measurement for high dimensional data is defined, they are applicable. CLIQUE is specifically designed for high dimensional data. It is suitable for finding subspace clusters in sparse high dimensional data. DBScan can use R*-tree or SR-tree for high dimensional data. CURE and WaveCluster can not be directly applied to high dimensional data because their clustering processes directly depend on the spatial property of the data space.

## 2.4 Clustering Efficiency Comparison

The computational complexity and space complexity of each algorithm is summarized in Table 2.4. From the table we see that hierarchical algorithms typically have big computation and space complexity, while grid-based methods have low computation and space complexity.

We recorded the clustering time and memory required for each implemented algorithm. Although this may somehow relate to the way of programming, still the results can reflect characteristics of each algorithm. The program of k-means, CURE, DBSCAN, CHAMELEON, WaveCluster and CLIQUE are tested on a 800MHz Pentium III with 256M memory. Because of high requirement of memory, the ROCK code is tested on a 1.5GHz Pentium IV with 1G memory. The experimental results are shown in Table 2.5. Our implementation experimental results in Table 2.5 are consistent with the analysis results in Table 2.4. Notice that although k-means, WaveCluster and CLIQUE are all $O(n)$ in computational complexity, k-means' experimental clustering time is obviously higher than the two grid-based methods. This is because grid-based methods just scan the $n$ input data objects once, and put these data into a matrix of cells. The real clustering process just need to handle these data

| **Algorithm** | **Computational Complexity** | **Space Complexity** |
|---|---|---|
| K-means | $O(n)$ | $O(n)$ |
| CURE | $O(n^2)$ for low dimention<br>$O(n^2\ log\ n)$ for high dimention | $O(n)$ |
| ROCK | $O(n^2 + nm_m m_a + n^2 logn)$ | $min(O(n^2), nm_m m_a)$<br>$m_m$ for maximum number of neighbours<br>$m_a$ for average number of neighbours |
| DBSCAN | $O(nlogn)$ | $O(n)$ |
| CHAMELEON | $O(nm + nlogn + m^2 logm)$<br>$m$ for number of sub-graph after phase#1. | $O(kn)$,<br>$k$ for $k$-nearest graph |
| WaveCluster | $O(n)$ | $O(n)$ |
| CLIQUE | $O(n)$ | $O(n)$ |
| TURN | $O(n^2)$ | $O(n)$ |
| TURN* | $O(n)$ | $O(n)$ |

Table 2.4: Computational complexity and space complexity of algorithms

cells, the number of which is usually much less than the number of original data objects.

| **Algorithm** | **Clustering time (second)** | **Memory Size (byte)** |
|---|---|---|
| K-means | 8.44 | 5.5M |
| CURE | 155.59 | 4.6M |
| ROCK | 526.19 | 1.145G |
| DBSCAN | 5.02(phase1)+5.51(phase2) | 1.4M |
| CHAMELEON | 1667.86 | 8.6M |
| WaveCluster | 0.82 | 0.8M |
| CLIQUE | 0.70 | 0.7M |

Table 2.5: Clustering Speed and Memory Size Results upon a data set with 10,000 data points

Comments for each algorithm's efficiency is following:

- **K-means.** Techniquely, k-means is the most simple one, and it does not need complex structure to store data. So k-means has very good speed and memory efficiency, but it is not a good clustering algorithm considering its poor effectiveness.

- **CURE.** CURE is the quickest among the three hierarchical clustering algorithms, and it has the smallest memory space. This is because it has relatively simple similarity measurement and simple data structure when compared with ROCK and CHAMELEON.

- **ROCK.** ROCK is slow, and its memory requirement is terrible. The reason behind this is: it does not throw any noise, and each noise point tends to be a cluster by itself; thus in most of cluster process, most of the clusters are not real clusters, but they are there occupying space and CPU time. Also, each cluster needs to keep a ordered heap of all the other clusters according to the merging goodness, thus ROCK's memory requirement is $O(n^2)$. One suggested improvement for ROCK would be: instead of keeping a ordered heap of all the clusters, it can keep only a heap of top $k$ clusters with top goodness.

- **CHAMELEON.** CHAMELEON is slow. This is Typical for a hierarchical clustering algorithm. Because this program is locally developed, some part of it may not be efficient, but still we cannot expect it to be much faster than the other two hierarchical algorithms even after optimization.

- **DBSCAN.** DBSCAN has very good efficiency among the tested algorithms. Because of usage of R*-tree, it is efficient in both memory and speed.

- **WaveCluster.** As a grid-based algorithm, WaveCluster is very efficient in memory and in CPU, because its computational complexity and memory complexity only depend on the resolution selected, while the clustering process has nothing to do with the number of input data points. Among all the tested algorithms, WaveCluster is the one which has top clustering quality, memory and computational complexity.

- **CLIQUE.** CLIQUE has very similar memory and computational complexity with WaveCluster since they are both grid based algorithms.

However, CLIQUE directly cluster upon the data space while WaveCluster cluster in the wavelet-transformed space. Although CLIQUE has very good efficiency, it does not give good cluster quality as WaveCluster does.

# Chapter 3

# Cluster Web Sessions

The previous chapter presents a thorough survey on different known clustering methods. This chapter studies a real application of clustering techniques – clustering web sessions. The difficulty in clustering web sessions lies in that the session data are not numerical. Most of the algorithms we discussed in the last chapter are proven effective in clustering numerical data, however so far there is no effective similarity measurement on session data and no effective real clustering example on web sessions has been reported. This chapter suggests a new session similarity measurement and it clusters session data by using three different clustering techniques.

## 3.1 Introduction to the Problem

The problem of clustering web sessions is part of a larger work of web usage mining. Web usage mining is the application of data mining techniques to discover usage patterns from Web data [SCDT00]. Generally speaking, web usage mining has three main steps: 1) data preprocessing; 2) data mining; 3) mining results visualization. Session cluster discovery is an important part of web data mining.

For a given raw web-log, algorithms have been developed to clean the web log, identify user sessions, and create IDs for web pages. The cleaning involves the removal of entries which contain an error flag, requests for images and other embedded files, applets and other script codes, requests generated by web agents whose function is to pre-fetch pages for caching, requests from

proxies, and requests reset by visitors. etc.

User sessions have been identified using a 25-minute timeout threshold. Web page IDs are constructed based on assigning an ID to each component of the URL so two web page IDs can be compared for similarity or "closeness".

After cleaning the raw web log data, two data files are provided which contain the cleaned data. The format of one data file is like this:

```
01000102 962 945458058
01000102 962 945458060
01000102 483 945458060
01000102 484 945458060
01100001 965 937344265
01100001 963 937340669
01100001 964 937340670
01100001 964 937341439
```

Each column of this file represents **session number**, **page number**, and **time stamp** respectively. Another file has this format:

```
7 : 0060 : /Courses/TECH142/TeachingStaff/index.html
8 : 007 : /Courses/TECH142/index.html
9 : 008 : /Courses/TECH142/side.html
10 : 01 : /Courses/TECH150
11 : 0100 : /Courses/TECH150/CourseDescription/index.html
12 : 0110 : /Courses/TECH150/Evaluation/index.html
```

Each column of this file represents **page number**, **page ID** and **URL** respectively. **page ID** is a unique string used to represent a web page; each letter in this string represents one level of URL path.

Our problem of web session clustering is based on these two session files. The results of clustering can give insight to the user's behaviour in a web site and have significant applications in personalization, recommendation system, adaptive sites, etc.

## 3.2 Survey on clustering web sessions

Most of the studies in the area of web usage mining are very new, and the topic of clustering web sessions has recently become hot in the field of real application of clustering techniques. Shahabi et al. [SZAS97] introduced the idea of Path Feature Space to represent all the navigation paths. Similarity between each two paths in the Path Feature Space is measured by the definition of Path Angle which is actually based on the Cosine similarity between two vectors. In this work, k-means cluster method is utilized to cluster user navigation patterns. Fu et al. [FSS99] cluster users based on clustering web sessions. Their work employed attribute oriented induction to transfer the web session data into a space of generalized sessions, then apply the clustering algorithm BIRCH [ZRL96] to this generalized session space. Their method scaled well over increasing large data. However, problems of BIRCH include that it needs the setting of a similarity threshold and it is sensitive to the order of data input. The paper does not discuss in detail how they measure the closeness between sessions and how they set the similarity threshold which are very important for clustering. Mobasher et al. [MCS99] used clustering on a web log using the Cosine coefficient and a threshold of 0.5. No detail is mentioned of the actual clustering algorithm used as the paper is principally on Association Rule mining. One recent paper which bears some similarity to our work is by Banerjee and Ghosh [BG01]. This paper introduced a new method for measuring similarity between web sessions: The longest common subsequences between two sessions is first found through dynamic programming, then the similarity between two sessions is defined through their relative time spent on the longest common subsequences. Applying this similarity definition, the authors built an abstract similarity graph for the set of sessions to be clustered, then the graph partition method was applied to "cut" the abstract graph into clusters. Our method has a similar basic idea on measuring session similarity – consider each session as a sequence and borrow the idea of sequence alignment to measure similarity between sequences. However we look into more detail of each web page by first defining a similarity between each

two pages, then instead of simply finding the longest common subsequence, our method utilizes dynamic programming to find the "*Best Matching*" between two session sequences. In our method, similarity between sessions are measured through their *Best Matching*.

Other works indirectly related to the topic of web session clustering include: Pitkow et al. [JP99] explored predictive modeling techniques by introducing a statistic Longest Repeating Subsequence model which can be used for modeling and predicting user surfing paths. Spiliopoulou et al. [ML99] built a mining system, WUM, for the discovery of interesting navigation patterns. In their system, interestingness criteria for navigation patterns are dynamically specified by the human expert using WUM's mining language MINT. Mannila and Meek [HC00] presented a method for finding partial orders that describe the ordering relationships between the events in a collection of sequences. Their method can be applied to the discovery of partial orders in the data set of session sequences.

## 3.3 Similarity Measurement for Web Sessions

The first and foremost question needed to be answered in clustering web sessions is how to measure the similarity between two web sessions. A web session is naturally a stream of hyper link clicks. Most of the previous related works applies either Euclidean distance for vector or set similarity measurements, Cosine or Jaccard Coefficient. Shortcomings for doing this is obvious: (1) the transfered space could be of very high dimension; (2) The original click stream is naturally a click sequence which cannot be fully represented by a vector or a set of URL visiting; (3) Euclidean distance has been proven in practice not suitable for measuring similarity in categorical vector space.

Here we propose to consider the original session data as a set of sequences, and apply sequence similarity measurement to measure similarity between sessions. Sequence alignment actually is not a new topic; there exist several algorithms for solving sequence alignment problem [KJD00]. Our method for measuring similarity between session sequences borrows the basic ideas from

these algorithms.

There exist two steps in our definition of session similarity. First we need to define similarity between two web pages because each session includes several web pages; the second step is to define session similarity using page similarity as a inner function.

### 3.3.1 Similarity Between Web Pages

We noticed that there exist similarities between many different web pages. One example is like the following two URLs:

```
URL#1: http://www.cs.ualberta.ca/labs/database/current.html
URL#2: http://www.cs.ualberta.ca/labs/database/publications.html
```

Similarity between these two URLs is obvious: They are very similar pages with similar topic about the research work in the DataBase group of the University of Alberta.

In another example, the similarity between the two URLs is not that obvious, but there definitely exists *some* similarity:

```
URL#1: http://www.cs.ualberta.ca/labs/database/current.html
URL#3: http://www.cs.ualberta.ca/theses/
```

URL#1 is about the current research work in the database lab of the Department of Computing Science at the University of Alberta; URL#3 is about the theses finished in the recent years in the Department of Computing Science with the University of Alberta. We feel that there is some similarity between URL#1 and URL#3, but the similarity is not as strong as the similarity between URL#1 and URL#2. We need a systematic method to give numerical measurement for the similarity between two URLs.

In order to measure the similarity between two web pages, we first represent each level of a URL by a token; the token string of the full path of a URL is thus the concatenation of all the representative tokens of each level. This process corresponds to marking the tree structure of a web site as shown in

Figure 3.1. Notice here we assume that the URL path can fully reflect the content of URL, also we assume that the URL connection is tree structured and there is no loop exists. These assumption can be often true for intranets of certain organizations. For example, the tele-learning company's intranet in our project application.

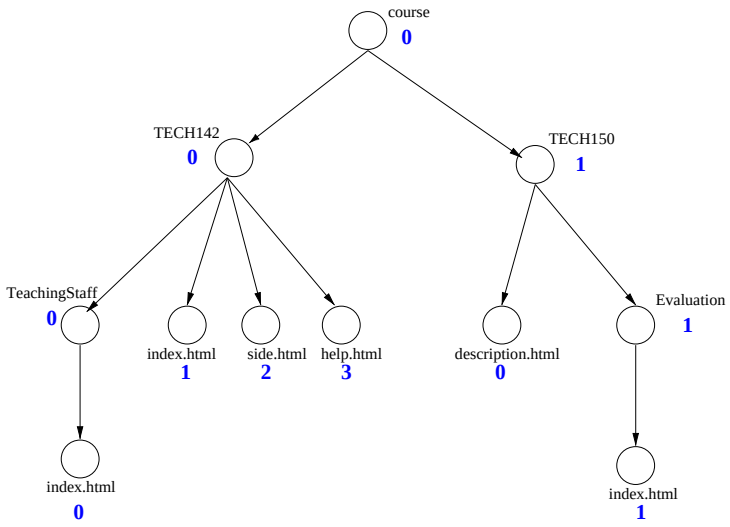


Figure 3.1: Labeling a tree structure of a web site

In Figure 3.1, the web page "/course/TECH142/index.html" is represented by the token string "**001** ", the webpage "/course/TECH150/description.html" is represented by the token string "**010**". The computation of web page similarity is based on comparing the token string of web pages.

Our web similarity computation works in two steps:

- **Step1:** We compare each corresponding token of the two token strings one by one from the beginning, and this process stops at the first pair of tokens which are different. For example, let us compare the web page "/course/TECH142/TeachingStuff/index.html" and the web page "/course/TECH142/side.html". The token string of the first web page is "**0000**", and the second web page's token string is "**001**". Now compare the two token strings in Figure 3.2:

  From Figure 3.2 we see that the two token strings have two same corresponding tokens. Notice that if we compare the token string "**0111**"

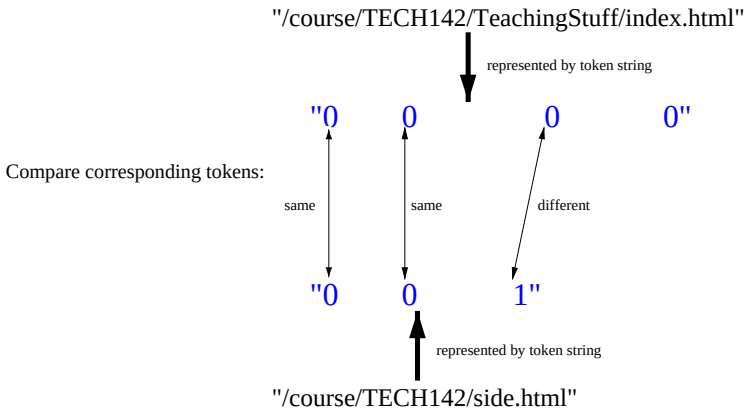


Figure 3.2: Compare token strings

and "**0101**", they have only two same corresponding tokens because the comparing process stops at the first pair of different tokens.

- **Step2:** compute the similarity of two web pages. The similarity between two web pages are computed in this way: suppose the length of the first token string is $length1$, and the length of the second token string is $length2$, select the the longer string's length $longer\_length = (length1 > length2)\ ?\ length1\ :\ length2$, then we give weight to each level of the longer token: the last level is given weight 1, the second to the last level is given weight 2, the third to the last level is given weight 3, and so on and so forth, until the first level which is given weight $longer\_length$. For the given example, the length of token string "**0000** " is 4, and length of the token string "**001** " is 3, thus $longer\_length = 4$, and weight for each level is shown in Figure 3.3

| | | | | |
|---|---|---|---|---|
| Weight of each token: | 4 | 3 | 2 | 1 |
| token string1: | 0 | 0 | 0 | 0 |
| token string2: | 0 | 0 | 1 | |

Figure 3.3: Weight each token level

Next, the similarity between two token strings is defined as the sum of

the weight of those matching tokens divided by the sum of the total weights. For the given example, we see that their similarity is thus:

$$(4+3)/(4+3+2+1) = 0.7$$

For any pair of URLs, using this similarity measurement, the two pages' similarity is between 0.0 and 1.0. If the two pages are totally different, i.e. no same corresponding token, their similarity is 0.0. If the two pages are exactly same, their similarity will be 1.0.

The reason for giving higher weight to higher level of web pages is because we think that higher path level usually more important than lower level. For example, people will think that the URL

`http://www.cs.ualberta.ca/research/labs/database/current.html`

and the URL

`http://www.cs.ualberta.ca/research/labs/database/publications.html`

are very similar although their last path level are different. By our similarity measurement, the similarity between this two web pages is 0.93. This reflects the truth that the two URLs are very similar, but they are not identical.

### 3.3.2 Similarity Between Sessions

Using the similarity definition in the web page level, now we can define the similarity between two web sessions.

Our basic idea of measuring session similarity is to consider each session as a sequence of web page visiting, and use dynamic programming techniques to find the best matching between two sequences. In this process, web similarity technique discussed in the previous section serves as a page matching goodness function. The final similarity between the two sequences is based on their matching goodness and the length of the sequences.

One difference between our similarity measurement and many of the previous works is: we consider session as a sequence, while many of previous results measure session similarity in either Euclidean space or sets, for example Jaccard Coefficient is widely used. The definition of Jaccard Coefficient is like this:

$$sim(T_1, T_2) = \frac{T_1 \cap T_2}{T_1 \cup T_2}$$

By this definition, if two sessions contain no common web page, their similarity is 0; if two sessions contain same set of web pages, their similarity is 1.

We argue that a URL sequence can better represent the nature of a session than a set. For example, using Jaccard Coefficient similarity measurement there is no difference between the session "**abcd**", "**bcad**" and "**abdc**". Using our session sequence similarity measurement, it can tell you that the three are different, and "**abcd**' is more similar to "**abdc**" than to "**bcad**".

There are many papers [SC70] [TM81] in the area of bioinfomatics area talking about sequence alignment. Their objects are DNA or protein sequences instead of web page sequences. One difference between web page sequences and DNA sequences is: Each DNA sequences contains a sequence of amino acids, and there are tens of different amino acids; However for web session sequences, each sequence contains a sequence of web pages, and there can be thousands of different web pages. Another difference between our web page sequences, i.e. web sessions and their protein sequences is that a protein sequence is typically hundreds of elements, while a session sequence is usually much shorter than a protein sequence. In our real session data set, the average session length is about 11.371 web pages. We don't need to consider some typical problems such as the tradeoff between memory efficiency and computational efficiency in protein sequence alignment.

We use a scoring system which helps finding the optimal matching between two session sequences. An optimal matching is an alignment with the highest score. The score for the optimal matching is then used to calculate the similarity between two sessions. These are the principles in matching the

sequences:

- The session sequences can be shifted right or left to align as many pages as possible. For example, session#1 includes a sequence of visiting to URLs **1, 2, 21, 22**, here each web page is represented by its token string as described in the web page similarity part. Suppose session#2 includes a sequence of visiting to URLs **2, 21, 22**. The best matching between the two session sequences can be achieved by shifting session#2:

  ```
  session#1: 1  2  21  22
  session#2: -  2  21  22
  ```

  In our program, each identical matching, i.e. a pair of pages with similarity 1.0, is given a positive score 20; Each mis-matching, i.e. a pair of pages with similarity 0.0 or match a page with a gap, is given a penalty score $-10$. For a pair of pages with similarity $\alpha$, where $0.0 \leq \alpha \leq 1.0$, the score for their matching is between $-10$ and 20. The final score for the best matching of this example pair of sessions is 50.

- Gaps are allowed to be inserted into the middle, beginning or end of session sequences. This is helpful for achieving better matching. For example, for the following two sessions, a inserted gap in session#2 helps getting the best matching. The final score for the best matching of the following pair of sessions is also 50.

  ```
  session#1: 1 2 21 22
  session#2: 1 2 -  22
  ```

- We do not simply count the number of identical web pages when we are aligning session sequences. Instead, we create a *scoring function* based on web page similarity measurement. For each pair of web pages, the scoring function gives a similarity score where higher score indicates higher similarity between web pages. A pair of identical web pages is only a special case of matching – the *scoring function* return 1.0 which

means the two pages are exactly the same. One example of matching non-identical pages is like following:

```
session#1: 1 2 21
session#2: 1 2 22
```

URL "**21** " in session#1 is matched with URL "**22**" in session#2, and the *scoring function* returns that the similarity between the two web pages is 0.67. The final score for the best matching of this pair of sessions is also 50.

The problem of finding the optimal matching can typically be solved using dynamic programming [KJD00] , and its process can be described by using a matrix as shown in Figure 3.4. One sequence is placed along the top of the matrix and the other sequence is placed along the left side. There is a gap added to the start of each sequence which indicates the starting point of matching. The process of finding the optimal matching between two sequences is actually finding a optimal path from the top left corner to the bottom right corner of the matrix. Any step in any path can only go right, down or diagonal. Every diagonal move corresponds to matching two web pages. A right move corresponds to the insertion of a gap in the vertical sequence and matches a web page in the horizontal sequence with a gap in the vertical sequence. A down move corresponds to the insertion of a gap in the horizontal sequence and matches a web page in the vertical sequence with a gap in the horizontal sequence.

In solving the optimal matching problem, the dynamic programming algorithm propagates scores from the matching start point (upper-left corner), to the destination point (lower-right corner) of the matrix.

The optimal path is then achieved through back propagating from destination point to starting point. In the given example, the optimal path found through back propagating is connected by arrows where the numbers in brackets indicate the step number in back propagating. This optimal path tells the best matching pattern.

The score of any element in the matrix is the maximum of the three scores that can be propagated from the element on its left, the element above it and the element above-left. The score that ends up in the lower-right corner is the optimal sequence alignment score [KJD00]. One example of our matching process and its result is shown in Figure 3.4.

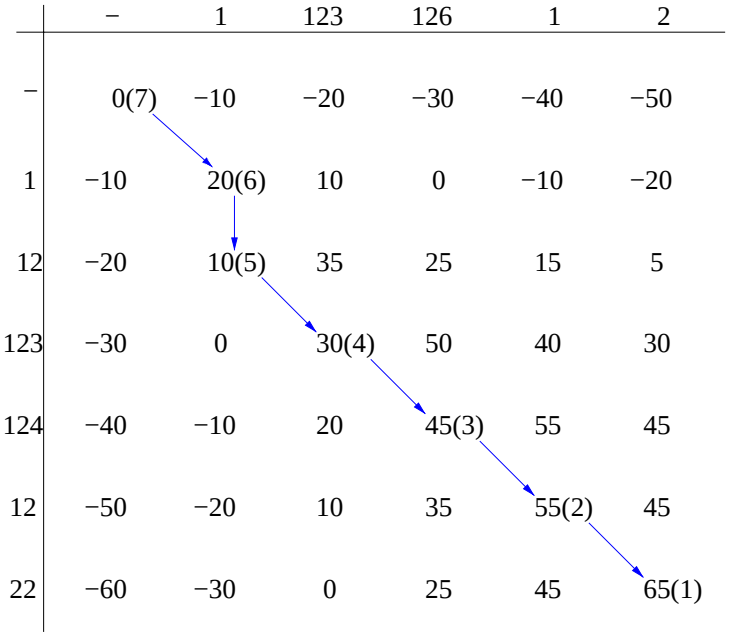


Figure 3.4: Session matching example

After finding the final score for the optimal session alignment, the final similarity between the two sessions is computed by considering the final optimal score and the length of the two sessions. In our method, we first get the length of the shorter session - $lengthShort$, then the similarity between the two sessions is achieved through dividing the optimal matching score by $20 * lengthShort$ because the optimal score can not be more than $20 * lengthShort$ in our scoring system. For the example in Figure 3.4, the similarity between the two sessions is $65/(20 * 5) = 0.65$.

We argue that our similarity measurement is better than previous set similarity measurements, for example Jaccard Coefficient. This is due to two reasons: (1) considering session as sequence of URLs is better than consider-

ing session as a set of URLs. As mentioned before, Jaccard Coefficient cannot differentiate session "**abcd**" from "**bcad**" and "**abdc**", here each token "**a**, **b**, **c** and **d** represents a URL. Our method can not only tell the difference, but also precisely measure the cross similarity between each two of them. (2) In measuring the similarity between sessions, our method considers URL similarity. This has been proven effective in reflecting session similarity in many cases. For example, for the following two sessions:

```
session#1: 12  123  124
session#2: 1   125  126
```

The two sessions have no common URLs, but they are actually about a similar topic. Jaccard Coefficient will tell us that the similarity between the two sessions is 0.0, however our method tells that the two sessions still bear some similarity and their similarity is 0.67. This result better reflects the true connection between the two sessions.

## 3.4 Web Sessions Clustering

The session similarity measurement method described in the last section can be applied to compute the similarity between each pair of sessions, and construct a similarity matrix. Proper clustering algorithms will be applied to this similarity matrix to find the session clusters.

For the known clustering algorithms, we tried ROCK, CHAMELEON and TURN on our testing data set. K-modes is also applicable for categorical data, but its similarity measurement is tightly based on vector space which is different from our sequence similarity measurement, also considering the common problem of k-means family algorithms, we did not try k-mode in our implementation. DBSCAN and WaveCluster can also be applied to some special categorical data sets, however they require that any dimension of the categorical data space can be somehow ordered into numerical order. In this sense, they are not truly algorithms for general categorical data.

Another important issue is how to evaluate the quality of clusters in the result. This issue by itself is worth a thesis. It is never a simple problem.

What people did in the past for 2-dimensional numerical data is to draw the 2-dimensional data into 2-dimensional pictures, and usually people could easily decide the correctness of clustering by vision intuition. However it is much harder to evaluate categorical data, like session data. What we did is to order the resulting clusters according to their descending sizes, and draw a three dimensional picture to represent the cross similarity between sessions in different clusters. For example, suppose for an ideal case we have three clusters in a 1000 sessions data set, cluster #1 has 500 sessions, cluster #2 has 300 sessions, and cluster #3 has 200 sessions. The cross similarity between each pair of sessions within a same cluster is 1.0, and cross similarity between each pair of sessions from two different clusters is 0.0. For the 3-dimensional picture, we use x-axis and y-axis to represent 1000 sessions, z-axis is used to represent the similarity between the two sessions from x-axis and y-axis. For this given very simple and special example, if we represent the third dimension - similarity by colour: golden colour means the similarity is 1.0, white colour means the similarity is 0.0, we expect to see a 3-dimensional picture like Figure 3.5. For this very special example, we see golden blocks along the diagonal of the x-y two dimensional space. This tells us the similarities between sessions within a same cluster are high, and the similarities between sessions which belong to different clusters are low. For real problems, the cluster result can rarely be so clear as the given example, but for successful clustering, we should see higher square areas along the diagonal of x-y space, and all the other areas should be lower.

Our testing session set has 1000 randomly selected sessions from a real tele-learning system. Both Jaccard similarity and our Dynamic-Programming-Based similarity measurement method are used to provide similarity matrix for the given session set. ROCK, CHAMELEON and TURN are then applied on the similarity matrices to each produce clustering result. Clustering results of ROCK, CHAMELEON and TURN are shown in Figure 3.6 through Figure 3.11. From the clustering results, we found that ROCK tends to find bigger clusters with lower average similarity. This is consistent with the testing of ROCK on 2-dimensional numerical data in last chapter where we see ROCK

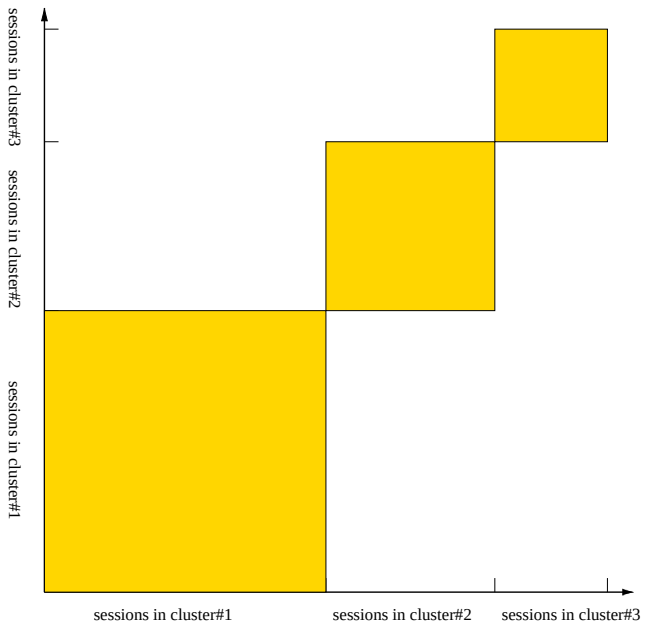


Figure 3.5: session clustering result visualization example

tends to merge several genuine clusters into a bigger cluster. CHAMELEON and TURN can find clusters with high internal cross similarity. The difference between CHAMELEON and TURN is that TURN can identify outliers while CHAMELEON cannot.

In this work, we cannot compare the cluster quality between clusters using Jaccard Coefficient similarity measurement and using Dynamic-Programming-Based similarity measurement, however, we found that Jaccard Coefficient measurement tends to give more clusters than Dynamic-Programming-Based measurement. This is because in Jaccard Coefficient similarity computation, similarities between sessions are concentrated into certain values due to the way Jaccard Coefficient is computed. However in Dynamic-Programming-Based measurement, similarities are not concentrated.

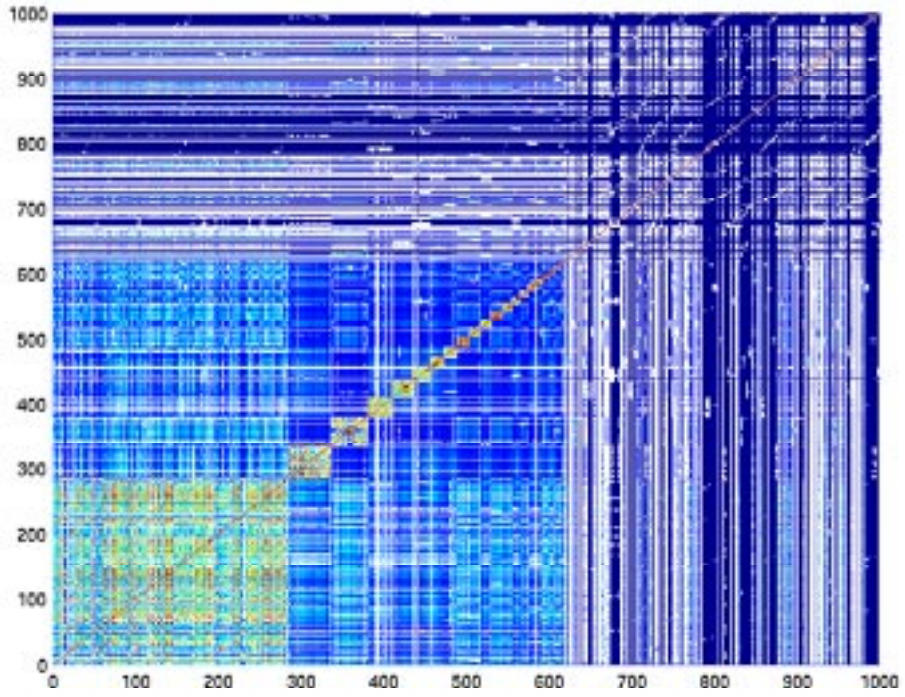


Figure 3.6: ROCK's clustering result on Jaccard Coefficient similarity matrix

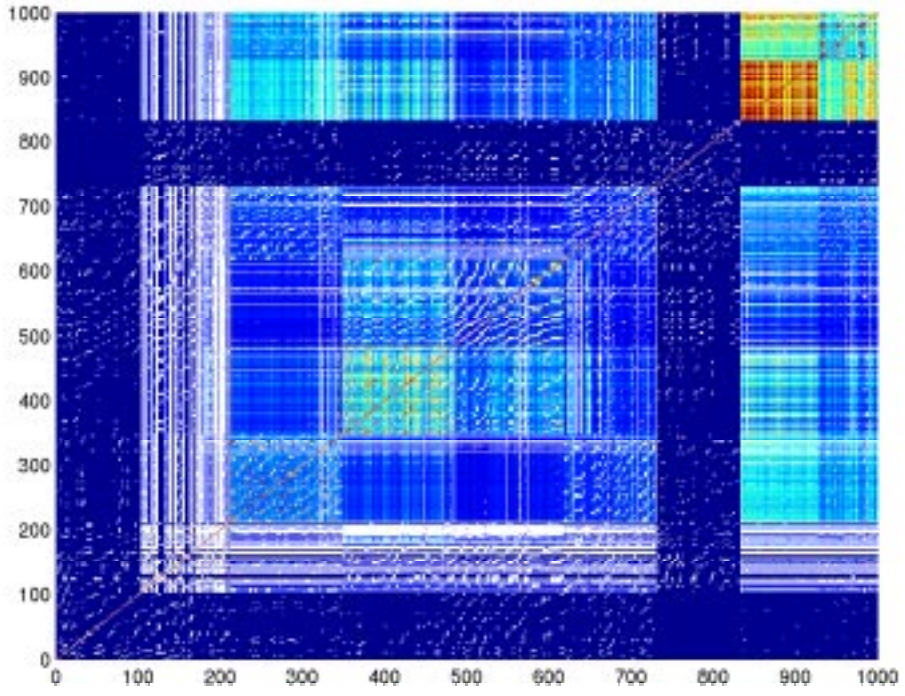


Figure 3.7: CHAMELEON's clustering result on Jaccard Coefficient similarity matrix

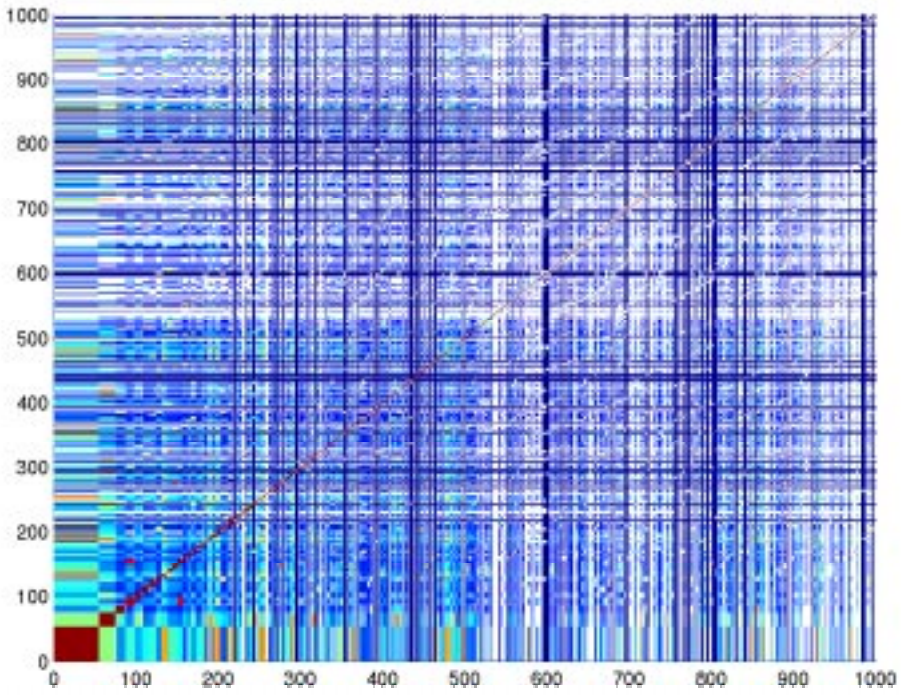


Figure 3.8: TURN's clustering result on Jaccard Coefficient similarity matrix

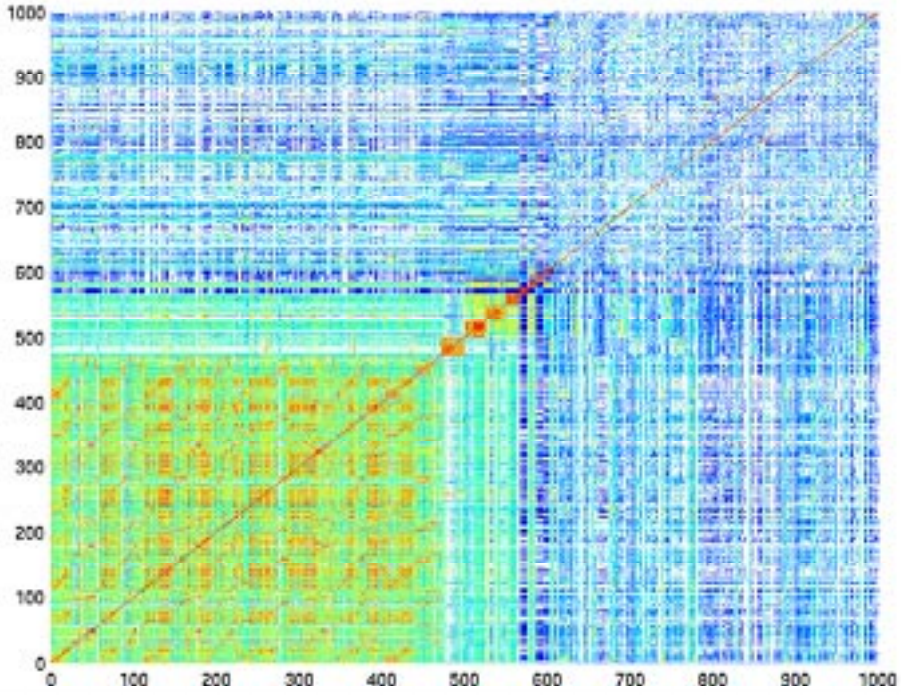


Figure 3.9: ROCK's clustering result on Dynamic-Programming-Based similarity matrix

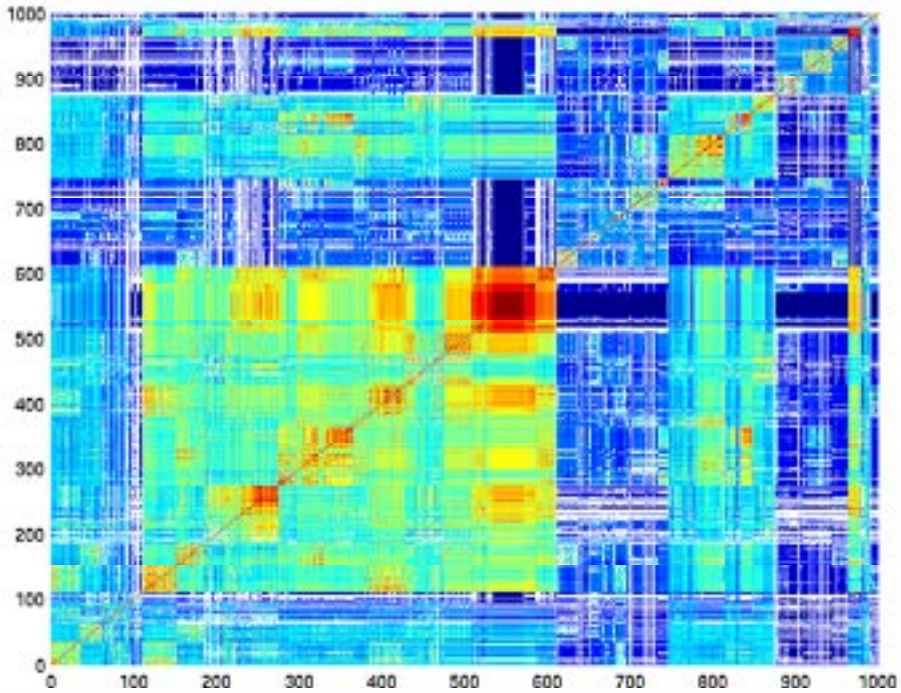


Figure 3.10: CHAMELEON's clustering result on Dynamic-Programming-Based similarity matrix

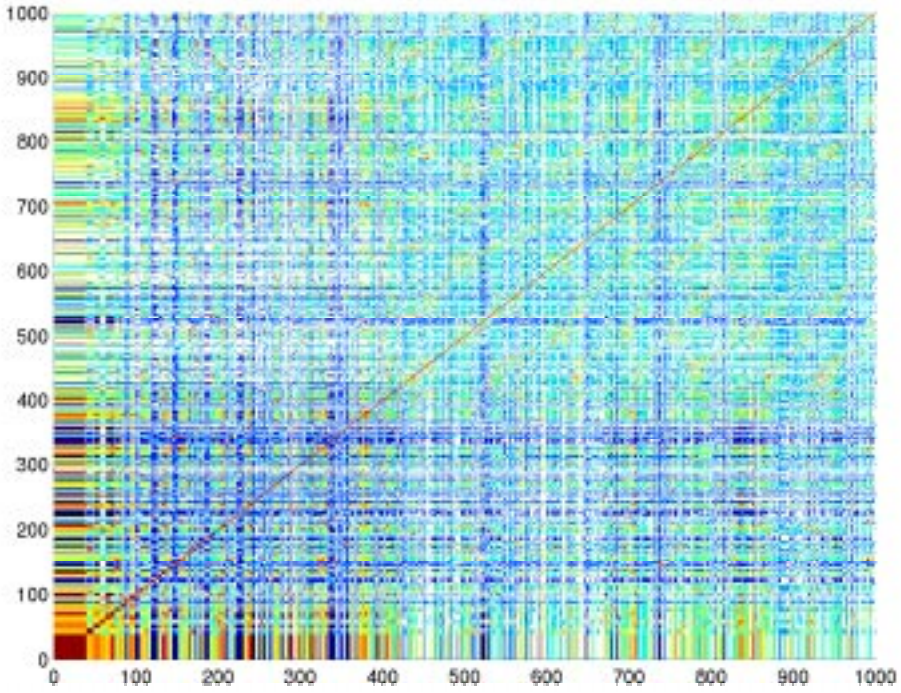


Figure 3.11: TURN's clustering result on Dynamic-Programming-Based similarity matrix

# Chapter 4

# Conclusion and Further Work

This work started with a survey and implementation on major and recently devised clustering algorithms, then it paid attention to the practical problem of clustering web sessions. A new similarity measurement for session data based on dynamic programming is presented. Three algorithms applicable for categorical data are applied to the real session data.

Through the survey and implementation, we achieved deeper understanding of the major clustering algorithms. In conclusion, we feel that there is still not a single method which can claim fully satisfying all the requirements for a good clustering method. However, in experiments we see that WaveCluster, DBSCAN/OPTICS, CHAMELEON and TURN* have better effectiveness and efficiency in clustering 2 dimensional spatial data than other methods. For this part, further work includes applying more testing data sets, especially high dimensional data sets to the algorithms.

In the web session clustering section, through analysis and examples, we introduced a new session similarity measurement: Dynamic-Programming based session similarity measurement. In the experiment, we can compare the clustering characteristics of the three algorithms (TURN, ROCK and CHAMELEON) on one session similarity measurement (Jaccard Coefficient or Dynamic Programming Based measurement), however we cannot compare the clustering goodness between the two similarity measurements because we don't know what is the *correct* clustering result for this randomly selected real session set. For this part, further work includes more precisely evaluating the clustering

results. This includes comparing the clustering results between using Jaccard Coefficient similarity measurement and using Dynamic-Programming-Based similarity measurement. Also we wish to more precisely evaluate the clustering results of ROCK, CHAMELEON and TURN. This can be achieved only if we synthetically create one testing session data set. If we exactly know which session should belong to which cluster, then precise measurement of the quality of clustering can be achieved.

# Bibliography


[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 94–105, 1998.

[AMP99] A.K.Jain, M.N.Murty, and P.J.Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[BG01] Arindam Banerjee and Joydeep Ghosh. Clickstream clustering using weighted longest common subsequences. In *Proc. of Workshop on Web Mining in First International SIAM Conference on Data Mining (SDM2001)*, pages 33–40, Chicago, April 2001.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, 1996.

[FSS99] Yongjian Fu, Kanwalpreet Sandhu, and Ming-Yi Shih. Clustering of web users based on access patterns. *Workshop on Web Usage Analysis and User Profiling (WEBKDD99)*, August 1999.

[FWZ01] Andrew Foss, Weinan Wang, and Osmar R. Zaïane. A non-parametric approach to web log analysis. In *Proc. of Workshop on Web Mining in First International SIAM Conference on Data Mining (SDM2001)*, pages 41–50, Chicago, April 2001.

[GRS99] Studipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: a robust clustering algorithm for categorical attributes. In *15th Int'l Conf. on Data Eng.*, 1999.

[HC00] H.Mannila and C.Meek. Global partial orders from sequential data. In *Proc. 6th Intl. Conf. on Knowledge Discovery and Data Mining (KDD2000)*, pages 161–168, August 2000.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[HK01] David Harel and Yehuda Koren. Clustering spatial data using random walks. In *Proceedings of KDD-2001*, 2001.

[HKT01] Jiawei Han, Micheline Kamber, and Anthony K.H. Tung. Spatial clustering methods in data mining: A survey. In H. Miller and

J. Han, editors, *Geographic Data Mining and Knowledge Discovery*. Taylor and Francis, 2001.

[Hua98] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2:283–304, 1998.

[J.M67] J.MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Math. Statist. Prob.*, 1967.

[JP99] J.Pitkow and P.Pirolli. Mining longest repeating subsequences to predict world wide web surfing. In *Proc. 2nd USENIX symposium on Internet Technologies and Systems (USITS'99)*, October 1999.

[KHK99] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32(8):68–75, August 1999.

[KJD00] K.Charter, J.Schaeffer, and D.Szafron. Sequence alignment using fastlsa. In *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS'2000)*, pages 239–245, 2000.

[LP90] L.Kaufman and P.J.Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[MCS99] B. Mobasher, R. Cooly, and J. Srivastava. Automatic personalization based on web usage mining. Technical Report TR99-010, Department of Computer Science, Depaul University, 1999.

[ML99] M.Spiliopoulou and L.C.Faulstich. Wum: A tool for web utilization analysis. In *Extended version of Proc. EDBT Workshop WebDB'98*, pages 184–203, Springer Verlag, 1999.

[MMKJ99] M.Ankerst, M.Breunig, H.-P. Kriegel, and J.Sander. Optics: Ordering points to identify the clustering structure. In *Proc. 1999 ACM-SIGMOD Conf. on Management of Data (SIGMOD'99)*, pages 49–60, 1999.

[RJ94] R.Ng and J.H. Efficient and effective clustering method for spatial data mining. In *Conf. on Very Large Data Bases (VLDB'94)*, pages 144–155, Santiago, Chile, September 1994.

[SC70] S.Needleman and C.Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[SCDT00] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations*, Jan 2000.

[SCZ98] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: a multi-resolution clustering approach for very large spatial databases. In *24th VLDB Conference*, New York, USA, 1998.

[SG98] K. Shim S. Guha, R. Rastogi. CURE: An efficient clustering algorithm for large databases. In *SIGMOD'98*, Seattle, Washington, 1998.

[SZAS97] C. Shahabi, A.M. Zarkesh, J. Adibi, and V. Shah. Knowledge discovery from users web-page navigation. In *workshop on Research Issues in Data Engineering*, Birminghan, England, 1997.

[TM81] T.Smith and M.Waterman. Identification of common molecular sequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[WJR97] W.Wang, J.Yang, and R.Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97)*, pages 186–195, 1997.

[ZFW02] Osmar R. Zaïane, Andrew Foss, and Weinan Wang. Turn* for an unsupervised clustering approach. In *Second International SIAM Data Mining Conference*, 2002.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *1996 ACM SIGKDD Int. Conf. Managament of Data*, pages 103–114, June 1996.