CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling

George Karypis Eui-Hong (Sam) Han Vipin Kumar

Department of Computer Science and Engineering
University of Minnesota
4-192 EECS Bldg., 200 Union St. SE
Minneapolis, MN 55455, USA

Technical Report #99-007
To Appear in the *IEEE Computer*: Special Issue on Data Analysis and Mining

{karypis,han,kumar}@cs.umn.edu

Abstract

Clustering in data mining is a discovery process that groups a set of data such that the intracluster similarity is maximized and the intercluster similarity is minimized. Existing clustering algorithms, such as K-means, PAM, CLARANS, DBSCAN, CURE, and ROCK are designed to find clusters that fit some static models. These algorithms can breakdown if the choice of parameters in the static model is incorrect with respect to the data set being clustered, or if the model is not adequate to capture the characteristics of clusters. Furthermore, most of these algorithms breakdown when the data consists of clusters that are of diverse shapes, densities, and sizes. In this paper, we present a novel hierarchical clustering algorithm called CHAMELEON that measures the similarity of two clusters based on a dynamic model. In the clustering process, two clusters are merged only if the inter-connectivity and closeness (proximity) between two clusters are high relative to the internal inter-connectivity of the clusters and closeness of items within the clusters. The merging process using the dynamic model presented in this paper facilitates discovery of natural and homogeneous clusters. The methodology of dynamic modeling of clusters used in CHAMELEON is applicable to all types of data as long as a similarity matrix can be constructed. We demonstrate the effectiveness of CHAMELEON in a number of data sets that contain points in 2D space, and contain clusters of different shapes, densities, sizes, noise, and artifacts. Experimental results on these data sets show that CHAMELEON can discover natural clusters that many existing state-of-the art clustering algorithms fail to find.

Keywords: Clustering, data mining, dynamic modeling, graph partitioning, k-nearest neighbor graph.

1 Introduction

Clustering in data mining [SAD+93, CHY96] is a discovery process that groups a set of data such that the intracluster similarity is maximized and the intercluster similarity is minimized [JD88, KR90, PAS96, CHY96]. These discovered clusters can be used to explain the characteristics of the underlying data distribution, and thus serve as the foundation for other data mining and analysis techniques. The applications of clustering include characterization of different customer groups based upon purchasing patterns, categorization of documents on the World Wide Web [BGG+99a, BGG+99b], grouping of genes and proteins that have similar functionality [HHS92, NRS+95, SCC+95, HKKM98], grouping of spatial locations prone to earth quakes from seismological data [BR98, XEKS98], etc.

Existing clustering algorithms, such as K-means [JD88], PAM [KR90], CLARANS [NH94], DBSCAN [EKSX96], CURE [GRS98], and ROCK [GRS99] are designed to find clusters that fit some static models. For example, K-means, PAM, and CLARANS assume that clusters are hyper-ellipsoidal (or globular) and are of similar sizes. DBSCAN assumes that all points within genuine clusters are density reachable [1] and points across different clusters are not. Agglomerative hierarchical clustering algorithms, such as CURE and ROCK use a static model to determine the most similar cluster to merge in the hierarchical clustering. CURE measures the similarity of two clusters based on the similarity of the closest pair of the representative points belonging to different clusters, without considering the internal closeness (i.e., density or homogeneity) of the two clusters involved. ROCK measures the similarity of two clusters by comparing the aggregate inter-connectivity of two clusters against a user-specified static inter-connectivity model, and thus ignores the potential variations in the inter-connectivity of different clusters within the same data set. These algorithms can breakdown if the choice of parameters in the static model is incorrect with respect to the data set being clustered, or if the model is not adequate to capture the characteristics of clusters. Furthermore, most of these algorithms breakdown when the data consists of clusters that are of diverse shapes, densities, and sizes.

In this paper, we present a novel hierarchical clustering algorithm called CHAMELEON that measures the similarity of two clusters based on a dynamic model. In the clustering process, two clusters are merged only if the inter-connectivity and closeness (proximity) between two clusters are comparable to the internal inter-connectivity of the clusters and closeness of items within the clusters. The merging process using the dynamic model presented in this paper facilitates discovery of natural and homogeneous clusters. The methodology of dynamic modeling of clusters used in CHAMELEON is applicable to all types of data as long as a similarity matrix can be constructed. We demonstrate the effectiveness of CHAMELEON in a number of data sets that contain points in 2D space, and contain clusters of different shapes, densities, sizes, noise, and artifacts.

The rest of the paper is organized as follows. Section 2 gives an overview of related clustering algorithms. Section 3 presents the limitations of the recently proposed state of the art clustering algorithms. We present our new clustering algorithm in Section 4. Section 5 gives the experimental results. Section 6 contains conclusions and directions for future work.

2 Related Work

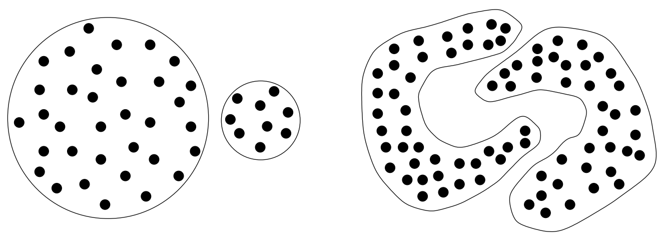
In this section, we give a brief description of existing clustering algorithms.

[1] A point p is density reachable from a point q, if they are connected by a chain of points such that each point has minimal number of data points, including the next point in the chain, within a fixed radius [EKSX96].

2.1 Partitional Techniques

Partitional clustering attempts to break a data set into K clusters such that the partition optimizes a given criterion [JD88, KR90, NH94, CS96]. Centroid-based approaches, as typified by K means [JD88] and ISODATA [BH64], try to assign points to clusters such that the mean square distance of points to the centroid of the assigned cluster is minimized. Centroid-based techniques are suitable only for data in metric spaces (e.g., Euclidean space) in which it is possible to compute a centroid of a given set of points. Medoid-based methods, as typified by PAM (Partitioning Around Medoids) [KR90] and CLARANS [NH94], work with similarity data, i.e., data in an arbitrary similarity space [GRG$^+$99]. These techniques try to find representative points (medoids) so as to minimize the sum of the distances of points from their closest medoid.

A major drawback of both of these schemes is that they fail for data in which points in a given cluster are closer to the center of another cluster than to the center of their own cluster. This can happen in many natural clusters [HKKM97, GRS99]; for example, if there is a large variation in cluster sizes (as in Figure 1 (a)) or when cluster shapes are convex (as in Figure 1 (b)).



a) Clusters of widely differente sizes   b) Clusters with convex shapes

Figure 1: Data sets on which centroid and medoid approaches fail.

2.2 Hierarchical Techniques

Hierarchical clustering algorithms produce a nested sequence of clusters, with a single all-inclusive cluster at the top and single point clusters at the bottom. Agglomerative hierarchical algorithms [JD88] start with all the data points as a separate cluster. Each step of the algorithm involves merging two clusters that are the most similar. After each merge, the total number of clusters decreases by one. These steps can be repeated until the desired number of clusters is obtained or the distance between two closest clusters is above a certain threshold distance.

There are many different variations of agglomerative hierarchical algorithms [JD88]. These algorithms primarily differ in how they update the similarity between existing clusters and the merged clusters. In some methods [JD88], each cluster is represented by a centroid or medoid of the points contained in the cluster, and the similarity between two clusters is measured by the similarity between the centroids/medoids of the clusters. Like partitional techniques, such as K-means and K-medoids, these method also fail on clusters of arbitrary shapes and different sizes.

In the single link method [JD88], each cluster is represented by all the data points in the cluster. The similarity between two clusters is measured by the similarity of the closest pair of data points belonging to different clusters. Unlike the centroid/medoid based methods, this method can find clusters of arbitrary shape and different sizes. However, this method is highly susceptible to noise, outliers, and artifacts.

CURE [GRS98] has been proposed to remedy the drawbacks of both of these methods while combining their advantages. In CURE, instead of using a single centroid to represent a cluster, a constant number of representative points are chosen to represent a cluster. The similarity between two clusters is measured by the similarity of the closest pair of the representative points belonging to different clusters. New representative points for the merged clusters are determined by selecting a constant number of well scattered points from all the data points and shrinking them towards the centroid of the cluster according to a shrinking factor. Unlike centroid/medoid based methods, CURE is capable of finding clusters of arbitrary shapes and sizes, as it represents each cluster via multiple representative points. Shrinking the representative points towards the centroid helps CURE in avoiding the problem of noise and outliers present in the single link method. The desirable value of the shrinking factor in CURE is dependent upon cluster shapes and sizes, and amount of noise in the data.

In some agglomerative hierarchical algorithms, the similarity between two clusters is captured by the aggregate of the similarities (i.e., interconnectivity) among pairs of items belonging to different clusters. The rationale for this approach is that subclusters belonging to the same cluster will tend to have high interconnectivity. But the aggregate inter-connectivity between two clusters depends on the size of the clusters involved, and in general pairs of larger clusters will have higher inter-connectivity. Hence, many such schemes normalize the aggregate similarity between a pair of clusters with respect to the expected inter-connectivity of the clusters involved. For example, the widely used group-average method [JD88] assumes fully connected clusters, and thus scales the aggregate similarity between two clusters by $n \times m$, where n and m are the size of the two clusters, respectively. ROCK [GRS99], a recently developed agglomerative algorithm that operates on a derived similarity graph, scales the aggregate inter-connectivity with respect to a user-specified inter-connectivity model.

Most of the algorithms discussed above work implicitly or explicitly with the $n \times n$ similarity matrix such that (i, j) element of the matrix represents the similarity between i^{th} and j^{th} data items. Some algorithms derive a new similarity matrix using the original matrix [JP73, GK78, JD88, GRS99], and then apply one of the existing techniques on this derived similarity matrix. In many cases, the new derived similarity matrix is just a sparsified version of this original similarity matrix from which certain entries (e.g., those whose value is below a threshold) have been deleted. In other cases, the derived similarity matrix has entirely different values [JP73, GK78, GRS99]. The sparsified derived matrix can help eliminate/reduce noise from the data, and substantially reduce the execution time of many algorithms. In some cases, it can also provide a better model of similarities for the problem domain. For example, mutual shared method presented in [JP73] helps remove noise and outliers and is shown to provide a better model to capture similarities among transactions in [GRS99].

A sparse similarity matrix can be represented by a sparse graph, and tightly connected clusters of this graph can be found by divisive hierarchical clustering algorithms such as those based upon minimal spanning tree (MST) [JD88] or graph-partitioning algorithms [KK98b, KK99a]. MST-based algorithms are highly susceptible to noise and artifacts just like the single link method. Graph-partitioning based methods are much more robust, but they tend to break genuine clusters if there is a large variations in cluster sizes.

3 Limitations of Existing Hierarchical Schemes

A major limitation of existing agglomerative hierarchical schemes such as the Group Averaging Method [JD88], ROCK [GRS99], and CURE [GRS98] is that the merging decisions are based upon static modeling of the clusters to be merged. In other words, these schemes fail to take into account special characteristics of individual clusters, and thus can make incorrect merging decisions when the underlying data does not follow the assumed model, or when noise is present. For example, consider the four sub-clusters of points in 2D shown in Figure 2. The selection mechanism of CURE (and of the single link method) will prefer merging clusters (a) and (b) over merging clusters (c) and (d), since the minimum distances between the representative points of (a) and (b) will be smaller than those for clusters (c) and (d). But clusters (c) and (d) are better candidates for merging because the minimum distances between the boundary points of (c) and (d) are of the same order as the average of the minimum distances of any points within these clusters to other points. Hence, merging (c) and (d) will lead to a more homogeneous and natural cluster than merging (a) and (b).

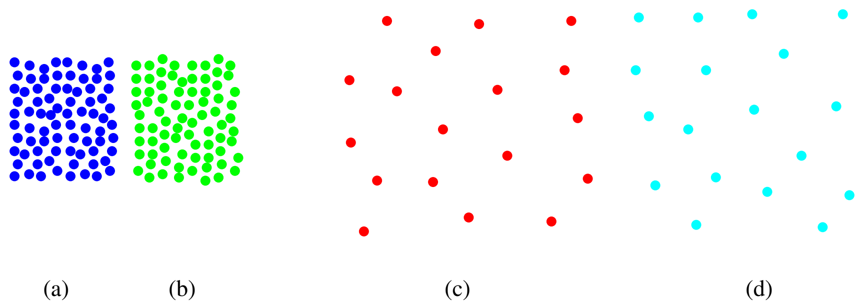


Figure 2: Example of clusters for merging choices.

In agglomerative schemes based upon group averaging [JD88] and related schemes such as ROCK, connectivity among pairs of clusters is scaled with respect to the expected connectivity between these clusters. However, the key limitation of all such schemes is that they assume a static, user supplied inter-connectivity model, which is inflexible and can easily lead to wrong merging decisions when the model under- or over-estimates the inter-connectivity of the data set or when different clusters exhibit different inter-connectivity characteristics. Although some schemes allow the connectivity to be different for different problem domains (*e.g.*, ROCK [GRS99]), it is still the same for all clusters irrespective of their densities and shapes. Consider the two pairs of clusters shown in Figure 3, where each cluster is depicted by a sparse graph where nodes indicate data items and edges represent that their two vertices are similar. The number of items in all four clusters is the same. Let us assume that in this example all edges have equal weight (*i.e.*, they represent equal similarity). Then both ROCK selection mechanism (irrespective of the assumed model of connectivity) and the group averaging method will select pair {(c),(d)} for merging, whereas the pair {(a),(b)} is a better choice.

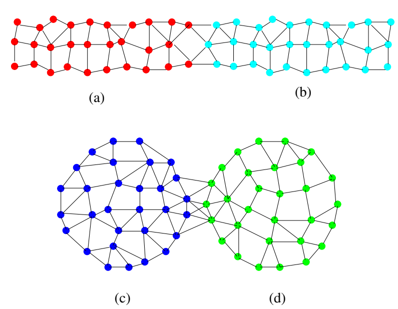


Figure 3: Example of clusters for merging choices.

The selection mechanism in CURE (and related algorithms such as single link method [JD88]) considers only the minimum distance between the representative points of two clusters, and does not consider the aggregate interconnectivity among the two clusters. Similarly, the selection mechanism of algorithms such as ROCK only considers the aggregate inter-connectivity across the pairs of clusters (appropriately scaled by the expected value of the inter-connectivity), but ignores the value of the strongest edge (or edges) across clusters. However, by looking at only one of these two characteristics, these algorithm can easily select to merge the wrong pair of clusters. For instance, as the example in Figure 4 illustrates, an algorithm that focuses only on the closeness of two clusters will incorrectly prefer to merge clusters (c) and (d) over clusters (a) and (b). Similarly, as the example in Figure 5 illustrates, an algorithm that focuses only on the inter-connectivity of two clusters will incorrectly prefer to merge cluster (a) with cluster (c) rather than with (b). (Here we assume that the aggregate interconnectivity between items in clusters (a) and (c) is greater than that between items in clusters (a) and (b). However, the border points of cluster (a) are much closer than those of (b) than to those of (c).)

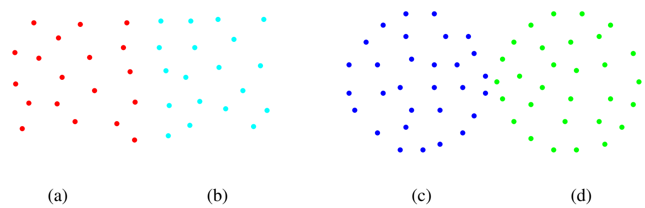


Figure 4: Example of clusters for merging choices.

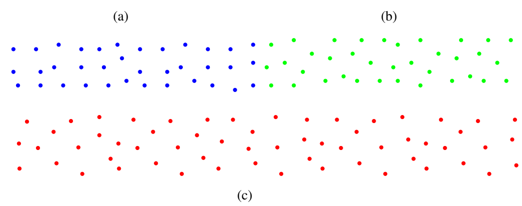


Figure 5: Example of clusters for merging choices.

In summary, there are two major limitations of the agglomerative mechanisms used in existing schemes. First, these schemes do not make use of information about the nature of individual clusters being merged. Second, one set of schemes (CURE and related schemes) ignore the information about the aggregate interconnectivity of items in two clusters, whereas the other set of schemes (ROCK, the group averaging method, and related schemes) ignore information about the closeness of two clusters as defined by the similarity of the closest items across two clusters.

In the following section, we present a novel scheme that addresses both of these limitations.

4 CHAMELEON: Clustering Using Dynamic Modeling

4.1 Overview

In this section we present CHAMELEON, a new clustering algorithm that overcomes the limitations of existing agglomerative hierarchical clustering algorithms discussed in Section 3. Figure 6 provides an overview of the overall approach used by CHAMELEON to find the clusters in a data set.

CHAMELEON operates on a sparse graph in which nodes represent data items, and weighted edges represent similarities among the data items. This sparse graph representation of the data set allows CHAMELEON to scale to large data sets and to operate successfully on data sets that are available only in similarity space [GRG+99] and not in metric spaces [GRG+99]. CHAMELEON finds the clusters in the data set by using a two phase algorithm. During the first phase, CHAMELEON uses a graph partitioning algorithm to cluster the data items into a large number of relatively small sub-clusters. During the second phase, it uses an agglomerative hierarchical clustering algorithm to find the genuine clusters by repeatedly combining together these sub-clusters.

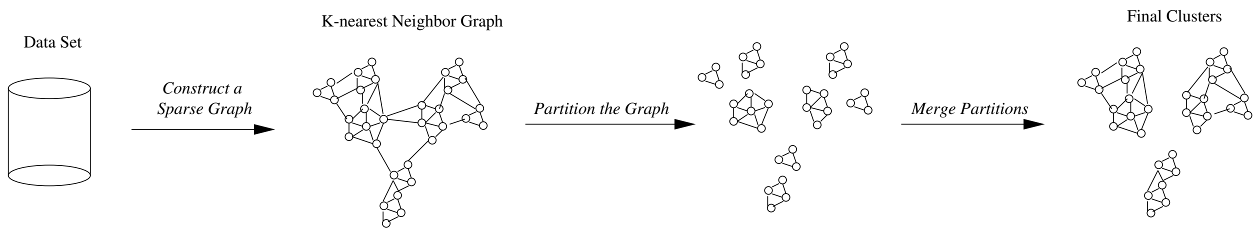


Figure 6: Overall framework CHAMELEON.

The key feature of CHAMELEON's agglomerative hierarchical clustering algorithm is that it determines the pair of most similar sub-clusters by taking into account both the inter-connectivity as well as the closeness of the clusters; and thus it overcomes the limitations discussed in Section 3 that result from using only one of them. Furthermore, CHAMELEON uses a novel approach to model the degree of inter-connectivity and closeness between each pair of clusters that takes into account the internal characteristics of the clusters themselves. Thus, it does not depend on a static user supplied model, and can automatically adapt to the internal characteristics of the clusters being merged.

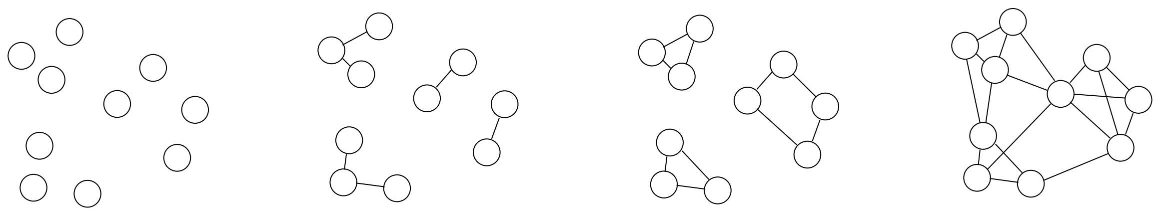
In the rest of this section we provide details on how to model the data set, how to dynamically model the similarity between the clusters by computing their *relative inter-connectivity* and *relative closeness*, how graph partitioning is used to obtain the initial fine-grain clustering solution, and how the relative inter-connectivity and relative closeness are used to repeatedly combine together the sub-clusters in a hierarchical fashion.

4.2 Modeling the Data

Given a similarity matrix, many methods can be used to find a graph representation [JP73, GK78, JD88, GRS99]. In fact, modeling data items as a graph is very common in many hierarchical clustering algorithms. For example, agglomerative hierarchical clustering algorithms based on single link, complete link, or group averaging method [JD88] operate on a complete graph. ROCK [GRS99] first constructs a sparse graph from a given data similarity matrix using a similarity threshold and the concept of shared neighbors, and then performs a hierarchical clustering algorithm on the sparse graph. CURE [GRS98] also implicitly employs the concept of a graph. In CURE, when cluster representative points are determined, a graph containing only these representative points is implicitly constructed. In this graph, edges only connect representative points from different clusters. Then the closest edge in this graph is identified and the clusters connected by this edge is merged.

CHAMELEON's sparse graph representation of the data items is based on the commonly used k-nearest neighbor graph approach. Each vertex of the k-nearest neighbor graph represents a data item, and there exists an edge between two vertices, if data items corresponding to either of the nodes is among the k-most similar data points of the data point corresponding to the other node. Figure 7 illustrates the 1-, 2-, and 3-nearest neighbor graphs of a simple data set. Note that since CHAMELEON operates on a sparse graph, each cluster is nothing more than a sub-graph of the original sparse graph representation of the data set.

There are several advantages of representing data using a k-nearest neighbor graph G_k. Firstly, data points that are far apart are completely disconnected in the G_k. Secondly, G_k captures the concept of neighborhood dynamically. The neighborhood radius of a data point is determined by the density of the region in which this data point resides. In a dense region, the neighborhood is defined narrowly and in a sparse region, the neighborhood is defined more widely. Compared to the model defined by DBSCAN [EKSX96] in which a global neighborhood density is specified, G_k captures more natural neighborhood. Thirdly, the density of the region is recorded as the weights of the edges. The edge weights of dense regions in G_k (with edge weights representing similarities) tend to be large and the edge weights



(a) Original Data in 2D (b) 1-nearest neighbor graph (c) 2-nearest neighbor graph (d) 3-nearest neighbor graph

Figure 7: k-nearest graphs from an original data in 2D.

of sparse regions tend to be small. As the consequence, a min-cut bisection of the graph represents the interface layer of sparse region of the graph. Finally, G_k provides a computational advantage over a full graph in many algorithms operating on graphs, including graph partitioning and partitioning refinement algorithms.

4.3 Modeling the Cluster Similarity

To address the limitations of agglomerative schemes discussed in Section 3, CHAMELEON determines the similarity between each pair of clusters C_i and C_j by looking both at their relative inter-connectivity $RI(C_i, C_j)$ and their relative closeness $RC(C_i, C_j)$. CHAMELEON's hierarchical clustering algorithm selects to merge the pair of clusters for which both $RI(C_i, C_j)$ and $RC(C_i, C_j)$ are high; *i.e.*, it selects to merge clusters that are well inter-connected as well as close together with respect to the internal inter-connectivity and closeness of the clusters. By selecting clusters based on both of these criteria, CHAMELEON overcomes the limitations of existing algorithms that look either at the absolute inter-connectivity or absolute closeness. For instance, in the examples shown in Figures 4 and 5 and discussed in Section 3, CHAMELEON will select to merge the correct pair of clusters.

In the remaining of this section we describe how the relative inter-connectivity and relative closeness is computed for a pair of clusters.

Relative Inter-Connectivity The relative inter-connectivity between a pair of clusters C_i and C_j is defined as the absolute inter-connectivity between C_i and C_j normalized with respect to the internal inter-connectivity of the two clusters C_i and C_j. The absolute inter-connectivity between a pair of clusters C_i and C_j is defined to be as the sum of the weight of the edges that connect vertices in C_i to vertices in C_j. This is essentially the edge-cut of the cluster containing both C_i and C_j such that the cluster is broken into C_i and C_j. We denote this by $EC_{\{C_i,C_j\}}$. The internal inter-connectivity of a cluster C_i can be easily captured by the size of its min-cut bisector EC_{C_i} (*i.e.*, the weighted sum of edges that partition the graph into two roughly equal parts). Recent advances in the graph-partitioning technology has made it possible to find such bisector quite efficiently [KK98b, KK99a].

Thus the relative inter-connectivity between a pair of clusters C_i and C_j is given by

$$RI(C_i, C_j) = \frac{|EC_{\{C_i,C_j\}}|}{\frac{|EC_{C_i}|+|EC_{C_j}|}{2}}, \tag{1}$$

which normalizes the absolute inter-connectivity with the average internal inter-connectivity of the two clusters.

By focusing on the relative inter-connectivity between clusters, CHAMELEON can overcome the limitations of existing algorithms that use static inter-connectivity models. For instance, in the example shown in Figure 3 that was discussed in Section 3, CHAMELEON will correctly prefer to merge clusters (a) and (b) over clusters (c) and (d), because the relative inter-connectivity between clusters (a) and (b) is higher than the relative inter-connectivity between clusters (c) and (d), even though the later pair of clusters have a higher absolute inter-connectivity. Thus, the relative inter-connectivity is able to take into account differences in shapes of the clusters (as in Figure 3) as well as differences in degree of connectivity of different clusters.

Relative Closeness The relative closeness between a pair of clusters C_i and C_j is defined as the absolute closeness between C_i and C_j normalized with respect to the internal closeness of the two clusters C_i and C_j. The absolute closeness between a pair of clusters can be captured in a number of different ways. Many existing schemes, capture this closeness by focusing on the pair of points between all the points (or representative points [GRS98]) from C_i and C_j that are closest. A key drawback of these schemes is that by relying only on a single pair of points, they are less tolerant to outliers and noise. For this reason, CHAMELEON measures the closeness of two clusters by computing the average similarity between the points in C_i that are connected to points in C_j. Since these connections are determined using the k-nearest neighbor graph, their average strength provides a very good measure of the affinity between the data items along the interface layer of the two sub-clusters, and at the same time is tolerant to outliers and noise. Note that this average similarity between the points from the two clusters is equal to the average weight of the edges connecting vertices in C_i to vertices in C_j.

The internal closeness of each cluster C_i can also be measured in a number of different ways. One possible approach is to look at all the edges connecting vertices in C_i (*i.e.*, edges that are internal to the cluster), and compute the internal closeness of a cluster as the average weight of these edges. One can argue that in a hierarchical clustering setting, the edges used for agglomeration early on are stronger than those used in later stages. Hence, average weights of the edges on the internal bisection of C_i and C_j will tend to be smaller than the average weight of all the edges in these clusters. But the average weight of these edges is a better indicator of the internal closeness of these clusters.

Hence in CHAMELEON, the relative closeness between a pair of clusters C_i and C_j is computed as,

$$RC(C_i, C_j) = \frac{\overline{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}}, \tag{2}$$

where $\overline{S}_{EC_{C_i}}$ and $\overline{S}_{EC_{C_j}}$ are the average weights of the edges that belong in the min-cut bisector of clusters C_i and C_j, respectively, and $\overline{S}_{EC_{\{C_i,C_j\}}}$ is the average weight of the edges that connect vertices in C_i to vertices in C_j. Also note that a weighted average of the internal closeness of clusters C_i and C_j is used to normalize the absolute closeness of the two clusters, that favors the absolute closeness of cluster that contains the larger number of vertices.

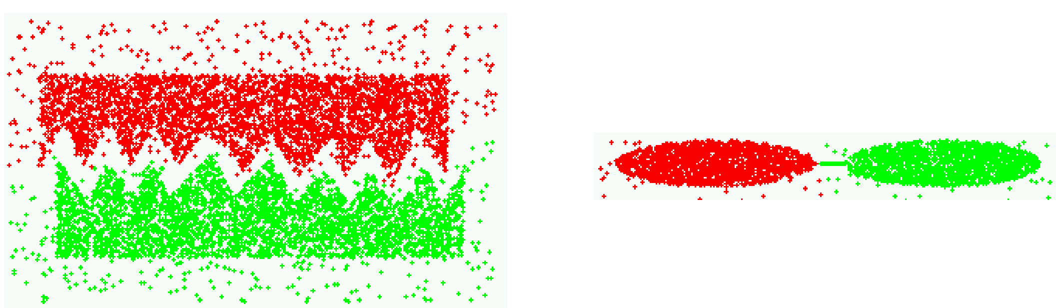
By focusing on the relative closeness between clusters, CHAMELEON can overcome the limitations of existing algorithms that look only at the absolute closeness. For instance, in the example shown in Figure 2 that was discussed in Section 3, CHAMELEON will correctly prefer to merge the clusters (c) and (d) over the clusters (a) and (b). This is because, the relative closeness of clusters (c) and (d) is higher than the relative closeness between clusters (a) and (b), even though the later pair of clusters have a higher absolute closeness. Thus, by looking at the relative closeness, CHAMELEON correctly prefers to merge clusters whose resulting cluster exhibits a uniformity in the degree of closeness between the items in the cluster. Also note that the relative closeness between two clusters is in general smaller than one, because the edges that connect vertices in different clusters have a smaller weight.

4.4 CHAMELEON: A Two-phase Clustering Algorithm

The dynamic framework for modeling the similarity between clusters discussed in Section 4.3 can be only applied when each cluster contains a sufficiently large number of vertices (*i.e.*, data items). This is because in order to compute the relative inter-connectivity and relative closeness of clusters, CHAMELEON needs to compute the internal inter-connectivity and closeness of each cluster. Both of which cannot be accurately calculated for clusters containing only a few data points. For this reason, CHAMELEON uses an algorithm that consists of two distinct phases. The purpose of the first phase is to cluster the data items into a large number of sub-clusters that contain a sufficient number of items to allow dynamic modeling. The purpose of the second phase, is to discover the genuine clusters in the data set by using the dynamic modeling framework to merge together these sub-clusters in a hierarchical fashion. In the remainder of this section, we present the algorithms used for these two phases of CHAMELEON.

Phase I: Finding Initial Sub-clusters CHAMELEON finds the initial sub-clusters using a graph partitioning algorithm to partition the k-nearest neighbor graph of the data set into a large number of partitions such that the *edge-cut*, *i.e.*, the sum of the weight of the edges that straddle partitions, is minimized. Since each edge in the k-nearest neighbor graph represents the similarity among data points, a partitioning that minimizes the edge-cut effectively minimizes the relationship (affinity) among data points across the resulting partitions. The underlying assumption is that links within clusters will be stronger and more plentiful than links across clusters. Hence, the data in each partition are highly related to other data items in the same partition.

Recent research on graph partitioning has lead to the development of fast and accurate algorithms that are based on the multilevel paradigm [KK99a, KK99b]. Extensive experiments on graphs arising in many application domains have shown that multilevel graph partitioning algorithms are very effective in capturing the *global structure* of the graph and are capable of computing partitionings that have a very small edge-cut. Hence, when used to partition the k-nearest neighbor graph, they are very effective in finding the natural separation boundaries of clusters. For example, Figure 8 shows the two clusters produced by applying a multilevel graph partitioning algorithm on the k-nearest-neighbor graphs for two spatial data sets. As we can see from this figure, the partitioning algorithm is very effective in finding the low-density separating region in the first example, and the small connecting region in the second example.



(a) **(b)**

Figure 8: An example of the bisections produced by multilevel graph partitioning algorithms on two spatial data sets. (a) The partitioning algorithm cuts through the sparse region. (b) The partitioning algorithms cuts through a small connecting region.

CHAMELEON utilizes such multilevel graph partitioning algorithms to find the initial sub-clusters. In particular, it uses the graph partitioning algorithm that is part of the hMETIS library [KK98a]. hMETIS has been shown [KK98c, KK99b, Alp98] to quickly produce high-quality partitionings for a wide range of unstructured graphs and hypergraphs.

In CHAMELEON we primarily use hMETIS to split a cluster C_i into two sub-clusters C_i^A and C_i^B such that the edge-cut between C_i^A and C_i^B is minimized and each one of these sub-clusters contains at least 25% of the nodes in C_i. Note that this last requirement, often referred to as the *balance constraint*, is an integral part of using a graph partitioning approach to find the sub-clusters. hMETIS is effective in operating within the allowed balance constraints to find a bisection that minimizes the edge-cut. However, this balance constraint can force hMETIS to break a natural cluster.

CHAMELEON obtains the initial set of sub-clusters as follows. It initially starts with all the points belonging to the same cluster. It then repeatedly selects the largest sub-cluster among the current set of sub-clusters and uses hMETIS to bisect. This process terminates when the larger sub-cluster contains fewer than a specified number of vertices, that we will refer to it as MINSIZE. The MINSIZE parameter essentially controls the granularity of the initial clustering solution. In general, MINSIZE should be set to a value that is smaller than the size of most of the clusters that we expect to find in the data set. At the same time, MINSIZE should be sufficiently large such that most of the sub-clusters contain a sufficiently large number of nodes to allow us to evaluate the inter-connectivity and closeness of the items in each sub-cluster in a meaningful fashion. For most of the data sets that we encountered, setting MINSIZE to about 1% to 5% of the overall number of data points worked fairly well.

Phase II: Merging Sub-Clusters using a Dynamic Framework As soon as the fine-grain clustering solution produced by the partitioning-based algorithm of the first phase is found, CHAMELEON then switches to an agglomerative hierarchical clustering that combines together these small sub-clusters. As discussed in Section 2, the key step of agglomerative hierarchical algorithm is that of finding the pair of sub-clusters that are the *most similar*.

CHAMELEON's agglomerative hierarchical clustering algorithm utilizes the dynamic modeling framework discussed in Section 4.3 to select the most similar pairs of clusters by looking both at their relative inter-connectivity and their relative closeness. There are many ways to develop an agglomerative hierarchical clustering algorithm that takes into account both of these measures. Two different schemes have been implemented in CHAMELEON.

The first scheme merges only those pairs of clusters whose relative inter-connectivity and relative closeness are both above some user specified threshold T_{RI} and T_{RC}, respectively. In this approach, CHAMELEON visits each cluster C_i, and checks to see if any one of its adjacent clusters C_j satisfy the following two conditions:

$$RI(C_i, C_j) \geq T_{RI} \quad \text{and} \quad RC(C_i, C_j) \geq T_{RC}. \tag{3}$$

If more than one of the adjacent clusters satisfy the above conditions, then CHAMELEON selects to merge C_i with the cluster that it is most connected to; *i.e.*, it selects the cluster C_j such that the absolute inter-connectivity between these two clusters is the highest. Once every cluster has been given the opportunity to merge with one of its adjacent clusters, the combinations that have been selected are performed, and the entire process is repeated. Note that this algorithm is different than traditional hierarchical clustering algorithms, as it allows multiple pairs of clusters to be merged together at any given iteration. The parameters T_{RI} and T_{RC} can be used to control the characteristics of the desired clusters. In particular, the parameter T_{RI} allows us to control the variability in the degree of inter-connectivity of the items in the cluster. The parameter T_{RC} allows us to control the uniformity of the similarity among items that belong to a particular cluster. Depending on the choice of the T_{RI} and T_{RC} parameters, CHAMELEON's merging algorithm may reach a point from which it cannot proceed any further because none of the adjacent clusters satisfy the two conditions of Equation 3. At this point we have the choice of either terminating the algorithm and output the current clustering as the solution or try to merge additional pairs of clusters by successively relaxing the two parameters, possibly at different rates.

The second scheme implemented in CHAMELEON uses a function to combine the relative inter-connectivity and relative closeness, and then selects to merge the pair of clusters that maximizes this function. Since our goal is to merge together pairs for which both the relative inter-connectivity and the relative closeness are high, a natural way of defining such a function is to take their product. That is, select the pair of clusters C_i and C_j to merge that maximize $RI(C_i, C_j) * RC(C_i, C_j)$. This formula gives an equal importance to both of these parameters. However, quite often we may prefer clusters that give a higher preference to one of these two measures. For this reason, CHAMELEON selects the pair of clusters that maximizes

$$RI(C_i, C_j) * RC(C_i, C_j)^{\alpha}, \tag{4}$$

where α is a user specified parameter. If $\alpha > 1$, then CHAMELEON gives a higher importance to the relative closeness, and when $\alpha < 1$, it gives a higher importance on the relative inter-connectivity. In the experimental results presented in Section 5 we used this second approach as it allows us to easily generate the entire dendrogram for the hierarchical clustering.

4.5 Performance Analysis

The overall computational complexity of CHAMELEON depends on the amount of time it requires to construct the k-nearest neighbor graph and the amount of time it requires to perform the two phases of the clustering algorithm.

The amount of time required to compute the k-nearest neighbor graph depends on the dimensionality of the underlying data set. In particular, for low-dimensional data sets, algorithms based on $k - d$ trees [Sam90] can be used to quickly compute the k nearest neighbors. It has been shown that for n items, the average cost of inserting, as well as the expected k-nearest neighbor search time is $O(\log n)$ [FBF77], leading to an overall complexity of $O(n \log n)$. However, for high dimensional data sets, schemes based on $k - d$ trees are not applicable [BBKK97, BBK98]. For such data sets, the amount of time required to find the k-nearest neighbors of a data item is $O(n)$, leading to an overall complexity of $O(n^2)$.

The amount of time required by CHAMELEON's two-phase clustering algorithm depends on the number m of initial sub-clusters produced by the graph partitioning algorithm used in the first phase. To simplify the analysis, we will assume that (i) each initial sub-cluster has the same number of nodes n/m, and (ii) during each successive merging step, CHAMELEON selects to merge only a single pair of clusters. Moreover, our analysis will be focused on CHAMELEON's second scheme for combining the relative inter-connectivity and relative closeness described in Section 4.4. However, the overall complexity is similar for the first scheme as well.

The amount of time required by the the Phase I of CHAMELEON depends on the amount of time required by hMETIS. Given a graph $G = (V, E)$, hMETIS requires $O(|V| + |E|)$ [KK98c, KK99b] time to compute a bisection. Since CHAMELEON operates on the k-nearest neighbor graph, $|E| = O(|V|)$; thus, the computational complexity of hMETIS is $O(|V|)$. CHAMELEON's Phase I algorithm obtains m clusters by repeatedly partitioning successively smaller graphs; hence, its overall computational complexity is $O(n \log(n/m))$ which is bounded by $O(n \log n)$. Note that one can potentially use a faster partitioning algorithm to obtain the initial m clusters in time $O(n + m \log m)$ using the multilevel m-way partitioning algorithm described in [KK99b].

The amount of time required by the second phase depends on (a) the amount of time needed to compute the internal inter-connectivity and internal closeness for each initial as well as intermediate cluster, and (b) the amount of time needed to select the *most similar* pair of clusters to merge. Since the internal inter-connectivity and internal closeness for a particular cluster is computed by bisecting the corresponding k-nearest neighbor sub-graph of the cluster, its complexity is proportional to the number of items in each cluster. In particular, the amount of time required to bisect each one of the initial m clusters is $O(n/m)$, leading to an overall complexity of $O(n)$. Next, during each of the merging steps, CHAMELEON needs to bisect the resulting cluster, and there are a total of $m-1$ such steps (*i.e.*, until all the initial sub-clusters have been merged together). The worst case complexity is obtained when the merging algorithm repeatedly selects the same cluster and merges it with another; *i.e.*, it grows a single large cluster. This corresponds to the worst case because during each merging step, the algorithm needs to bisect a cluster that has the largest possible number of data items. In this case, the amount of time required to bisect the $m-1$ intermediate sub-clusters is $\sum_{i=2}^{m-1}(i \times n/m)$ which is $O(nm)$.

The overall amount of time required to find the *most similar* pair of clusters is $O(m^2 \log m)$ by using a heap-based priority queue. In the worst case, the initial clustering solution can be such that each cluster is connected to all the remaining clusters. In this case, it takes $O(m^2 \log m)$ time to insert the similarity of the $O(m^2)$ possible pairs of sub-clusters into the priority queue. Now, during each merging step, the pair residing at the top of the priority queue is selected, and the similarity of recently combined cluster to the remaining sub-clusters is updated. Each of these update operations requires $O(m \log m)$ time, leading to an overall complexity of $O(m^2 \log m)$, as a total of $m-1$ such updates needs to be performed (one for each pair of clusters that gets merged).

Thus, the overall complexity of CHAMELEON's two-phase clustering algorithm is $O(nm + n \log n + m^2 \log m)$.

5 Experimental Results

In this section, we present experimental evaluation of CHAMELEON, and compare its performance with a publicly available version of DBSCAN and a locally implemented version of CURE. Even though CHAMELEON is applicable to any data set for which a similarity matrix is available (or can be constructed), we chose to perform evaluation for data sets containing points in two dimensional space for two reasons. First, similar data sets have been used to evaluate the performance of other state-of-the art algorithms such as DBSCAN and CURE. Second, clusters in 2D data sets are easy to visualize, making the comparison of different schemes much easier. We do not report results of ROCK [GRS99] (and other interconnectivity based agglomerative schemes such as group averaging method [JD88]), as they tend to perform worse than algorithms such as CURE on metric space data sets. Many of these results are available at URL http://www.cs.umn.edu/~han/chameleon.html.

5.1 Data Sets

We experimented with five different data sets containing points in two dimensions whose geometric shape are shown in Figure 9. The first data set, DS1, has five clusters that are of different size, shape, and density, and contains noise points as well as special artifacts. The second data set, DS2, contains two clusters that are close to each other and different regions of the clusters have different densities. The third data set, DS3, has six clusters of different size, shape, and orientation, as well as random noise points and special artifacts such as streaks running across clusters. The fourth data set, DS4, has eight clusters of different shape, size, and orientation, some of which are inside the space enclosed by other clusters. Moreover, DS4 also contains random noise and special artifacts, such as a collection of points forming vertical streaks. Finally, the fifth data set, DS5, has eight clusters of different shape, size, density, and orientation, as well as random noise. A particularly challenging feature of this data set is that clusters are very close to each other and they have different densities. The size of these data sets ranges from 6,000 to 10,000 points, and their exact size is indicated in Figure 9. Note that DS1 was obtained from [GRS98], whereas we synthetically generated the remaining data sets.

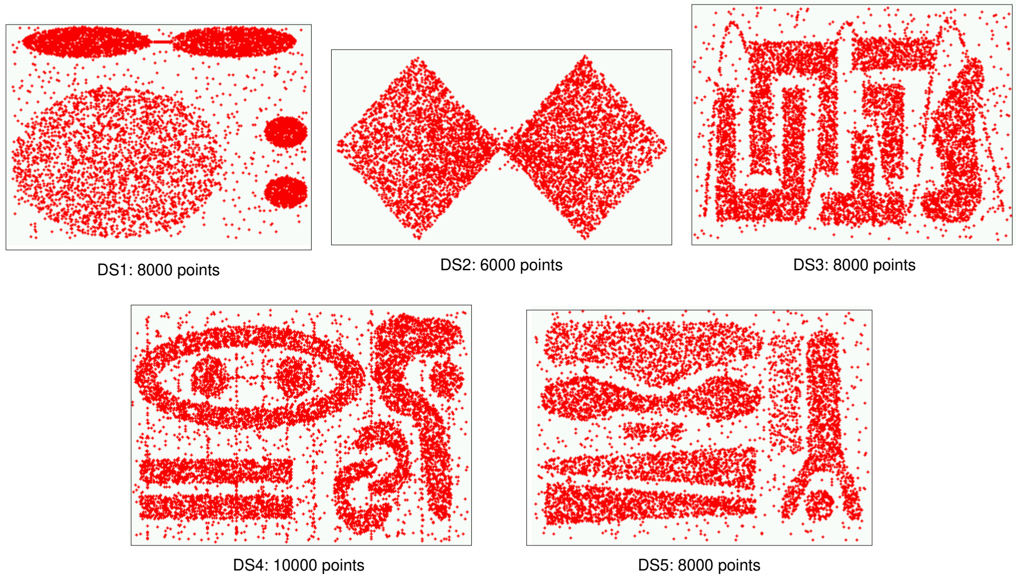


Figure 9: The five data sets used in our experiments.

5.2 Qualitative Comparison

CHAMELEON To cluster a data set using CHAMELEON, we need to specify the following parameters: the value of k for computing the k-nearest neighbor graph, the value of MINSIZE for the Phase I of the algorithm, and the choice of scheme for combining relative inter-connectivity and relative closeness and associated parameters. In the experiments presented in this section, we used the same set of parameter values for all five data sets. In particular, we used $k = 10$, MINSIZE $= 2.5\%$ of the total items in the data set, and used the second scheme for combining RI and RC, and used $\alpha = 2.0$ in Equation 4 for combining relative inter-connectivity and relative closeness of each pair of clusters. We also performed a parameter study to determine the sensitivity of CHAMELEON on the above set of parameters by using $k = \{5, 10, 15, 20\}$, MINSIZE $= \{2\%, 3\%, 4\%\}$ and $\alpha = \{1.5, 2.0, 2.5, 3.0\}$. Our results (not shown here) show that CHAMELEON is not very sensitive on the above choice of parameters, and it was able to discover the correct clusters for all of these combinations of values for k, MINSIZE, and α.

Figure 10 shows the clusters found by CHAMELEON for each one of the five data sets. The points in the different clusters are represented using a combination of different colors and different glyphs. As a result, points that belong to the same cluster have both the same color as well as their points are drawn using the same glyph. For example, in the clustering solution shown for DS4, there are two clusters that have cyan color (one contains the points in the region between the two circles inside the ellipse, and the other contains the points that form a line between the two horizontal bars and the 'c' shaped cluster), and there are two clusters that have a dark blue color (one corresponds to the upside-down 'c' shaped cluster and the other corresponds to the circle inside the candy-cane); however, their points are represented using different glyphs (bells and squares for the first pair, and squares and bells for the second pair), so they denote different clusters.

Since CHAMELEON is hierarchical in nature, it creates a dendrogram of possible clustering solutions at different levels of granularity. The clustering solutions shown in Figure 10 correspond to the earliest point in the agglomerative process in which CHAMELEON was able to find the genuine clusters in the data set. That is, they correspond to the lowest level of the dendrogram at which the genuine clusters in the data set have been identified and each one has been placed together in one cluster. As a result, the number of clusters shown in Figure 10 for each one of the data sets can be larger than the number of genuine clusters, and these additional clusters contain points that are outliers.

Looking at Figure 10, we can see that CHAMELEON is able to correctly identify the genuine clusters in all five data sets. In the case of DS1, CHAMELEON finds six clusters, five of which correspond to the genuine clusters in the data set, and the sixth one (shown with brown-colored '*' glyphs) corresponds to outlier points connecting the two ellipsoid clusters. In the case of DS2, CHAMELEON finds two clusters, each one corresponding to a genuine cluster in the data set. In the case of DS3, CHAMELEON finds eleven clusters, out of which six of them correspond to the genuine clusters in the data set, and the rest contain outliers. In the case of DS4, CHAMELEON also finds eleven clusters, out of which nine of them correspond to the genuine clusters, and the rest contain outlier points. Finally, in the case of DS5, CHAMELEON finds eight clusters, each one corresponding to a genuine cluster in the data set. As these experiment illustrate CHAMELEON is very effective in finding clusters of arbitrary shape, density, and orientation, and is tolerant to outlier points, as well as artifacts such as streaks running across clusters.

CURE We evaluated the performance of CHAMELEON against CURE (described in Section 2) which has been shown to be effective in finding clusters in two dimensional point data sets [GRS98]. CURE was able to find the right clusters for DS1 and DS2, but it failed to find the right clusters on the remaining three data sets. Figure 11 shows the results obtained by CURE for each one of the DS3, DS4, and DS5 data sets. Since CURE is also hierarchical clustering algorithm, it also produces a dendrogram of possible clustering solutions at different levels of granularity. For each one of the data sets, Figure 11 shows two different clustering solutions containing different number of clusters. The first clustering solution (first column of Figure 11) corresponds to the earliest point in the agglomerative process in which CURE merges together sub-clusters that belong to two different genuine clusters. As we can see from Figure 11, in the case of DS3, CURE selects the wrong pair of clusters to merge it together when going from 18 down to 17 clusters, resulting in the red-colored sub-cluster which contains portions of the two π-shaped clusters. Similarly, in the case of DS4, CURE makes a mistake when going from 26 down to 25 clusters, as it selects to merge together one of the circles inside the ellipse with a portion of the ellipse. Finally, in the case of DS5, CURE also makes a mistake when going from 26 down to 25 clusters by merging together the small circular cluster with a portion of the upside-down 'Y'-shaped cluster. The second clustering solution corresponds to solutions that contain as many clusters as those discovered by CHAMELEON. These solutions are considerably worse than the first set of solutions (especially for DS4 and DS5), indicating that the merging scheme used by CURE performs multiple mistakes.

For the results shown in Figure 11 experiments, the shrinking factor is 0.3 and the number of representative points is 10, which are the default values recommended in [GRS98]. We also performed experiments with shrinking factor varying from 0.1 to 0.9 and the number of representative points varying from 10 to 100. These experiments showed similar trends as shown in Figure 11. Furthermore, to facilitate fair comparisons, we also removed the noise as suggested in [GRS98], by identifying "slowly" growing clusters as noise point and removed them. For comparison purposes we reassigned these noisy data points back to the final clusters using the assignment method discussed in [GRS98], *i.e.*, noise points are assigned to the cluster with the closest representative points. Note that these assignments did not affect the overall clustering results.

DBSCAN DBSCAN [EKSX96] is a well-known spatial clustering algorithm that has been shown to find clusters of arbitrary shapes. DBSCAN defines a cluster to be a maximum set of density-connected points. Every core point in a cluster must have at least a minimum number of points (MinPts) within a given radius (Eps). DBSCAN can find arbitrary shape of clusters if the right density of the clusters can be determined in a priori and the density of clusters is uniform.

DBSCAN finds the right clusters on data sets DS3 and DS4 as long as it is supplied the right combination of Eps and MinPts. If MinPts is fixed to 4 (default value specified in [EKSX96]), then the algorithm works fine as long as Eps is within the range (5.0,5.4) for DS3 and (5.7,6.1) for DS4. However, it fails to perform well on DS1, DS2, and DS5, as these data sets contain clusters of different densities.

Figure 12 shows the clusters found by DBSCAN for DS1 and DS2 for different values of the *Eps* parameter. Following the recommendation of [EKSX96], the *MinPts* was fixed to 4 and *Eps* was changed in these experiments. The clusters produced for DS1 illustrate that DBSCAN cannot effectively find clusters of different density. In the first clustering solution (Figure 12(a)), when *Eps = 0.5*, DBSCAN puts the two ellipses into the same cluster, because the outlier points connecting satisfy the density requirements as dictated by the *Eps* and *MinPts* parameters. These clusters can be separated by decreasing the value of *Eps* as was done in the clustering solution shown in Figure 12(b), for which *Eps=0.4*. However, DBSCAN keeps the ellipses together, but now it has fragmented the lower density cluster into a large number of small sub-clusters. Our experiments has shown that DBSCAN exhibits similar characteristics on DS5. The clusters produced for DS2 illustrate that DBSCAN cannot effectively find clusters that their internal density varies. The sequence of the three clustering solutions (Figure 12(c)–(e)) for decreasing values of the *Eps* parameter illustrates that as we decrease *Eps* in hope of separating the two clusters, the natural clusters in the data set are fragmented into a large number of smaller clusters. On DS3 and DS4, DBSCAN managed to find the genuine clusters with right parameter values. Figure 12 (f)–(h) shows the sensitivity of DBSCAN with respect to the Eps parameter.

6 Concluding Remarks

In this paper, we have presented a novel hierarchical clustering algorithm called CHAMELEON which takes into account the dynamic model of clusters. CHAMELEON can discover natural clusters of different shapes and sizes, because its merging decision dynamically adapts to the different clustering model characterized by the clusters in consideration. Experimental results on several data sets with varying characteristics show that CHAMELEON can discover natural clusters that many existing clustering algorithms fail to find.

All the data sets presented in the paper are in 2D space, partly because similar data sets have been used most extensively by other authors [NH94, EKSX96, GRS98], and partly because it is easy to evaluate the quality of clustering on 2D data sets. Note that many of these schemes [EKSX96, GRS98] are specifically suited for spatial data and/or data in metric spaces. Hence, it is noteworthy that our scheme outperforms them, even though it does not make use of the metric-space/spatial nature of the data. The methodology of dynamic modeling of clusters in agglomerative hierarchical methods is applicable to all types of data as long as a similarity matrix is available or can be constructed.

Even though we chose to model the data using k-nearest neighbor graph in this paper, it is entirely possible to use other graph representations suitable for particular application domains, e.g., such as those based upon mutual shared neighbors [GK78, JD88, GRS99]. Furthermore, different domains may require different models for capturing relative closeness and inter-connectivity of pairs of clusters. In any of these situations, we believe that the two-phase framework of CHAMELEON would still be highly effective. Our future research includes the verification of CHAMELEON on different application domains and the study of effectiveness of different techniques for modeling data as well as cluster similarity.

In this paper, we ignored the issue of scaling to large data sets that cannot fit in the main memory. These issues are orthogonal to the ones discussed here and are covered in [ZRL96, BFR98, GRS98, GRG$^+$99].

Acknowledgements

We would like to thank M. Ester, H.P. Kriegel, J. Sander, and X. Xu for providing the DBSCAN program.

References

[Alp98] C. J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. of the Intl. Symposium of Physical Design*, pages 80–85, 1998.

[BBK98] S. Berchtold, C. Bohm, and H.-P. Kriegel. They pyramid-technique: Towards breaking the curse of dimensionality. In *Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data*, 1998.

[BBKK97] S. Berchtold, C. Bohm, D. Keim, and H.-P. Kriegel. Cost model for nearest neighbor search in high-dimensional data space. In *Proc. of Symposium on Principles of Database Systems*, Tucson, Arizona, 1997.

[BFR98] P.S. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, 1998.

[BGG+99a] D. Boley, M. Gini, R. Gross, E.H. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Document categorization and query generation on the world wide web using WebACE. *AI Review (accepted for publication)*, 1999.

[BGG+99b] D. Boley, M. Gini, R. Gross, E.H. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Partitioning-based clustering for web document categorization. *Decision Support Systems (accepted for publication)*, 1999.

[BH64] G.H. Ball and D.J. Hall. Some fundamental concepts and sysnthesis procedures for pattern recognition preprocessors. In *International Conference on Microwaves, Circuit Theory, and Information Theory*, 1964.

[BR98] Simon Byers and Adrian E. Raftery. Nearest neighbor clutter removal for estimating features in spatial point processes. *J. American Statistical Association*, 93:577–584, June 1998.

[CHY96] M.S. Chen, J. Han, and P.S. Yu. Data mining: An overview from database perspective. *IEEE Transactions on Knowledge and Data Eng.*, 8(6):866–883, December 1996.

[CS96] P. Cheeseman and J. Stutz. Baysian classification (autoclass): Theory and results. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI/MIT Press, 1996.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the Second Int'l Conference on Knowledge Discovery and Data Mining*, Portland, OR, 1996.

[FBF77] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.

[GK78] K.C. Gowda and G. Krishna. Agglomerative clustering using the concept of mutual nearest neighborhood. *Pattern Recognition*, 10:105–112, 1978.

[GRG+99] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French. Clustering large datasets in arbitrary metric spaces. In *Proc. of the 15th Int'l Conf. on Data Eng.*, 1999.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data*, 1998.

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

[HHS92] N. Harris, L. Hunter, and D. States. Mega-classification: Discovering motifs in massive datastreams. In *Proceedings of the Tenth International Conference on Artificial Intelligence (AAAI)*, 1992.

[HKKM97] E.H. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. Technical Report TR-97-019, Department of Computer Science, University of Minnesota, Minneapolis, 1997.

[HKKM98] E.H. Han, G. Karypis, V. Kumar, and B. Mobasher. Hypergraph based clustering in high-dimensional data sets: A summary of results. *Bulletin of the Technical Committee on Data Engineering*, 21(1), 1998.

[JD88] A.K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[JP73] R.A. Jarvis and E.A. Patrick. Clustering using a similarity measure based on shared nearest neighbors. *IEEE Transactions on Computers*, C-22(11), November 1973.

[KK98a] G. Karypis and V. Kumar. hMETIS 1.5: A hypergraph partitioning package. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/˜metis*.

[KK98b] G. Karypis and V. Kumar. METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/˜metis*.

[KK98c] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998. Also available on WWW at URL http://www.cs.umn.edu/˜karypis.

[KK99a] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1), 1999. Also available on WWW at URL http://www.cs.umn.edu/˜karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.

[KK99b] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of the Design and Automation Conference*, 1999.

[KR90] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[NH94] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. of the 20th VLDB Conference*, pages 144–155, Santiago, Chile, 1994.

[NRS+95] T. Newman, E.F. Retzel, E. Shoop, E. Chi, and C. Somerville. Arabidopsis thaliana expressed sequence tags: Generation, analysis and dissemination. In *Plant Genome III: International Conference on the Status of Plant Genome Research*, San Diego, CA, 1995.

[PAS96] L.J. Hubert P. Arabie and G. De Soete. *Clustering and Classification*. World Scientific, 1996.

[SAD+93] M. Stonebraker, R. Agrawal, U. Dayal, E. J. Neuhold, and A. Reuter. DBMS research at a crossroads: The vienna update. In *Proc. of the 19th VLDB Conference*, pages 688–692, Dublin, Ireland, 1993.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[SCC+95] E. Shoop, E. Chi, J. Carlis, P. Bieganski, J. Riedl, N. Dalton, T. Newman, and E. Retzel. Implementation and testing of an automated EST processing and analysis system. In Lawrence Hunter and Bruce Shriver, editors, *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, volume 5, pages 52–61. IEEE Computer Society Press, 1995.

[XEKS98] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander. A distribution-based clustering algorithm for mining large spatial databases. In *Proc. of the 14th Int'l Conf. on Data Eng.*, 1998.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Linvy. Birch: an efficient data clustering method for large databases. In *Proc. of 1996 ACM-SIGMOD Int. Conf. on Management of Data*, Montreal, Quebec, 1996.

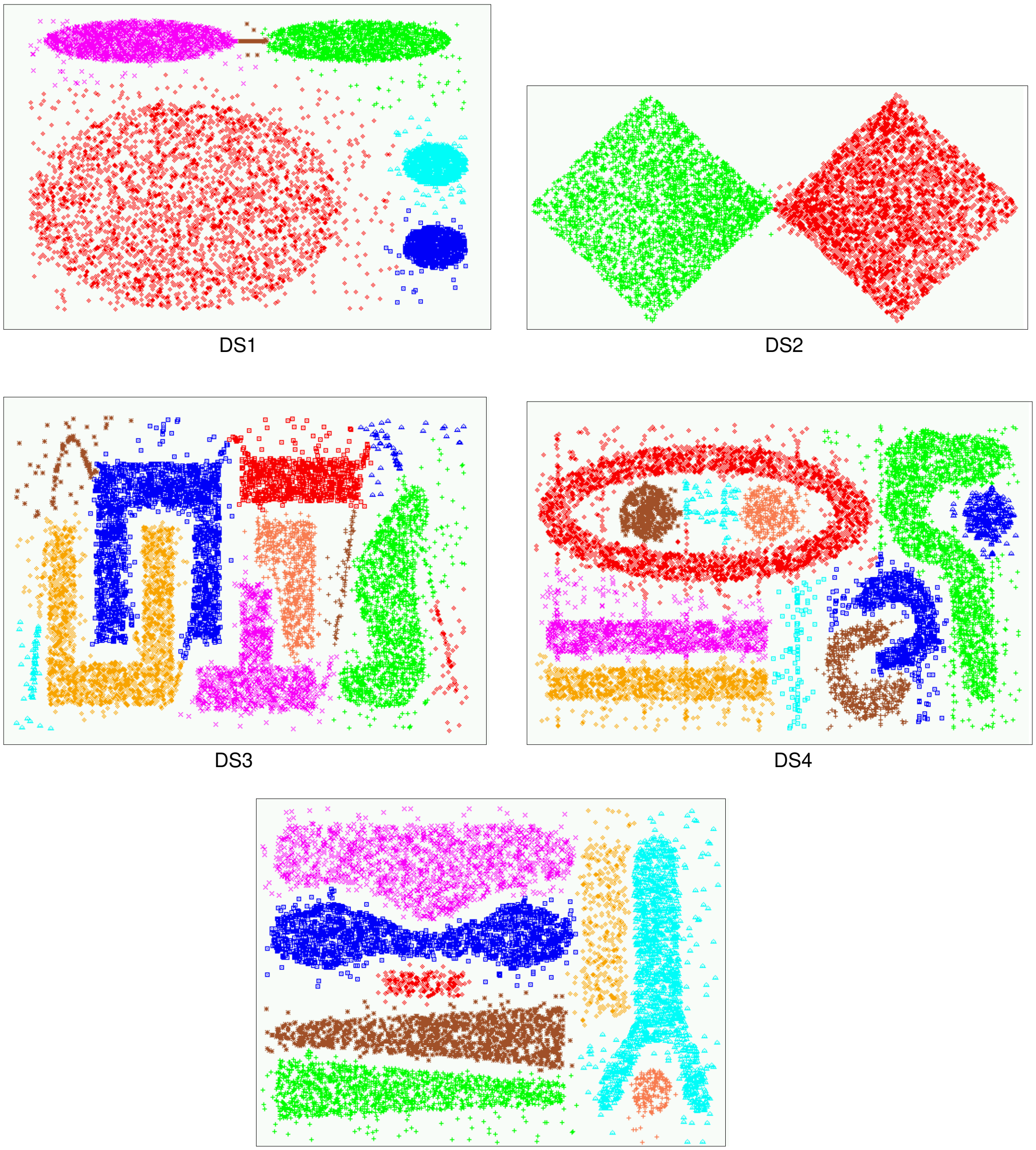


Figure 10: The clusters discovered by CHAMELEON for the five data sets.

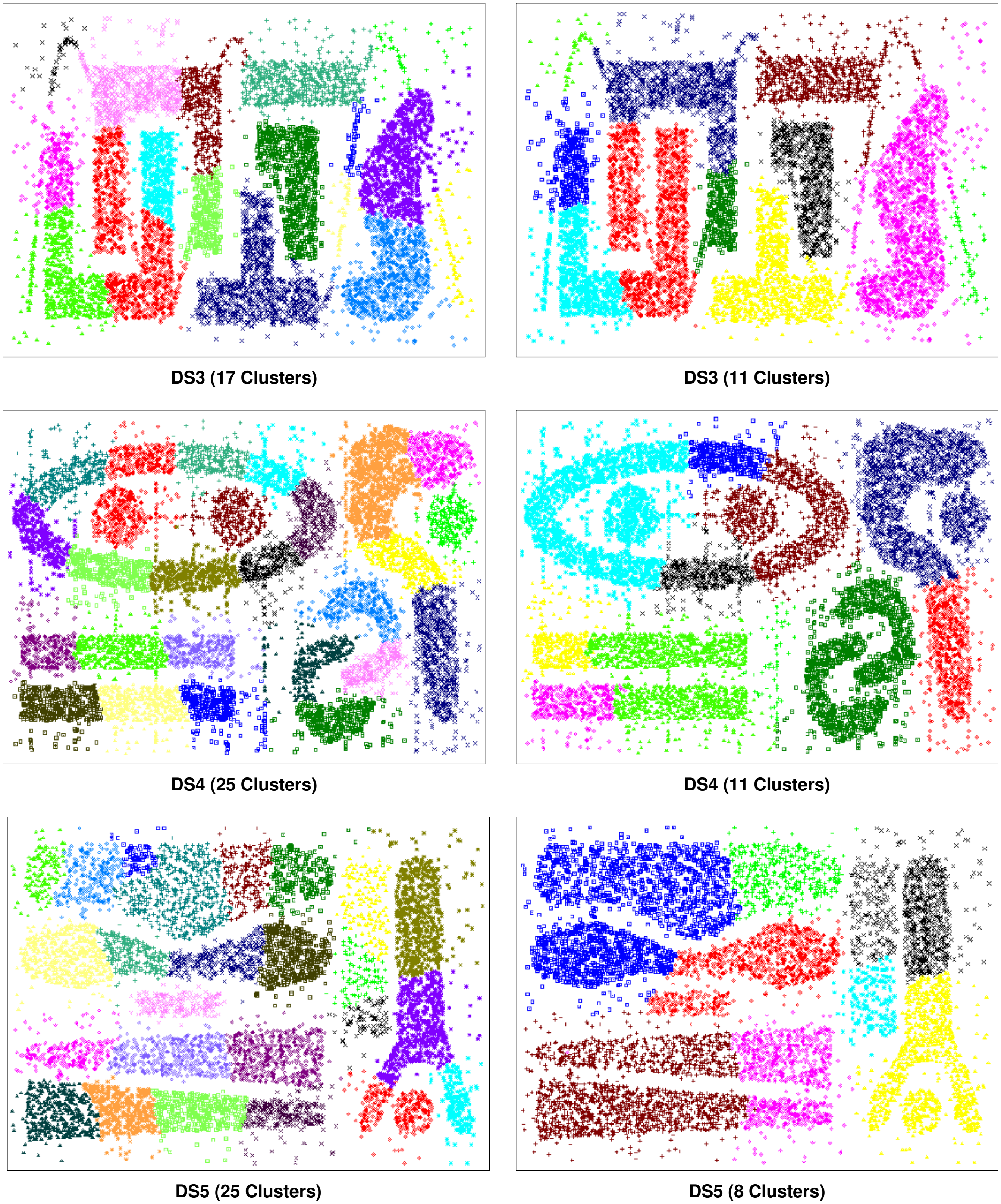
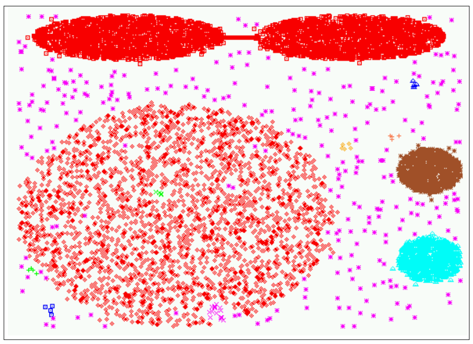
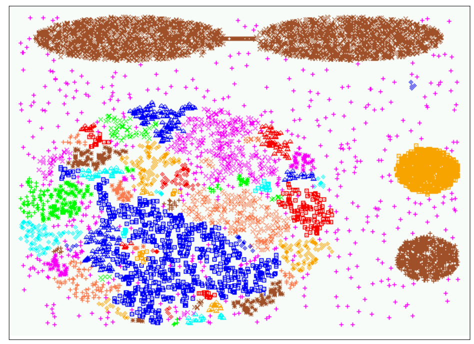


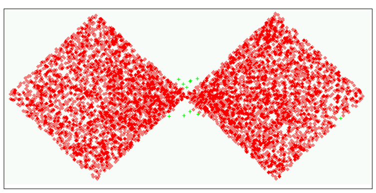
Figure 11: Clusters of CURE with shrinking factor 0.3 and number of representative points 10.



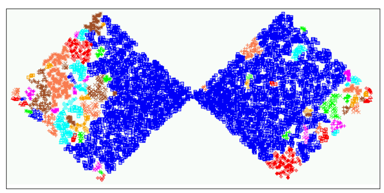


(a) DS1: Eps=0.5, MinPts=4

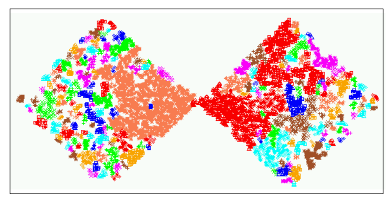
(b) DS1: Eps=0.4, MinPts=4



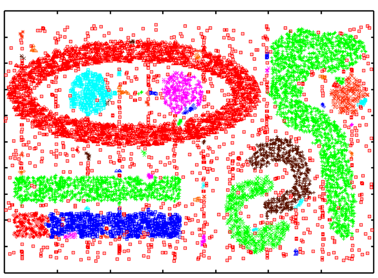
(c) DS2: Eps=5.0, MinPts=4



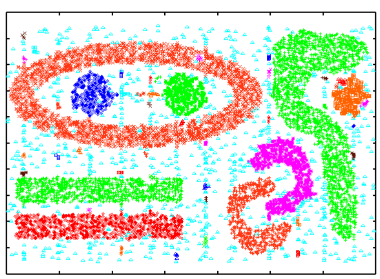
(d) DS2: Eps=3.5, MinPts=4



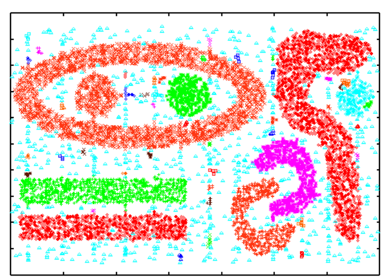
(e) DS2: Eps=3.0, MinPts=4



(f) DS4: Eps=5.5, MinPts=4



(g) DS4: Eps=5.9, MinPts=4



(h) DS4: Eps=6.2, MinPts=4

Figure 12: DBSCAN on the DS1, DS2, and DS4 data sets with different values of the *Eps* parameter.