SPARCL: Efficient and Effective Shape-based Clustering

Vineet Chaoji*, Mohammad Al Hasan, Saeed Salem, and Mohammed J. Zaki
Dept. of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 12180
{chaojv, alhasan, salems, zaki}@cs.rpi.edu

Abstract

CLUSTERING is one of the fundamental data mining tasks. Many different clustering paradigms have been developed over the years, which include partitional, hierarchical, mixture model based, density-based, spectral, subspace, and so on. The focus of this paper is on full-dimensional, arbitrary shaped clusters. Existing methods for this problem suffer either in terms of the memory or time complexity (quadratic or even cubic). This shortcoming has restricted these algorithms to datasets of moderate sizes. In this paper we propose SPARCL, a simple and scalable algorithm for finding clusters with arbitrary shapes and sizes, and it has linear space and time complexity. SPARCL consists of two stages – the first stage runs a carefully initialized version of the Kmeans algorithm to generate many small seed clusters. The second stage iteratively merges the generated clusters to obtain the final shape-based clusters. Experiments were conducted on a variety of datasets to highlight the effectiveness, efficiency, and scalability of our approach. On the large datasets SPARCL is an order of magnitude faster than the best existing approaches.

1. Introduction

Given a set of n objects in d-dimensional space, cluster analysis assigns the objects into k groups such that each object in a group is more similar to other objects in its group as compared to objects in other groups. This notion of capturing similarity between objects lends itself to a variety of applications. As a result, cluster analysis plays an important role in almost every area of science and engineering, including bioinformatics, market research, privacy and security, image analysis, web search and so on.

Due to the large number of potential application domains, many flavors of clustering algorithms have been proposed [7]. Based on the mode of operation, they can be broadly categorized as variance-based, hierarchical, partitional, spectral, probabilistic/fuzzy and density-based. However, the common task among all algorithms is that they compute the similarities (distances) among the data points to solve the clustering problem. The definition of similarity or distance varies based on the application domain. If the clusters are of globular (spherical) shape and are of comparable size, typical distortion minimization criteria of Kmeans works well. However, in applications like image segmentation or spatial data mining, the above does not hold in general, since clusters in these applications generally are a dense set of points that can represent (physical) objects of arbitrary shapes. The Kmeans distortion minimization criteria fails to isolate those objects since it favors compact and spherical shaped clusters. In this paper, our focus is on this arbitrary-shape clustering task.

Shape-based clustering remains of active interest, and several previous approaches have been proposed; spectral [13], density-based (DBSCAN [3]), and nearest-neighbor graph based (Chameleon [8]) approaches are the most successful among the many shape-based clustering methods. However, they either suffer from poor scalability or are very sensitive to the choice of the parameter values. On the one hand, simple and efficient algorithms like Kmeans are unable to mine arbitrary-shaped clusters, and on the other hand, clustering methods that can cluster such datasets are not very efficient. Considering the data generated by current sources (e.g., geo-spatial satellites) there is a need for efficient algorithms in shape-based clustering domain that can scale to much larger datasets.

In this paper, we propose a simple, yet highly scalable algorithm for mining clusters of arbitrary shapes, sizes and densities. We call our new algorithm SPARCL (which is an anagram of the bold letters in **S**h**AP**e-based **CL**uste**R**ing). In order to achieve this we exploit the linear (in the number of objects) runtime of Kmeans based algorithms while avoiding its drawbacks. Recall that Kmeans based algorithms assign all points to the nearest cluster center; thus the center represents a set of objects that collectively approximates the shape of a d dimensional polyhedron [1]. When the number of centers are few, each such polyhedron covers a larger region, thus leading to incorrect partitioning of a dataset with arbitrary shapes. Increasing the number of

*Part of the work was done when the author was visiting Nokia Research Center, Palo Alto

[1] Each cluster boundary is a linear hyperplane, resulting in a polyhedron. As the number of clusters increase, the cluster boundary tends to resemble a hypersphere

centers reduces the region covered by each center. SPARCL exploits this observation by first using a smart strategy for sampling objects from the entire dataset. These objects are used as initial seeds of the Kmeans algorithm. On termination, Kmeans algorithm yields a set of centers. In the second step, a similarity metric for each pair of centers is computed. The similarity graph representing pairwise similarity between the centers is partitioned to generate the desired final number of clusters. To summarize we made the following key contributions in this work:

1. We define a new function that captures similarity between a pair of cluster centers, which is suitable for arbitrary shaped clusters.
2. We propose a new, highly scalable algorithm, SPARCL, for arbitrary shaped clusters, that combines partitional and hierarchical clustering in the two phases of its operation. The overall complexity of the algorithm is linear in the number of objects in the dataset.
3. SPARCL takes only two parameters – number of initial centers and the number of final clusters expected from the dataset. Note that the number of final clusters to find is typically a hyper-parameter of most clustering algorithms.
4. We perform a variety of experiments on both real and synthetic shape clustering datasets to show the strengths and weaknesses of our approach. We show that our method is an order of magnitude faster than the best current approaches.

2. Related Work

Here we review some of the pioneering methods in arbitrary shape clustering. DBSCAN [3] was one of the earliest algorithms that addressed arbitrary shape clustering. It defines two parameters – *eps* and *MinPts*. A point is labeled as a *core point* if the number of points within its *eps* neighborhood is at least *MinPts*. A cluster is defined as the maximal set of "*reachable*" core points. Points that are not core and not reachable from a core are labeled as noise. The main advantages of DBSCAN are that it does not require the number of desired clusters as an input, and it explicitly identifies outliers. On the flip side, DBSCAN can be quite sensitive to the values of *eps* and *MinPts*, and choosing correct values for these parameters is not that easy. DBSCAN is an expensive method, since in general it needs to compute the *eps* neighborhood for each point, which takes $O(n^2)$ time, especially with increasing dimensions; this time can be brought down to $O(n \log n)$ in lower dimensional spaces, via the use of spatial index structures like R^*-trees. DENCLUE [6] is another density based clustering algorithm based on kernel density estimation. Clusters are determined by identifying *density attractors* which are local maximas of the density function. DENCLUE shares some of the same limitations of DBSCAN, namely, sensitivity to parameter values, and its complexity is $O(n \log m + m^2)$, where n is the number of points, and m is the number of populated cells. In the worst case $m = O(n)$, and thus its complexity is also $O(n^2)$.

CURE [5] is a hierarchical agglomerative clustering algorithm that handles shape-based clusters. It follows the nearest neighbor distance to measure the similarity between two clusters, but reduces the computational cost significantly. The reduction is achieved by taking a set of representative points from each cluster and engaging only these points in similarity computations. CURE is still expensive with its quadratic complexity, and more importantly, the quality of clustering depends enormously on the sampling quality. In [8], the authors show several examples where CURE failed to obtain the desired shape-based clusters.

CHAMELEON [8] also formulates the shape-based clustering as a graph partitioning algorithm. A m nearest neighbor graph is generated for the input dataset, for a given number of neighbors m. This graph is partitioned into a predefined number of sub-graphs (also referred as subclusters). The partitioned sub-graphs are then merged to obtain the desired number of final k clusters. CHAMELEON introduces two measures – *relative interconnectivity* and *relative closeness* – that determine if a pair of clusters can be merged. Sub-clusters having high relative closeness and relative interconnectivity are merged. CHAMELEON is robust to the presence of outliers, partly due to the m-nearest neighbor graph which eliminates these noise points. This very advantage, turns into an overhead when the dataset size becomes considerably large, since computing the nearest neighbor graph can take $O(n^2)$ time as the dimensions increase.

The spectral clustering approach of Shi and Malik [13] is also capable of handling arbitrary shaped clusters. They represent the data points as a weighted undirected graph. They formulate the arbitrary shape clustering problem as a *normalized min-cut* problem, and approximate it by computing the eigen-vectors of the graph *Laplacian matrix*. Although, based on strong theoretical foundation, this method, unfortunately, is not scalable, due to its high computational time and space complexity. It requires $O(n^3)$ time to solve the Eigensystem of the symmetric Laplacian matrix, and storing the matrix also requires at least $\Omega(n^2)$ memory. There are some variations of this general approach [14], but all suffer from the poor scalabity problem.

Our proposed method SPARCL is based on the well known family of Kmeans based algorithms, which are widely popular for their simplicity and efficiency. Kmeans based algorithms operate in an iterative fashion. From an initial set of k selected objects, the algorithm iteratively refines the set of representatives with the objective of minimizing the mean squared distance (also known as *distortion*) from each object to its nearest representative. They are related to Voronoi tessellation, which leads to convex polyhedra in metric spaces [11]. As a consequence, Kmeans

based algorithms are unable to partition spaces with non-spherical clusters or in general arbitrary shapes. However, in this paper we show that one can use Kmeans type algorithms to obtain a set of seed representatives, which in turn can be used to obtain the final arbitrary shaped clusters. In this way, SPARCL retains the linear time complexity in terms of the data points, and is surprisingly effective as well, as we discuss next.

3. The SPARCL Approach

In this work we focus on a scalable algorithm for obtaining clusters with arbitrary shapes. In order to capture arbitrary shapes, we want to divide such shapes into convex pieces. This approach is motivated by the concept of *convex decomposition* [11] from computational geometry.

Convex Decomposition: Due to the simplicity of dealing with convex shapes, the problem of decomposing non-convex shapes into a set of convex shapes has been of great interest in the area of computational geometry. A *convex decomposition* is a partition, if the polyhedron is decomposed into disjoint pieces; and it is a cover, if the pieces are overlapping. While algorithms for convex decomposition are well understood in 2-dimensional space, the same cannot be said about higher dimensions. In this work, we approximate the convex decomposition of an arbitrary shape cluster by the convex polyhedra generated by the Kmeans centers that are within that cluster. Depending on the complexity of the shape, higher number of centers may be required to obtain a good approximation of that shape. Essentially, we can reformulate the original problem of identifying arbitrary shaped clusters in terms of a sampling problem. Ideally, we want to minimize the number of centers, with the constraint that the space covered by each center is a convex polyhedron. One can identify this optimization problem as a modified version of the facility location problem. In fact, this optimization problem is exactly the *Minimum Consistent Subset Cover Problem (MCSC)* [4]. Given a finite set S and a constraint, the MCSC problem considers finding a minimal collection T of subsets such that $\bigcup_{C \in T} C = S$, where $C \subset S$, and each C satisfies the given constraint c. In our case, S is the set of points and c the convex polyhedron constraint. The MCSC problem is NP-hard, and thus finding the optimal centers is hard. We thus rely on the iterative Kmeans type method to approximate the centers.

3.1. The SPARCL Algorithm

The pseudo-code for the SPARCL algorithm is given in Fig. 1. The algorithm takes two input parameters. The first one k is the final number of clusters desired. We refer to these as the *natural* clusters in the dataset, and like most other methods, we assume that the user has a good guess for k. In addition SPARCL requires another parameter K, which gives the number of seed centers to consider to approximate a good convex decomposition; we also refer to these seed centers as *pseudo-centers*. Note that $k < K \ll n = |D|$. Depending on the variant of Kmeans used to obtain the seeds centers, SPARCL uses a third parameter mp, denoting the number of nearest neighbors to consider during a smart initialization of Kmeans that avoids outliers as centers. The random initialization based Kmeans does not require the mp parameter.

SPARCL operates in two stages. In the first stage we run the Kmeans algorithm on the entire dataset to obtain K convex clusters. The initial set of centers for the Kmeans algorithm may be chosen randomly, or in such a manner that they are not outlier points. Following the Kmeans run, the second stage of the algorithm computes a similarity metric between every seed cluster pair. The resulting similarity matrix can act as input either for a hierarchical or a spectral clustering algorithm. It is easy to observe that this two-stage refinement employs a cheaper (first stage) algorithm to obtain a course grained clustering. The first phase has complexity $O(ndKe)$, where d is the data dimensionality and e is the number of iterations Kmeans takes to converge, which is linear in n. This approach considerably reduces the problem space as we only have to compute $O(K^2)$ similarity values in the second phase. For the second phase we can use a more expensive algorithm to obtain the final set of k natural clusters.

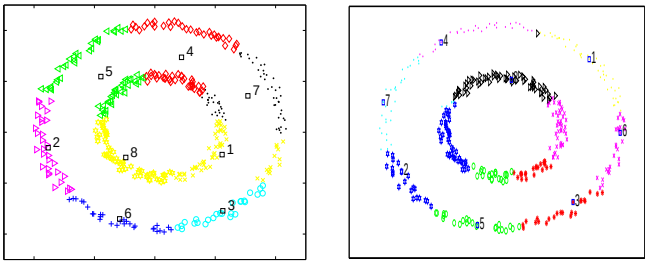
SPARCL($\mathcal{D}, K, k, mp$):
1. C_{init} = **seed_center_initialization**($\mathcal{D}, K, mp$)
2. C_{seed} = **Kmeans**(C_{init}, K)
3. **forall** distinct pairs $(C_i, C_j) \in C_{seed} \times C_{seed}$
4. $\quad S(i,j)$ = **compute_similarity**(C_i, C_j)
5. **cluster_centers**(C_{seed}, S, k)

Figure 1: The SPARCL Algorithm

3.1.1 Phase 1 – Kmeans Algorithm

The first stage SPARCL is shown in steps 1 – 2 of Fig. 1. This stage involves running the Kmeans algorithm with a set of initial centers C_{init} (line 1), until convergence, at which point we obtain the final seed clusters C_{seed}. There is one subtlety in this step; instead of using the mean point in each iteration of Kmeans, we actually use an actual data point in the cluster that is closest to the center mean. We do this for two reasons. First, if the cluster centers are not actual points in the dataset, chances are higher that points from two different natural clusters would belong to a seed cluster, considering that the clusters are arbitrarily shaped. When this happens, the hierarchical clustering in the second phase would merge parts of two different natural clusters. Second, our approach is more robust to outliers, since the mean point can get easily influenced by outliers. Fig. 2(a) outlines an example. There are two natural clusters in the form of the two rings. When we run a regular Kmeans, using the mean point as the center representative, we obtain some seed centers that lie in empty space, between the two ring-shaped clusters (e.g., 4, 5, and 7). By choosing an actual data point,

we avoid the "dangling" means problem, and are more robust to outliers, as shown in Fig. 2(b). This phase starts by

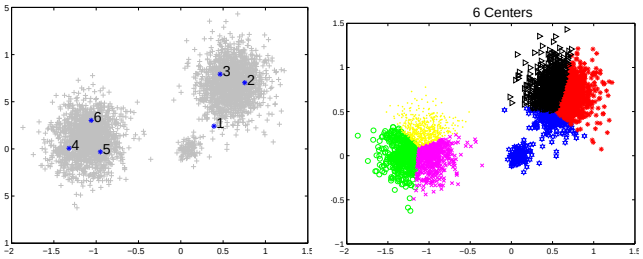


(a) Using mean point (b) Using actual data point

Figure 2: Effect of Choosing mean or actual data point

selecting the initial set of centers for the Kmeans algorithm. In order for the second stage to capture the natural clusters in the datasets, it is important that the final set of seed centers, C_{seed}, generated by the Kmeans algorithm satisfy the following properties: 1) Points in C_{seed} are not outlier points, 2) Representatives in C_{seed} are spread evenly over the natural clusters.

In general, random initialization is fast, and works well. However, selecting the centers randomly can violate either of the above properties, which can lead to ill-formed clusters for the second phase. Fig. 3 shows an example of such a case. In Fig. 3(a) seed center 1 is almost an outlier point. As a result the members belonging to seed center 1 come from two different natural clusters. This results in the small (middle) cluster merging with the larger cluster to its right. In order to avoid such cases and to achieve both the properties mentioned above we utilize the outlier and density insensitive selection of initial centers as described next.



(a) Randomly selected centers (b) Natural cluster split

Figure 3: Bad Choice of Cluster Centers

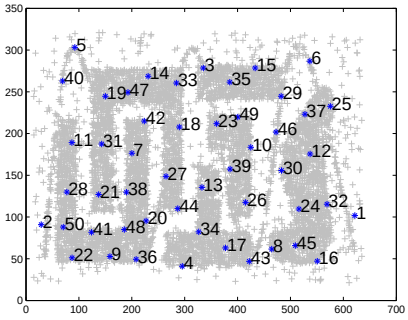
Initialization using Local Outlier Factor: We use the *local outlier factor (LOF)* criterion for selecting the initial set of cluster centers. LOF was proposed in [1] as a measure for determining the degree to which a point is an outlier. For a point $x \in D$, define the local neighborhood of x, given the minimum points threshold mp as follows:

$$N(x, mp) = \{y \in D \mid dist(x, y) \leq dist(x, x_{mp})\}$$

where x_{mp} is the mp-th nearest neighbor of x. The *density* of x is then computed as follows:

$$density(x, mp) = \left(\frac{\sum_{y \in N(x,mp)} distance(x, y)}{\mid N(x, mp) \mid} \right)^{-1}$$

In summary, the LOF score of x is the ratio of the average density of the points in $N(x, mp)$ to $density(x, mp)$. Thus LOF value represents the extent to which a point is an outlier. A point that belongs to a cluster has an LOF value approximately equal to 1, since its density and the density of its neighbors is approximately the same. LOF has three excellent properties: (1) It is very robust when the dataset has clusters with different sizes and densities. (2) Even though the LOF value may vary somewhat with mp, it is generally robust in making the decision whether a point is an outlier or not. (3) It leads to practically faster convergence of the Kmeans algorithm, i.e., fewer iterations. The following approach is used to select the initial seeds. Assume the i initial centers have been chosen. To choose the $i+1$-th center, first compute the distance of each point to each of the i chosen centers, and sort them in decreasing order of distance. Next, in that sorted order, pick the first point that is not an outlier as the next seed. This process is repeated until K initial seeds have been chosen. Finally, run the Kmeans algorithm to converge with those initial starting centers, to obtain the final set of seed centers C_{seed}. As an example of LOF-based seed selection, Fig. 4 shows the initial set of centers for one of the shape-based datasets. The overall complexity of this approach is $O(mp \times K^2 \times t \times n)$, where $t \ll n$ is the number of outliers in the data.



(a) Centers selected on DS1 dataset

Figure 4: Local Outlier Based Center Selection

3.1.2 Phase 2 – Merging Neighboring Clusters

As the output of the first phase of the algorithm, we have a relatively small number K of seed cluster centers (compared to the size of the dataset) along with the point assignments for each cluster. During the second phase of the algorithm, a similarity measure for each pair of seed clusters is computed (see lines 3-4 in Fig. 1). The similarity between clusters is then used to drive any clustering algorithm that can use the similarity function to merge the K seed clusters in the final set of k natural clusters. We applied both hierarchical as well as spectral methods on the

similarity matrix. Since the size of the similarity matrix is $O(K^2)$, as opposed to $O(n^2)$ even spectral methods can be conveniently applied.

Cluster Similarity: Let us consider that the d-dimensional points belonging to a cluster X are denoted by P_X and similarly points belonging to cluster Y are denoted by P_Y. The corresponding centers are denoted by c_X and c_Y, respectively. A similarity score is assigned to each cluster pair. Conceptually, each cluster can be considered to represent a Gaussian and the similarity captures the overlap between the Gaussians. Intuitively, two clusters should have a high similarity score if they satisfy the following conditions:

1. The clusters are close to each other in the Euclidean space.
2. The densities of the two clusters are comparable, which implies that one cluster is an extension of the other.
3. The face (hyperplane) at which the clusters meet is wide.

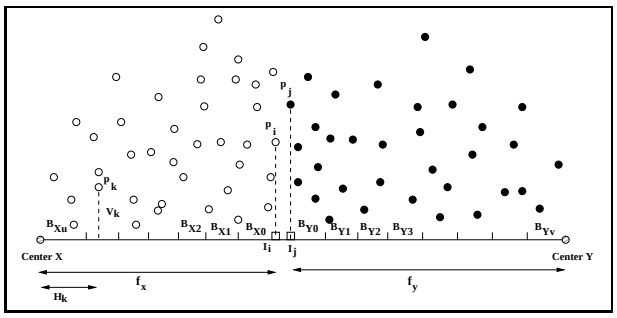


Figure 5: Projection of points onto line between centers

The **compute_similarity** function in Fig. 1 computes the similarity for a given pair of centers. For computing the similarity, points belonging to the two clusters are projected on the vector connecting the two centers as shown in Fig. 5. Even though the figure just shows points above the vector being projected, this is merely for the convenience of exposition and illustration. f_x represents the distance from the center X to the farthest projected image I_i of a point p_i belonging to X. H_i is the horizontal (along the vector joining the two centers) distance of the projection of point p_i from the center, and V_i is the perpendicular (vertical) distance of the point from its projection. The means (m_{H_X} and m_{H_Y}) and standard deviations (s_{H_X} and s_{H_Y}) of the horizontal distances for points belonging to the clusters are computed. Similarly, means and standard deviations for perpendicular distances are computed. A histogram with bin size of $\frac{s_i}{2}$ ($i \in \{H_X, H_Y\}$) is constructed for the projected points. The bins are numbered starting from the farthest projected point f_i ($i \in X, Y$), i.e., bin B_{X_0} is the first bin for the histogram constructed on points in cluster X. The number of bins for cluster X is given by $|B_X|$. Then, we compute the average of horizontal distances for points in each bin; d_{i_j} denotes the average distance for bin j in cluster i. $max\_bin_i = \arg\max_j d_{i_j}$ represents the bin with the largest number of projected points in cluster i. The number of points in bin X_i is given by $N[X_i]$. The ratio $\frac{N[X_i]}{N[X_{max\_bin_X}]}$ is denoted by $sz\_ratio_{X_i}$.

Now, the size based similarity between two bins in clusters X and Y is given by the equation:

$$size\_sim(B_{X_i}, B_{Y_j}) = sz\_ratio(B_{X_i}) * sz\_ratio(B_{Y_j}) \quad (1)$$

The distance-based similarity between two bins in clusters X and Y is given by the following equation, where $dist(B_{X_i}, B_{Y_j})$ is the horizontal distance between the bins X_i and Y_j:

$$dist\_sim(B_{X_i}, B_{Y_j}) = \frac{2 * dist(B_{X_i}, B_{Y_j})}{s_{H_X} + s_{H_Y}} \quad (2)$$

The overall similarity between the clusters X and Y is then given as

$$S(X,Y) = \sum_{i=0}^{t} size\_sim(B_{X_i}, B_{Y_i}) * \exp^{-dist\_sim(B_{X_i}, B_{Y_i})} \quad (3)$$

where $t = \min(|B_X|, |B_Y|)$. Also, while projecting the points onto the vector, we discarded points that had a vertical distance greater than twice the vertical standard deviation, considering them as noise points.

Let us look closely at the above similarity metric to understand how it satisfies the above mentioned three conditions for good cluster similarity. Since the bins start from the farthest projected points, for bordering clusters the distance between X_0 and Y_0 will be very less. This gives a small value to $dist\_sim(B_{X_0}, B_{Y_0})$. As a result, the exponential function gets a high value due to the exponent taking a low value. This causes the first term of the summation in Equation 3 to be high, especially if the $size\_sim$ score is also high. A high value for the first term indicates that the two clusters are close by and that there are a large number of points along the surface of intersection of the two clusters. If the $size\_sim(B_{X_0}, B_{Y_0})$ is small, which can happen when the two clusters meet at a tangent point, the first term in the summation will be small. This is exactly as expected intuitively and captures conditions 1 and 3 mentioned above. Both the $size\_sim$ and $dist\_sim$ measures are averse to outliers and would give a low score for bins containing outlier points. For outlier bins, the sz_ratio will have a low score, resulting in a lower score for $size\_sim$. Similarly, clusters having outlier points would tend to have a high standard deviation, which would result in a low score for $dist\_sim$.

We considered the possibility of extending the histogram to multiple dimensions, along the lines of grid-based algorithms, but the additional computational cost does not justify the improvement in the quality of the results. Finally, once the similarity between pairs of seeds has been com-

puted, we can use spectral or hierarchical agglomerative clustering to obtain the final set of k natural clusters. For our experiments, we used the agglomerative clustering algorithm provided with CLUTO.

This similarity metric can be shown to be a *kernel*. The following lemmas regarding kernels allow us to prove that the similarity function is a kernel.

Lemma 1 *[12] κ_1 and κ_2 be kernels over $\mathbf{X} \times \mathbf{X}$, $\mathbf{X} \subseteq \mathbb{R}^n, f(.)$ a real-valued function on $\mathbf{X}$. Then the following functions are kernels:*

i. $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
ii. $\kappa(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$,
iii. $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$,

Lemma 2 *[12] Let $\kappa_1(\mathbf{x}, \mathbf{z})$ be a kernel over $\mathbf{X} \times \mathbf{X}$, where $\mathbf{x}, \mathbf{z} \in \mathbf{X}$. Then the function $\kappa(\mathbf{x}, \mathbf{z}) = \exp(\kappa_1(\mathbf{x}, \mathbf{z}))$ is also a kernel.*

Theorem 3.1 *Function $S(X, Y)$ in Equation 3 is a kernel function.*

PROOF*:* Since *dist* and *sz_ratio* are real valued functions, $dist\_sim$ and $size\_sim$ are kernels by Lemma 1(ii). This makes $\exp(-dist\_sim(.,.))$ a kernel by Lemma 2. Product of $size\_sim$ and $\exp(-dist\_sim(.,.))$ is a kernel by Lemma 1(iii). And finally, $S(X, Y)$ is a kernel since the sum of kernels is also a kernel by Lemma 1(i). ■

The matrix obtained by computing $S(X, Y)$ for all pairs of clusters turns out to be a *kernel matrix*. This nice property provides the flexibility to utilize any kernel based methods, such as spectral clustering [10] or kernel k-means [2], for the second phase of SPARCL.

3.2. Complexity Analysis

The first stage of SPARCL starts with computing initial K centers randomly or based on the local outlier factor. If we use random initialization phase 1 takes $O(Knde)$ time, where e is the number of Kmeans iterations. The time for computing the LOF-based seeds is $O(mp\ K^2\ n\ t)$, where t is the number of outliers in the dataset, followed by the $O(Knde)$ time of Kmeans. The second phase of the algorithm projects points belonging to every cluster pair on the vector connecting the centers of the two clusters. The projected points are placed in appropriate bins of the histogram. The projection and the histogram creation requires time linear in the number of points in the seed cluster. For the sake of simplifying the analysis, let us assume that each seed cluster has the same number of points, $\frac{n}{K}$. Projection and histogram construction requires $O(\frac{n}{K})$ time. In practice only points from a cluster that lie between the two centers are processed, reducing the computation by half on an average. Since there are $O(K^2)$ pairs of centers, the total complexity for generating the similarity map is $K^2 \times O(\frac{n}{K}) = O(Kn)$. The final stage applies a hierarchical or spectral algorithm to find the final set of k clusters. Spectral approach will take $O(K^3)$ time in the worst case, whereas agglomerative clustering will take time $O(K^2 \log K)$. Overall, the time for SPARCL is $O(Knd)$ (ignoring the small number of iterations it takes Kmeans to converge) if using the random initialization, or $O(K^3mnd)$, assuming $mp = O(K)$, and using the LOF-based initialization. In our experiment evaluation, we obtained comparable results using random initialization for datasets with uniform density of clusters. With random initialization, the algorithm runs in time linear in the number of points as well as the dimensionality of the dataset.

4. Experiments and Results

Experiments were performed to compare the performance of our algorithm with Chameleon [8], DBSCAN [3] and spectral clustering [13]. The Chameleon code was obtained as a part of the CLUTO [15][2] package. The DBSCAN implementation in Weka was used for comparison. Similarly, for spectral clustering a Matlab implementation *SpectraLIB*, [3] based on the Shi-Malik algorithm [13] was used.

Even though the implementations are in different languages, some of which might be inherently slower than others, the speedup due to our algorithm far surpasses any implementation biases. All the experiments were performed on a Mac G5 machine with a 2.66 GHz processor, running the Mac 10.4 OS X. Our code is written in C++ using the Computational Geometry Algorithms Library (CGAL). We show results for both LOF based as well as random initialization of seed clusters.

4.1. Datasets

4.1.1 Synthetic Datasets

We used a variety of synthetic and real datasets to test the different methods. DS1, DS2, DS3, and DS4, shown in Figs 7(a), 7(b), 7(c), and 7(d), have been used in previous studies including Chameleon and CURE. These are all 2d datasets with points ranging from 8000 to 100000. The Swiss-roll dataset in Fig.7(e) is the classic non-linear manifold used in non-linear dimensionality reduction [9]. We split the manifold into four clusters to see how our methods handle this case.

For the scalability tests, and for generating 3d datasets, we wrote our own shape-based cluster generator. To generate a shape in 2d, we randomly choose points in the drawing canvas and accept points which lie within our desired shape. All the shapes are generated with point (0,0) as the origin. To get complex shapes, we combine rotated and translated forms of the basic shapes (circle, rectangle, ellipse, circular strip, etc.). Our 3d shape generation is built on the 2d

[2] http://glaros.dtc.umn.edu/gkhome/cluto/
[3] http://www.stat.washington.edu/spectral/

Name	$\|D\|(d)$	k	SPARCL (LOF/Random)	Chameleon	DBSCAN	Spectral
DS1	8000 (2)	6	5.74/1.277	4.02	14.16	-
DS2	10000 (2)	9	8.6/1.386	5.72	24.2	-
DS3	8000 (2)	8	6.88/1.388	4.24	14.52	-
DS4	100000 (2)	5	35.24/20.15	280.47		-
Swiss-roll	19200 (3)	4	23.92/17.89	19.38		-

Table 1: Runtime Performance on Synthetic Datasets. All times are reported in seconds. '-' for Spectral method denotes the fact that it ran out of memory for all these cases.

shapes. We randomly choose points in the 3 coordinates – if the x and y coordinates satisfy the shape, we randomly choose the z-axis from within a given range. This approach generates true 3d shapes, and not only layers of 2d shapes. Similar to the case for 2d, we combine rotated and translated basic 3d shapes to get more sophisticated shapes. Once we generate all the shapes, we randomly add noise (1% to 2%) to the drawing frame. An example of a synthetic 3d dataset is shown in Fig. 8(b). This 3d dataset has 100000 points, and 10 clusters.

4.1.2 Real Datasets

We used two real shape-based datasets. The first is a set of 2d images from benign breast cancer. The actual images are divided into 2d grid cells (80×80) and the intensity level is used to assign each grid to either a cell or background. The final dataset contains only the actual cells, along with their x and y co-ordinates.

The second dataset consists of protein structures. Proteins are 3d objects where the coordinates of the atoms represent points. Since proteins are deposited in the protein data bank (PDB) in different reference frames, the coordinates of the protein are centered, as a preprocessing step. We translate the proteins to get separated clusters. Once the translation is done, we add the noise points. Our protein dataset has 15000 3d points obtained from the following proteins: 1A1T (865 atoms), 1B24 (619 atoms), 1DWK (11843 atoms), 1B25 (1342 atoms), and 331 noise points.

4.2. Results on Synthetic Datasets

Results of SPARCL on the synthetic datasets are shown in Table 1, and in Figs 7(a), 7(b), 7(c), 7(d), and 7(e). We refer the reader to [8] for the clustering results of Chameleon and DBSCAN on datasets DS1-4. To summarize, Chameleon is able to perfectly cluster these datasets, whereas both DBSCAN and CURE make mistakes, or are very dependent on the right parameter values to find the clusters. As we can see SPARCL also perfectly identifies the shape-based clusters in these datasets. On the nonlinear Swiss-roll dataset, SPARCL does make minor mistakes at the boundaries (Fig. 7(e)). The reason for this is that SPARCL is designed mainly for full-space clusters, whereas this is a 2d manifold embedded in a 3d space. Further, it is a nonlinear subspace cluster. What is remarkable is that SPARCL can actually find a fairly good clustering even in this case.

Table 1 shows the characteristics of the synthetic datasets along with their running times. The default parameters for running Chameleon in CLUTO were retained (number of neighbors was set at 40). Parameters that were set for Chameleon include the use of graph clustering method (*clmethod=graph*) with similarity set to inverse of Euclidean distance (*sim=dist*) and the use of agglomeration (*agglofrom=30*), as suggested by the authors. Results for both the LOF and random initialization are presented for SPARCL. Also, we used $K = 50, 60, 70, 50$ for each of the datasets DS1-4, respectively. For swiss-roll we use $K = 530$.

We can see that DBSCAN is 2-3 times slower than both SPARCL and Chameleon on smaller datasets. However, even for these small datasets, the spectral approach ran out of memory. The times for SPARCL (with LOF) and Chameleon are comparable for the smaller datasets, though the random initialization gives the same results and can be 3-4 times faster. For the larger DS4 dataset SPARCL has an order of magnitude faster performance, showing the real strength of our approach. For DBSCAN we do not show the results for DS4 and Swiss-roll since it returned only one cluster, even when we played with different parameter settings.

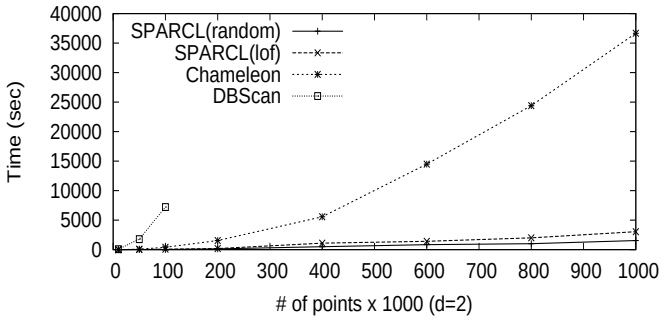
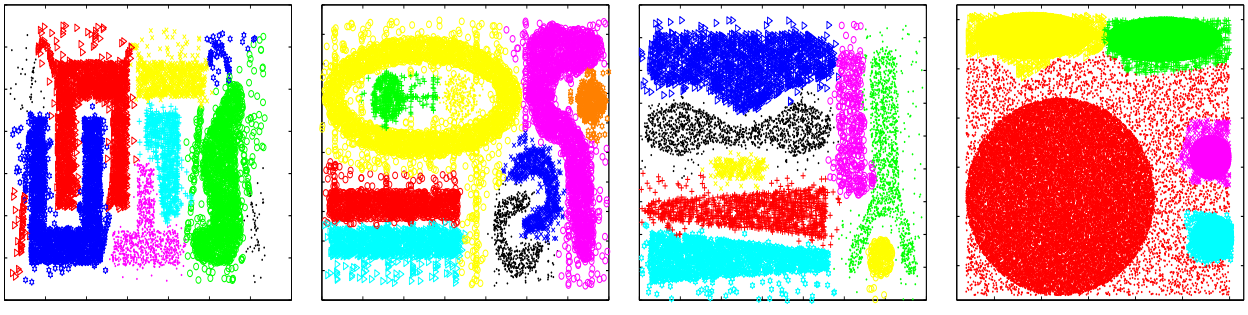


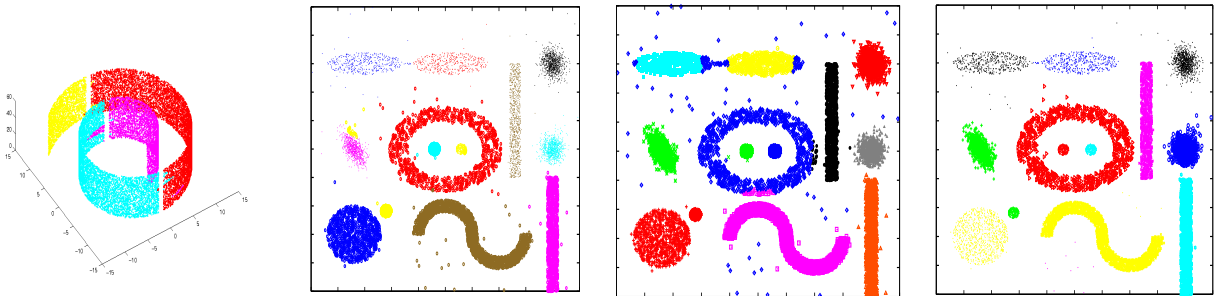
Figure 6: Scalability Results on Dataset DS5

4.2.1 Scalability Experiments

Using our synthetic dataset generator, we generated DS5, in order to perform experiments on varying number of points, varying densities and varying noise levels. For studying the scalability of our approach, different versions of DS5 were generated with different number of points, but keeping the number of clusters constant at 13. The noise level was kept at 1% of the dataset size. Fig. 6 compares the runtime performance of Chameleon, DBScan and our approach for dataset sizes ranging from 100000 points to 1



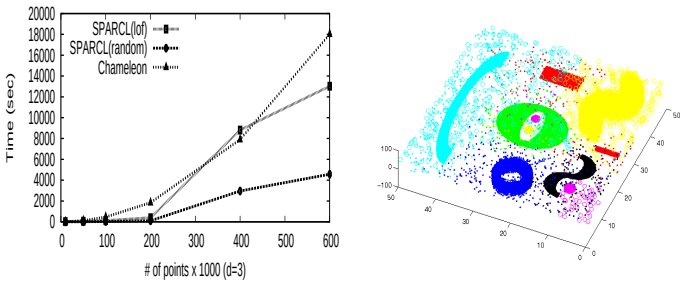
(a) Results on DS1 (b) Results on D2 (c) Results on D3 (d) Results on D4



(e) Results on Swiss-roll (f) SPARCL ($K = 100$) (g) DBSCAN (*minPts=150, eps=0.05*) (h) Chameleon (*agglofrom=30, sim=dist, clmethods=graph*)

Figure 7: Clustering Quality on Synthetic Datasets

million points. We chose not to go beyond 1 million as the time taken by Chameleon and DBSCAN was quite large. In fact, we had to terminate DBSCAN beyond 100K points. Fig. 6 shows that our approach, with random initialization, is around 22 times faster than Chameleon while it is around 12 time faster when LOF based initialization is considered. Note that the time for LOF also increases as the size of the dataset increases. For Chameleon, the parameters *aglofrom*, *sim*, *clmethod* were set to 30, *dist* and *graph*, respectively. For DBSCAN the *eps* was set at 0.05 and *MinPts* was set at 150 for the smallest dataset. *MinPts* was increased linearly with the size of the dataset. In our case, for all datasets, $K = 100$ seed centers were selected for the first phase and mp was set to 15. Figs 7(f), 7(g) and 7(h) shows the clusters obtained as a result of executing our algorithm, DBSCAN and Chameleon on the dataset DS5 of size 50K points. We can see that DBSCAN makes the most mistakes, whereas both SPARCL and Chameleon do well. Scalability experiments were performed on 3d datasets as well. Result for one of those datasets is shown in Fig. 8(b). The 3d dataset consists of shapes in full 3d space (and not 2d shapes embedded in 3d space). The dataset contained random noise too (2% of the dataset size). As seen in Fig. 8(a), SPARCL (with random initialization) can be more than four times as fast as Chameleon.



(a) (b)

Figure 8: Clustering Results on 3D Dataset

4.2.2 Clustering Quality

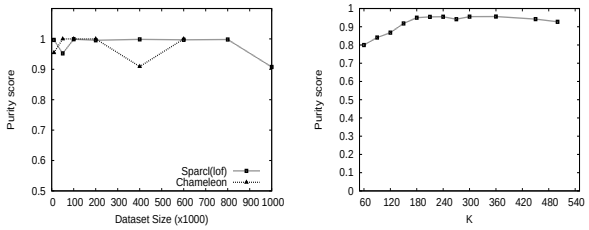
Since two points in the same cluster can be very far apart, traditional metrics such as cluster diameter, k-Means/k-Medoid objective function (sum of squared errors), compactness (avg. intra-cluster distance over the avg. inter-cluster distance), etc. are generally not appropriate for shape-based clustering. We apply supervised metrics, wherein the true clustering is known apriori, to evaluate clustering quality. Popular supervised metrics include *purity*, *Normalized Mutual Information*, *rank index*, etc. In this work, we use purity as the metric of choice due to its intuitive interpretation. Given the true set of clusters (referred to as *classes* henceforth to avoid confusion), $\mathcal{C}_T = \{c_1, c_2, \ldots, c_L\}$ and the clusters obtained from SPARCL $\mathcal{C}_S = \{s_1, s_2, \ldots, s_M\}$, *purity* is given by the expression:

$$purity(\mathcal{C}_S, \mathcal{C}_T) = \frac{1}{N} \sum_k \max_j \|s_k \cap c_j\| \qquad (4)$$

where N is the number of points in the dataset. Purity lies in the range [0,1], with a perfect clustering corresponding to purity value of 1.

Since DS1-DS4 and the real datasets do not provide the

class information, experiments were conducted on varying sizes of the DS5 dataset. The class information was recorded during the dataset generation. Fig. 9(a) shows the *purity* score for clusters generated by SPARCL and CHAMELEON (parameters *agglofrom=100, sim=dist, clmethod=graph*). Since these algorithms cluster noise points differently, for fair comparison they are ignored while computing the purity, although the noise points are retained during the algorithm execution. Note that for datasets larger than 600K, CHAMELEON did not finish in reasonable time. When CHAMELEON was run with the default parameters, which runs much faster, the purity score lowered to 0.6.



(a) Varying Dataset Size (b) Varying # of seed-clusters

Figure 9: Cluster Quality

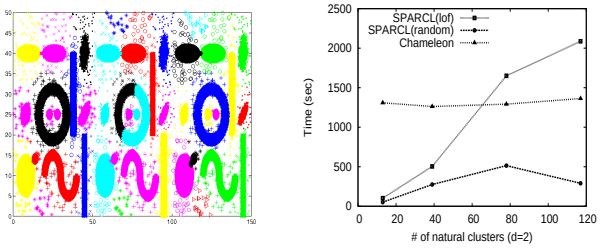
4.2.3 Varying Number of Clusters

Experiments were conducted to see the impact of varying the number of natural clusters k. To achieve this, the DS5 dataset was replicated by tiling the dataset in a grid form. Since the DS5 dataset contains 13 natural clusters, a 1×3 tiling contains 39 natural clusters (see Fig. 10(a)). The number of points are held constant at 180K. The number of natural clusters are varied from 13 to 117. The number of seed-clusters are set at 5 times number of natural clusters, i.e, $K = 5k$. We see that SPARCL finds most of the clusters correctly, but it does make one mistake, i.e., the center ring has been split into two. Here we find that since there are many more clusters, the time to compute the LOF goes up. In order to obtain each additional center the LOF method examines a constant number of points, resulting in a linear relation between the number of clusters and the runtime. Thus we prefer to use the random initialization approach when the number of clusters are large. With that SPARCL is still 4 times faster than Chameleon (see Fig. 10(b)).

Even though Chameleon produces results competent with that of SPARCL, it requires tuning the parameters to obtain these results. Especially when the nearest neighbor graph contains disconnected components CHAMELEON tends to break natural clusters in an effort to return the desired number of clusters. Hence CHAMELEON expects the user to have a certain degree of intuition regarding the dataset in order to set parameters that would yield the expected results.

4.2.4 Varying Number of Dimensions

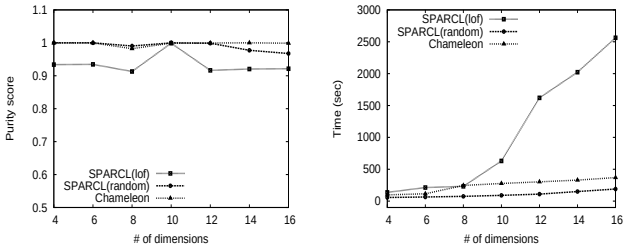
Synthetic data generator SynDECA (`http://cde.iiit.ac.in/~soujanya/syndeca/`) was used to generate higher dimensional datasets. The number of points and clusters were set to 500K and 10, respectively. 5% of



(a) 1x3 grid tiling. Results of SPARCL (b) Varying number of natural clusters.

Figure 10: Varying Number of Natural Clusters

the points were uniformly distributed as noise points. SynDECA can generate regular (circle, ellipse, square and rectangle) as well as random/irregular shapes. Although SynDECA can generate subspace clusters, for our experiments full dimensional clusters were generated. Fig. 11(b) shows the runtime for both LOF based and random initialization of seed clusters. With increasing number of dimensions, LOF computation takes substantial time. This effect can be attributed to a combination of two effects. First, since a kd-tree is used for nearest-neighbor queries the performance degrades with increasing dimensionality. Second, since we keep the number of points constant in this experiment, the sparsity of the input space increases for higher dimensions. On the other hand, random initialization is computationally inexpensive. Fig. 11(a) shows the purity for higher dimensions. Both SPARCL and CHAMELEON perform well on this measure.



(a) Purity (b) Runtime

Figure 11: Varying Number of Dimensions

4.2.5 Varying Number of Seed-Clusters (K)

SPARCL has a single parameter, K. Fig. 9(b) shows the effect of changing K on the quality of the clustering. The dataset used is same as in Fig. 10(a), with 300K points. As seen in the figure, the purity stabilizes around K=180 and remains almost constant till K=450. As K is increased further, a significant number of seed-centers lie between two clusters. As a result SPARCL tends to merge parts of one cluster with the other, leading to a gradual decline in the purity. Overall, the figure shows that SPARCL is fairly insensitive to the K value.

4.3. Results on Real Datasets

We applied SPARCL on the protein dataset. As shown in Fig. 12 SPARCL is able to perfectly identify the four proteins. On this dataset, Chameleon returns the same result. The results on the benign cancer datasets are shown in Fig. 13. Here too SPARCL successfully identifies the

regions of interest. We do not show the time for the real datasets since the datasets are fairly small and both Chameleon and SPARCL perform similarly.

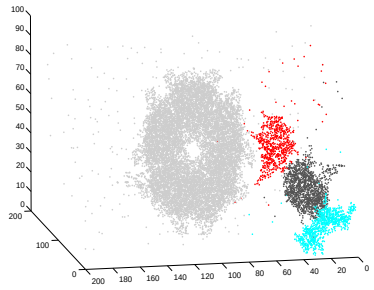
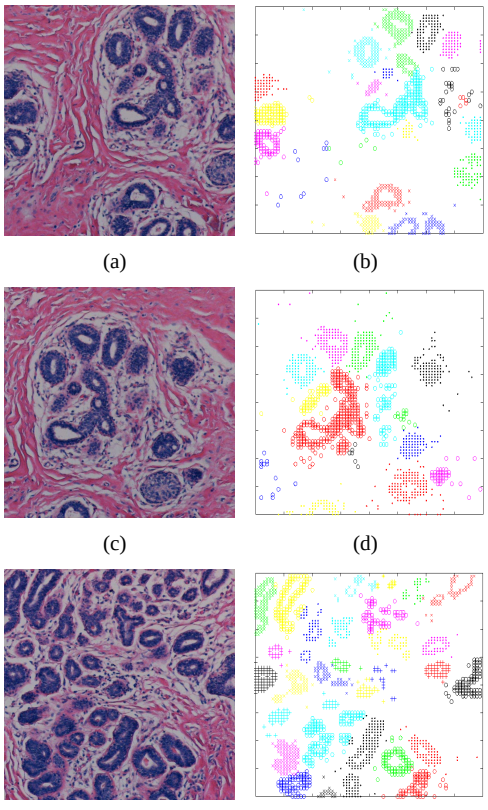


Figure 12: Protein Dataset



(a) (b) (c) (d) (e) (f)

Figure 13: Cancer Dataset: (a),(c),(e) are the actual benign tissue images. (b),(d),(f) gives the clustering of the corresponding tissues by SPARCL.

5. Conclusions

In this paper, we made use of a very simple idea, namely, to capture arbitrary shapes by means of convex decomposition, via the use of the highly efficient Kmeans approach. By selecting a large enough number of seed clusters K, we are able to capture most of the dense areas in the dataset. A similarity metric is defined that captures the extent to which points from the two clusters come close to the d-dimensional hyperplane separating them. The similarity is computed rapidly by projecting all the points in the two clusters on the line joining the two centers (which is reminiscent of linear discriminant analysis). We then apply a merging based approach to obtain the final set of user-specified (natural) clusters.

Our experimental evaluation shows that this simple approach, SPARCL, is quite effective in finding arbitrary shaped-based clusters in a variety of 2d and higher dimensional datasets. It has similar accuracy as Chameleon, a state of the art shape-based method, and at the same time it is over an order of magnitude faster, since its running time is essentially linear in the number of points as well as dimensions. In general SPARCL works well for full-space clusters, and is not yet tuned for subspace shape-based clusters, which we plan to tackle in future work.

References

[1] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *ACM SIGMOD Int. Conf. On Management of Data*, 2000.

[2] I. S. Dhillon, Y. Guan, and B. Julis. Kernel k-means, spectral clustering and normalized cuts. In *Proc. of KDD*, 2004.

[3] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *ACM SIGKDD*, pages 226–231, 1996.

[4] B. J. Gao, M. Ester, J.-Y. Cai, O. Schulte, and H. Xiong. The minimum consistent subset cover problem and its applications in data mining. In *ACM SIGKDD*, pages 310–319, 2007.

[5] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[6] A. Hinneburg and D. Keim. An efficient approach to clustering in multimedia databases with noise. In *4th Int'l Conf. on Knowledge Discovery and Data Mining*, 1999.

[7] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[8] G. Karypis, E.-H. S. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[9] J. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Springer, 2007.

[10] A. Y. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proc. of NIPS*, 2001.

[11] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, 1992.

[12] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[13] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[14] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *18th Annual Conference on NIPS*, 2004.

[15] Y. Zhao and G. Karypis. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, 2005.