CHAMELEON A hierarchical clustering algorithm using dynamic modeling

Article *in* Computer · September 1999

DOI: 10.1109/2.781637 · Source: IEEE Xplore

3 authors:

George Karypis

University of Minnesota Twin Cities

435 PUBLICATIONS **39,854** CITATIONS

SEE PROFILE

Eui-Hong Sam Han

Persistent Systems Ltd

32 PUBLICATIONS **3,640** CITATIONS

SEE PROFILE

Vipin Kumar

University of Minnesota Twin Cities

556 PUBLICATIONS **35,823** CITATIONS

SEE PROFILE

Chameleon: Hierarchical Clustering Using Dynamic Modeling

Many advanced algorithms have difficulty dealing with highly variable clusters that do not follow a preconceived model. By basing its selections on both interconnectivity and closeness, the Chameleon algorithm yields accurate results for these highly variable clusters.

George Karypis
Eui-Hong (Sam) Han
Vipin Kumar
University of Minnesota

Clustering is a discovery process in data mining.[1] It groups a set of data in a way that maximizes the similarity within clusters and minimizes the similarity between two different clusters.[1,2] These discovered clusters can help explain the characteristics of the underlying data distribution and serve as the foundation for other data mining and analysis techniques. Clustering is useful in characterizing customer groups based on purchasing patterns, categorizing Web documents,[3] grouping genes and proteins that have similar functionality,[4] grouping spatial locations prone to earthquakes based on seismological data, and so on.

Most existing clustering algorithms find clusters that fit some static model. Although effective in some cases, these algorithms can break down—that is, cluster the data incorrectly—if the user doesn't select appropriate static-model parameters. Or sometimes the model cannot adequately capture the clusters' characteristics. Most of these algorithms break down when the data contains clusters of diverse shapes, densities, and sizes.

Existing algorithms use a static model of the clusters and do not use information about the nature of individual clusters as they are merged. Furthermore, one set of schemes (the CURE algorithm and related schemes) ignores the information about the aggregate interconnectivity of items in two clusters. The other set of schemes (the Rock algorithm, group averaging method, and related schemes) ignores information about the closeness of two clusters as defined by the similarity of the closest items across two clusters. (For more information, see the "Limitations of Traditional Clustering Algorithms" sidebar.)

By only considering either interconnectivity or closeness, these algorithms can easily select and merge the wrong pair of clusters. For instance, an algorithm that focuses only on the closeness of two clusters will incorrectly merge the clusters in Figure 1a over those in Figure 1b. Similarly, an algorithm that focuses only on interconnectivity will, in Figure 2, incorrectly merge the dark-blue with the red cluster rather than the green one. Here, we assume that the aggregate interconnectivity between the items in the dark-blue and red clusters is greater than that of the dark-blue and green clusters. However, the border points of the dark-blue cluster are much closer to those of the green cluster than those of the red cluster.

CHAMELEON: CLUSTERING USING DYNAMIC MODELING

Chameleon is a new agglomerative hierarchical clustering algorithm that overcomes the limitations of existing clustering algorithms. Figure 3 (on page 70) provides an overview of the overall approach used by Chameleon to find the clusters in a data set.

The Chameleon algorithm's key feature is that it accounts for both interconnectivity *and* closeness in identifying the most similar pair of clusters. It thus avoids the limitations discussed earlier. Furthermore, Chameleon uses a novel approach to model the degree of interconnectivity and closeness between each pair of clusters. This approach considers 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 merged clusters.

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 allows Chameleon to scale to large data sets and to successfully use data sets that

0018-9162/99/$10.00 © 1999 IEEE

are available only in similarity space and not in metric spaces.[5] Data sets in a metric space have a fixed number of attributes for each data item, whereas data sets in a similarity space only provide similarities between data items.

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 several relatively small subclusters. During the second phase, it uses an algorithm to find the genuine clusters by repeatedly combining these subclusters.

Modeling the data

Given a matrix whose entries represent the similarity between data items, many methods can be used to find a graph representation.[2,8] In fact, modeling data items as a graph is very common in many hierarchical clustering algorithms.[2,8,9] Chameleon's sparse-graph representation of the items is based on the commonly used *k*-nearest-neighbor graph approach. Each vertex of the *k*-nearest-neighbor graph represents a data item. An edge exists between two vertices *v* and *u* if *u* is among the *k* most similar points of *v*, or *v* is among the *k* most similar points of *u*.

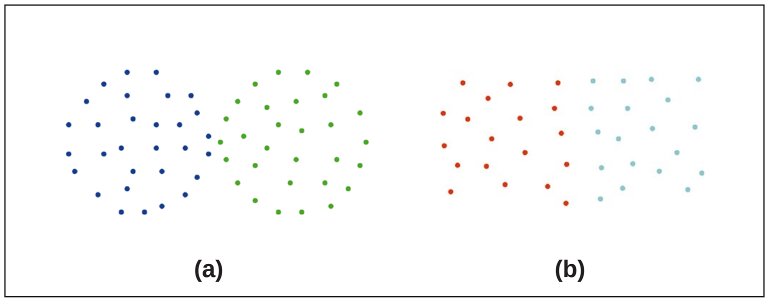

Figure 1. Algorithms based only on closeness will incorrectly merge (a) the dark-blue and green clusters because these two clusters are closer together than (b) the red and cyan clusters.

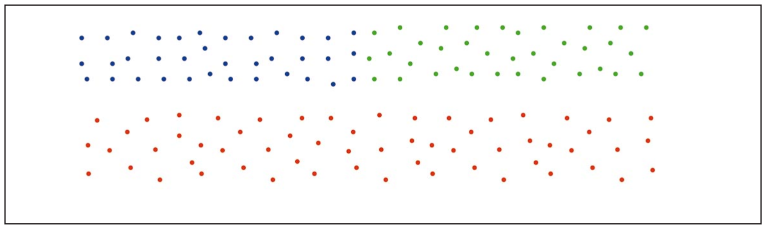

Figure 2. Algorithms based only on the interconnectivity of two clusters will incorrectly merge the dark-blue and red clusters rather than dark-blue and green clusters.

Limitations of Traditional Clustering Algorithms

Partition-based clustering techniques such as *K*-Means[2] and Clarans[6] attempt to break a data set into *K* clusters such that the partition optimizes a given criterion. These algorithms assume that clusters are hyper-ellipsoidal and of similar sizes. They can't find clusters that vary in size, as shown in Figure A1, or concave shapes, as shown in Figure A2.

DBScan[7] (Density-Based Spatial Clustering of Applications with Noise), a well-known spatial clustering algorithm, can find clusters of arbitrary shapes. DBScan defines a cluster to be a maximum set of *density-connected* points, which means that every core point in a cluster must have at least a minimum number of points (MinPts) within a given radius (Eps). DBScan assumes that all points within genuine clusters can be reached from one another by traversing a path of density-connected points and points across different clusters cannot. DBScan can find arbitrarily shaped clusters if the cluster density can be determined beforehand and the cluster density is uniform.

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[2] start with each data point as a separate cluster. Each step of the algorithm involves merging two clusters that are the most similar. After each merger, the total number of clusters decreases by one. Users can repeat these steps until they obtain the desired number of clusters or the distance between the two closest clusters goes above a certain threshold. The many variations of agglomerative hierarchical algorithms[2] primarily differ in how they update the similarity between existing and merged clusters.

In some hierarchical methods, each clus-

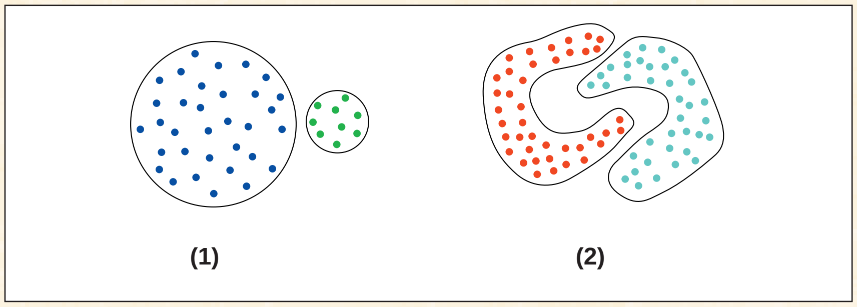

Figure A. Data sets on which centroid and medoid approaches fail: Clusters (1) of widely different sizes and (2) with concave shapes.

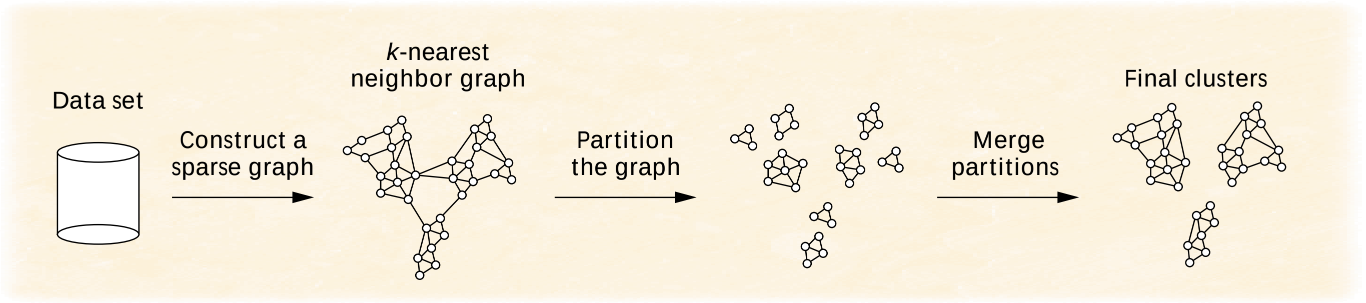

Figure 3. Chameleon uses a two-phase algorithm, which first partitions the data items into subclusters and then repeatedly combines these subclusters to obtain the final clusters.

Figure 4 illustrates the 1-, 2-, and 3-nearest-neighbor graphs of a simple data set. There are several advantages to representing the items to be clustered using a *k*-nearest-neighbor graph. Data items that are far apart are completely disconnected, and the weights on the edges capture the underlying population density of the space. Items in denser and sparser regions are modeled uniformly, and the sparsity of the representation leads to computationally efficient algorithms. Because Chameleon operates on a sparse graph, each cluster is nothing more than a subgraph of the data set's original sparse-graph representation.

Modeling cluster similarity

Chameleon uses a dynamic modeling framework to determine the similarity between pairs of clusters by looking at their relative interconnectivity (RI) and relative closeness (RC). Chameleon selects pairs to merge for which both RI and RC are high. That is, it selects clusters that are well interconnected as well as close together.

Relative interconnectivity. Clustering algorithms typically measure the *absolute* interconnectivity between clusters C_i and C_j in terms of *edge cut*—the sum of the weight of the edges that straddle the two clusters, which we denote $EC(C_i, C_j)$. *Relative* interconnectivity between clusters is their absolute interconnectivity normalized with respect to their internal interconnectivities. To get the cluster's *internal* interconnectivity, we sum the edges crossing a min-cut bisection that splits the cluster into two roughly equal parts. Recent advances in graph partitioning have made it possible to efficiently find such quantities.[10]

Thus, the relative interconnectivity between a pair of clusters C_i and C_j is

$$RI(C_i, C_j) = \frac{|EC(C_i, C_j)|}{\dfrac{|EC(C_i)| + |EC(C_j)|}{2}}$$

By focusing on relative interconnectivity, Chameleon

ter is represented by a centroid or medoid—a data point that is the closest to the center of the cluster—and the similarity between two clusters is measured by the similarity between the centroids/medoids. Both of these schemes 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 situation occurs in many natural clusters,[4,8] for example, if there is a large variation in cluster sizes, as in Figure A1, or when cluster shapes are concave, as in Figure A2.

The single-link hierarchical method[2] measures the similarity between two clusters 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, it is highly susceptible to noise, outliers, and artifacts.

Researchers have proposed CURE[9] (Clustering Using Representatives) to remedy the drawbacks of both of these methods while combining their advantages. In CURE, a cluster is represented by selecting a constant number of well-scattered points and shrinking them toward the cluster's centroid, according to a shrinking factor. CURE measures the similarity between two clusters by the similarity of the closest pair of points belonging to different clusters.

Unlike centroid/medoid-based methods, CURE can find clusters of arbitrary shapes and sizes, as it represents each cluster via multiple representative points. Shrinking the representative points toward the centroid allows CURE to avoid some of the problems associated with noise and outliers. However, these techniques fail to account for special characteristics of individual clusters. They can make incorrect merging decisions when the underlying data does not follow the assumed model or when noise is present.

In some algorithms, the similarity between two clusters is captured by the aggregate of the similarities (that is, the 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 interconnectivity between two clusters depends on the size of the clusters; in general, pairs of larger clusters will have higher interconnectivity.

Many such schemes normalize the aggregate similarity between a pair of clusters with respect to the expected interconnectivity of the clusters involved. For example, the widely used group-average method[2] assumes fully connected clusters, and thus scales the aggregate similarity between two clusters by $n \times m$, where n and m are the number of members in the two clusters.

Rock[8] (Robust Clustering Using Links), a recently developed algorithm that operates on a derived similarity graph, scales the aggregate interconnectivity with respect to a user-specified interconnectivity model. However, the major limitation of all such schemes is that they assume a static, user-supplied interconnectivity model. Such models are inflexible and can easily lead to incorrect merging decisions when the model under- or overestimates the interconnectivity of the data set or when different clusters exhibit different interconnectivity characteristics. Although some schemes allow the connectivity to vary for different problem domains (as does Rock[8]), it is still the same for all clusters irrespective of their densities and shapes.

can overcome the limitations of existing algorithms that use static interconnectivity models. Relative interconnectivity can account for differences in cluster shapes as well as differences in the degree of interconnectivity for different clusters.

Relative closeness. Relative closeness involves concepts that are analogous to those developed for relative interconnectivity. The *absolute closeness* of clusters is the average weight (as opposed to the sum of weights for interconnectivity) of the edges that connect vertices in C_i to those in C_j. Since these connections come from the *k*-nearest-neighbor graph, their average strength provides a good measure of the affinity between the data items along the interface layer of the two clusters. At the same time, this measure is tolerant of outliers and noise.

To get a cluster's *internal closeness*, we take the average of the edge weights across a min-cut bisection that splits the cluster into two roughly equal parts.

The *relative closeness* between a pair of clusters is the absolute closeness normalized with respect to the internal closeness of the two clusters:

$$RC(C_i, C_j) = \frac{\bar{S}EC(C_i, C_j)}{\frac{|C_i|}{|C_i|+|C_j|}\bar{S}EC(C_i) + \frac{|C_j|}{|C_i|+|C_j|}\bar{S}EC(C_j)}$$

where $\bar{S}EC(C_i)$ and $\bar{S}EC(C_j)$ are the average weights of the edges that belong in the min-cut bisector of clusters C_i and C_j, and $\bar{S}EC(C_i, C_j)$ is the average weight of the edges that connect vertices in C_i and C_j. Terms $|C_i|$ and $|C_j|$ are the number of data points in each cluster. This equation also normalizes the absolute closeness of the two clusters by the weighted average of the internal closeness of C_i and C_j. This discourages the merging of small sparse clusters into large dense clusters.

In general, the relative closeness between two clusters is less than one because the edges that connect vertices in different clusters have a smaller weight.

By focusing on the relative closeness, Chameleon can overcome the limitations of existing algorithms that look only at the absolute closeness. By looking at the relative closeness, Chameleon correctly merges clusters so that the resulting cluster has a uniform degree of closeness between its items.

Process

The dynamic framework for modeling cluster similarity is applicable only when each cluster contains a sufficiently large number of vertices (data items). The reason is that to compute the relative interconnectivity and closeness of clusters, Chameleon needs to compute each cluster's internal interconnectivity and closeness, neither of which can be accurately calculated for clusters containing a few data points. For this reason, Chameleon has a first phase that clusters the data items into several subclusters that contain a sufficient number of items to allow dynamic modeling. In a second phase, it discovers the genuine clusters in the data set by using the dynamic modeling framework to hierarchically merge these subclusters.

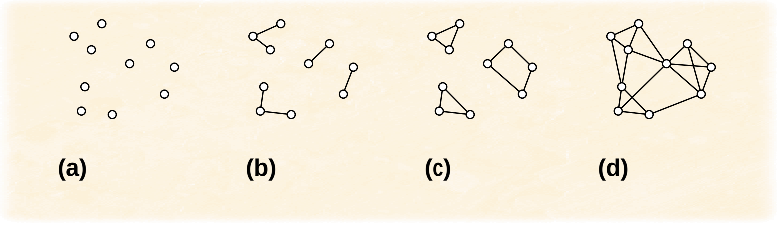

Figure 4. K-nearest-neighbor graphs from original data in two dimensions: (a) original data, (b) 1-, (c) 2-, and (d) 3-nearest neighbor graphs.

Chameleon finds the initial subclusters using hMetis,[10] a fast, high-quality graph-partitioning algorithm. hMetis partitions the *k*-nearest-neighbor graph of the data set into several partitions such that it minimizes the edge cut. 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 partitions.

After finding subclusters, Chameleon switches to an algorithm that repeatedly combines these small subclusters, using the relative interconnectivity and relative closeness framework. There are many ways to develop an algorithm that accounts for both of these measures. Chameleon uses two different schemes.

User-specified thresholds. The first merges only those pairs of clusters exceeding user-specified thresholds for relative interconnectivity (T_{RI}) and relative closeness (T_{RC}). In this approach, Chameleon visits each cluster C_i and checks for adjacent clusters C_j with RI and RC that exceed these thresholds.

If more than one adjacent cluster satisfies these conditions, then Chameleon will merge C_i with the cluster that it is most connected to—that is, the pair C_i and C_j with the highest absolute interconnectivity.

Once Chameleon chooses a partner for every cluster, it performs these mergers and repeats the entire process. Users can control the characteristics of the desired clusters with T_{RI} and T_{RC}.

Function-defined optimization. The second scheme implemented in Chameleon uses a function to combine the relative interconnectivity and relative closeness. Chameleon selects cluster pairs that maximize this function. Since our goal is to merge pairs for which both the relative interconnectivity and relative closeness are high, a natural way of defining such a function is to take their product. That is, select clusters that maximize $RI(C_i,C_j) \times RC(C_i,C_j)$. This formula gives equal importance to both 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

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

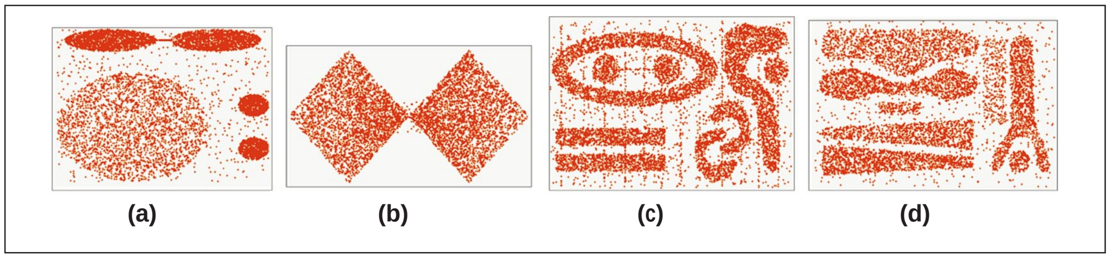

Figure 5. Four data sets used in our experiments: (a) DS1 has 8,000 points; (b) DS2 has 6,000 points; (c) DS3 has 10,000 points; and (d) DS4 has 8,000 points.

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 to the relative interconnectivity.

RESULTS AND COMPARISONS

We compared Chameleon's performance against that of CURE[9] and DBScan[7] on four different data sets. These data sets contain from 6,000 to 10,000 points in two dimensions; the points form geometric shapes as shown in Figure 5. These data sets represent some difficult clustering instances because they contain clusters of arbitrary shape, proximity, orientation, and varying densities. They also contain significant noise and special artifacts. Many of these results are available at http://www.cs.umn.edu/~han/chameleon.html.

Chameleon is applicable to any data set for which a similarity matrix is available (or can be constructed). However, we evaluated it with two-dimensional data sets because similar data sets have been used to evaluate other state-of-the-art algorithms. Clusters in 2D data sets are also easy to visualize and compare.

Figure 6 shows the clusters that Chameleon found for each of the four data sets, using the same set of parameter values. In particular, we used $k = 10$ in a function-defined optimization with $\alpha = 2.0$. Points in different clusters are represented using a combination of colors and glyphs. As a result, points that belong to the same cluster use both the same color and glyph.

For example, in the DS3 clusters, there are two cyan clusters. One contains the points in the region between the two circles inside the ellipse, and the other, the line between the two horizontal bars and the c-shaped cluster. Two clusters are dark blue—one corresponds to the upside-down c-shaped cluster and the other, the circle inside the candy cane. Their points use different glyphs (bells and squares for the first pair, and squares and

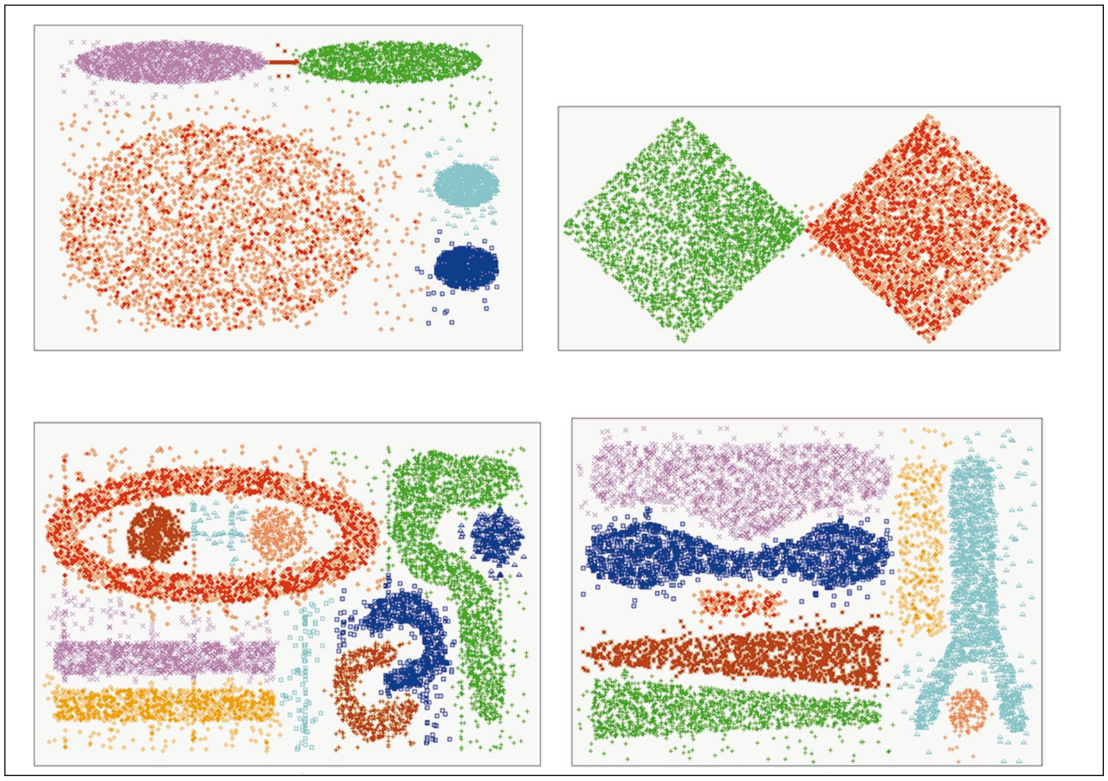

Figure 6. Clusters discovered by Chameleon for the four data sets.

bells for the second), so they denote different clusters.

The clustering shown in Figure 6 corresponds to the earliest point at which Chameleon was able to find the genuine clusters in the data set. That is, they correspond to the earliest iteration at which Chameleon identifies the genuine clusters and places each in a single cluster. Thus for a given data set, Figure 6 shows more than the number of genuine clusters; the additional clusters contain outliers.

Chameleon correctly identifies the genuine clusters in all four data sets. For example, in DS1, Chameleon finds six clusters, five of which are genuine clusters, and the sixth (shown in brown) corresponds to outliers that connect the two ellipsoid clusters. In DS3, Chameleon finds the nine genuine clusters and two clusters that contain outliers. In DS2 and DS4, Chameleon accurately finds only the genuine cluster.

CURE

We also evaluated CURE, which has been shown to effectively find clusters in two-dimensional data sets.[9] CURE was able to find the right clusters for DS1 and DS2, but it failed to find the right clusters on the remaining two data sets. Figure 7 shows the results obtained by CURE for the DS3 and DS4 data sets.

For each of the data sets, Figure 7 shows two different clustering solutions containing different numbers of clusters. The clustering solution in Figure 7a corresponds to the earliest point in the process in which CURE merges subclusters that belong to two different genuine clusters. As we can see from Figure 7b, in the case of DS3, CURE makes a mistake when going down to 25 clusters, as it merges one of the circles inside the ellipse with a portion of the ellipse.

Similarly, in the case of DS4 (Figure 7c), CURE also makes a mistake when going down to 25 clusters by merging the small circular cluster with a portion of the upside-down Y-shaped cluster. The second clustering solution (Figure 7d) corresponds to solutions that contain as many clusters as those discovered by Chameleon. These solutions are considerably worse than the first set of solutions, indicating that CURE's merging scheme makes multiple mistakes for these data sets.

DBSCAN

Figure 8 shows the clusters found by DBScan for DS1 and DS2, using different values of the Eps parameter. Following the recommendation of DBScan's

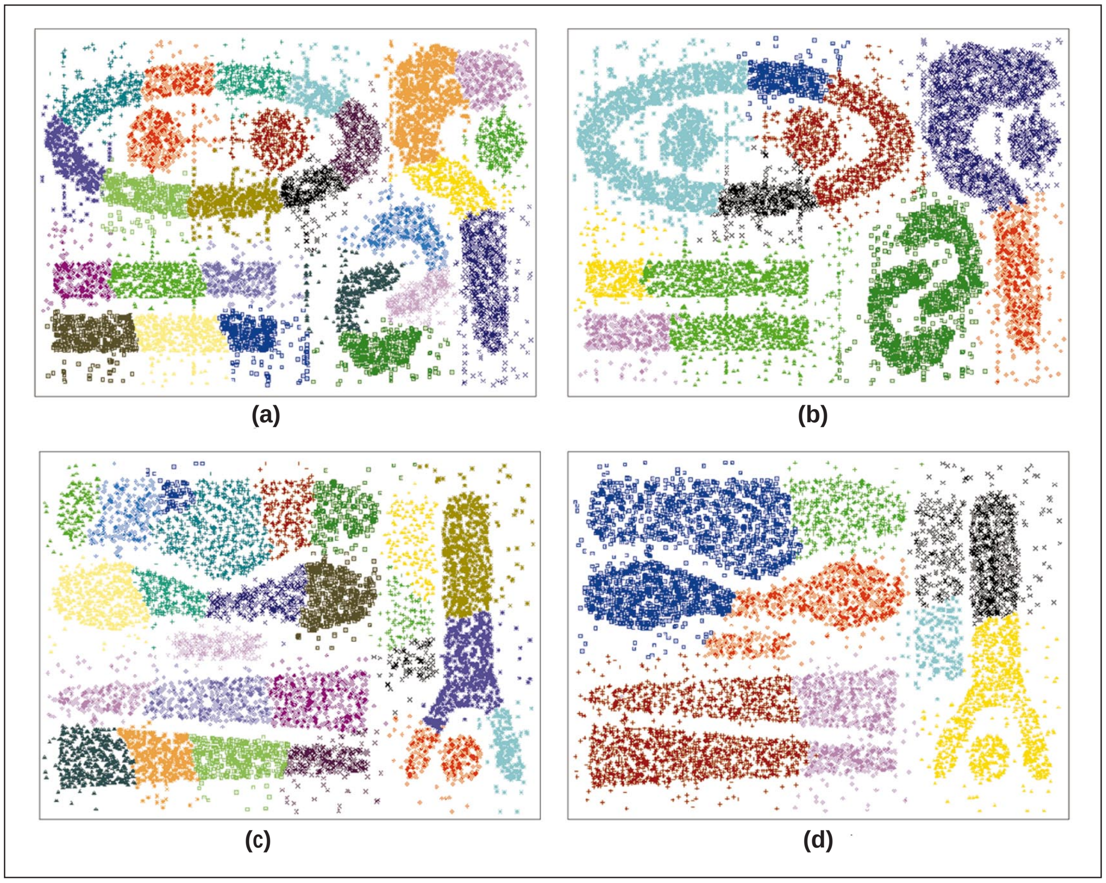

Figure 7. Clusters identified by CURE with shrinking factor 0.3 and number of representative points equal to 10. For DS3, CURE first merges subclusters that belong to two different subclusters at (a) 25 clusters. With (b) 11 clusters specified—the same number that Chameleon found—CURE obtains the result shown. CURE produced these results for DS4 with (c) 25 and (d) 8 clusters specified.

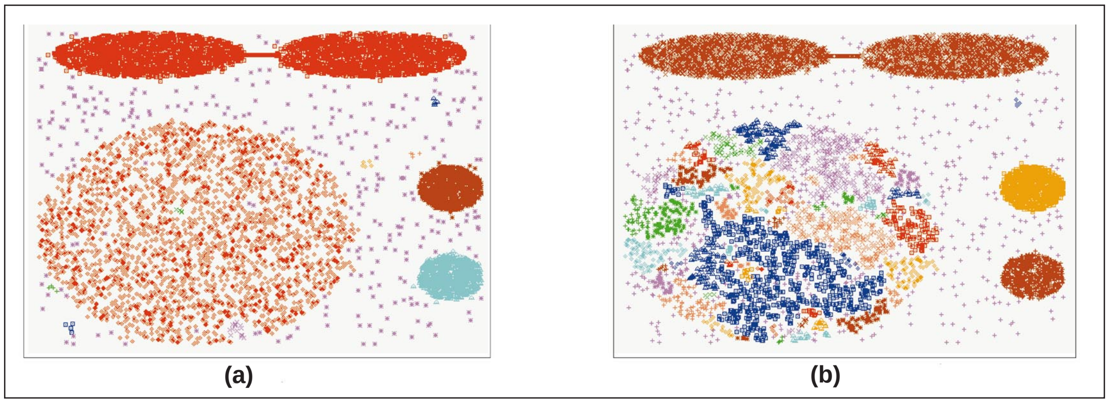

Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.

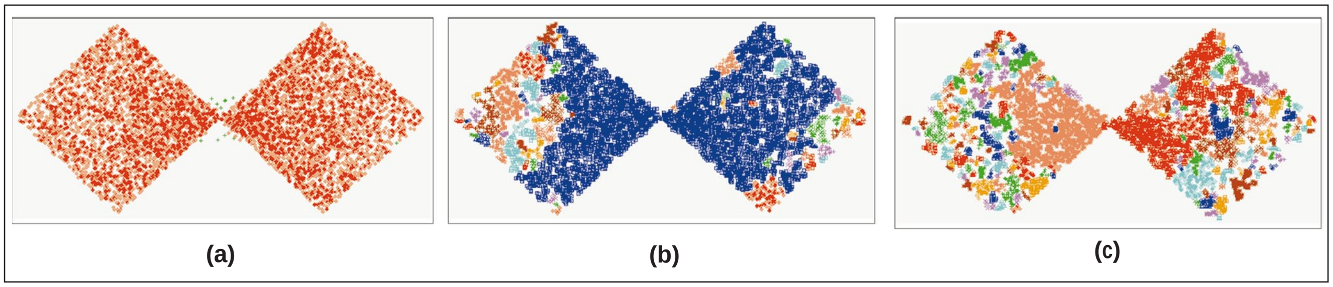

Figure 9. DBScan results for DS2 with MinPts at 4 and Eps at (a) 5.0, (b) 3.5, and (c) 3.0.

developers,[7] we fixed the MinPts at 4 and varied Eps in these experiments.

The clusters produced for DS1 illustrate that DBScan cannot effectively find clusters of different density. In Figure 8a, DBScan puts the two red ellipses (at the top of the image) into the same cluster because the outlier points satisfy the density requirements as dictated by the Eps and MinPts parameters. Decreasing the value of Eps can separate these clusters, as shown in Figure 8b for Eps at 0.4. Although DBScan separates the ellipses without fragmenting them, it now fragments the largest—but lower density—cluster into several small subclusters. Our experiments have shown that DBScan exhibits similar characteristics on DS4.

The clusters produced for DS2 illustrate that DBScan cannot effectively find clusters with variable internal densities. Figure 9a through Figure 9c show a sequence of three clustering solutions for decreasing values of Eps. As we decrease Eps in the hope of separating the two clusters, the natural clusters in the data set are fragmented into several smaller clusters.

DBScan finds the correct clusters in DS3 given the right combination of Eps and MinPts. If we fix MinPts at 4, then the algorithm works fine on this data set as long as Eps is from 5.7 to 6.1.

Even though we chose to model the data using a *k*-nearest-neighbor graph, it is entirely possible to use other graph representations—such as those based on mutual shared neighbors,[2,8,11]—which are suitable for particular application domains. Furthermore, different domains may require different models for capturing relative closeness and interconnectivity. 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. We also want to study the effectiveness of different techniques for modeling data as well as cluster similarity. ❖

Acknowledgments

This work was partially supported by Army Research Office contract DA/DAAG55-98-1-0441 and the Army High-Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the government's position or policy, and should infer no official endorsement. This work was also supported by an IBM Partnership Award. Access to computing facilities was provided by the Army High-Performance Computing Research Center and the Minnesota Supercomputer Institute.

We also thank Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu for providing the DBScan program.

References

1. M.S. Chen, J. Han, and P.S. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Trans. Knowledge and Data Eng.*, Dec. 1996, pp. 866-883.
2. A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Upper Saddle River, N.J., 1988.
3. D. Boley et al., "Document Categorization and Query Generation on the World Wide Web Using WebACE," to be published in *AI Review*, 1999.
4. E.H. Han et al., "Hypergraph-Based Clustering in High-Dimensional Data Sets: A Summary of Results," *Bull. Tech.*

Committee on Data Eng., Vol. 21, No. 1, 1998, pp. 15-22. Also a longer version available as Tech. Report TR-97-019, Dept. of Computer Science, Univ. of Minnesota, 1997.

5. V. Ganti et al., "Clustering Large Datasets in Arbitrary Metric Spaces," *Proc. 15th Int'l Conf. Data Eng.*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 502-511.
6. R. Ng and J. Han, "Efficient and Effective Clustering Method for Spatial Data Mining," *Proc. 20th Int'l Conf. Very Large Data Bases*, Morgan Kaufmann, San Francisco, 1994, pp. 144-155.
7. M. Ester et al., "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proc. 2nd Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1996, pp. 226-231.
8. S. Guha, R. Rastogi, and K. Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes," *Proc. 15th Int'l Conf. Data Eng.*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 512-521.
9. S. Guha, R. Rastogi, and K. Shim, "CURE: An Efficient Clustering Algorithm for Large Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, New York, 1998, pp. 73-84.
10. G. Karypis and V. Kumar, "hMETIS 1.5: A Hypergraph Partitioning Package," Tech. Report, Dept. of Computer Science, Univ. of Minnesota, 1998; http://winter.cs.umn.edu/~karypis/metis.
11. K.C. Gowda and G. Krishna, "Agglomerative Clustering Using the Concept of Mutual Nearest Neighborhood," *Pattern Recognition*, Feb. 1978, pp. 105-112.

George Karypis is an assistant professor of computer science at the University of Minnesota. His research interests include data mining, bio-informatics, parallel computing, graph partitioning, and scientific computing. Karypis has a PhD in computer science from the University of Minnesota. He is a member of the IEEE Computer Society and the ACM.

Eui-Hong (Sam) Han is a PhD candidate at the University of Minnesota. His research interests include data mining and information retrieval. Han has an MS in computer science from the University of Texas, Austin. He is a member of the ACM.

Vipin Kumar is a professor of computer science at the University of Minnesota. His research interests include data mining, parallel computing, and scientific computing. Kumar has a PhD in computer science from the University of Maryland, College Park. He is a member of the IEEE, ACM, and SIAM.

Contact the authors at the Univ. of Minnesota, Dept. of Computer Science and Eng., 4-192 EECS Bldg., 200 Union St. SE, Minneapolis, MN 55455; {karypis, han,kumar}@cs.umn.edu.