Knowl Inf Syst (2009) 21:201–229
DOI 10.1007/s10115-009-0216-0

REGULAR PAPER

SPARCL: an effective and efficient algorithm for mining arbitrary shape-based clusters

Vineet Chaoji · Mohammad Al Hasan · Saeed Salem · Mohammed J. Zaki

Received: 30 December 2008 / Revised: 28 February 2009 / Accepted: 3 April 2009 /
Published online: 3 June 2009
© Springer-Verlag London Limited 2009

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.

Keywords Clustering · Spatial · Kmeans · Hierarchical · Linear time

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 [21], market research [34], privacy and security [23], image analysis [38], web search [43] and so on.

This work was supported in part by NSF Grants EMT-0829835, and CNS-0103708, and NIH Grant 1R01EB0080161-01A1.

V. Chaoji (✉) · M. Al Hasan · S. Salem · M. J. Zaki
Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
e-mail: chaojv@cs.rpi.edu

Due to the large number of potential application domains, many flavors of clustering algorithms have been proposed [22,31]. 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. For instance, if the data instance is modeled as a point in d-dimensional linear subspace, Euclidean distance generally works well. However, in applications like image segmentation or spatial data mining, Euclidean distance based measure does not generate the desired clustering solution. Clusters in these applications generally are dense set of points that can represent (physical) objects of arbitrary shapes. The Euclidean distance measure 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 [38], density-based (DBSCAN [13]), and nearest-neighbor graph based (Chameleon [24]) 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 hypersphere. When the number of centers are few, each such hypersphere covers a larger region, thus leading to incorrect partitioning of a dataset with arbitrary shapes. Increasing the number of 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.

The rest of the paper is organized as follows. The next section provides a comprehensive outline of related work. Section 3 describes the SPARCL approach followed by an experimental evaluation of our proposed method in Sect. 4.

2 Related work

A comprehensive survey of arbitrary shape clustering with a focus toward spatial clustering is provided in [31]. Here we review some of the pioneer methods.

DBSCAN [13] was one of the earliest algorithms that addressed arbitrary shape clustering. It defines two parameters—*eps* which is the radius of the neighborhood of a point, and *MinPts* which is the minimum threshold for the number of points within *eps* radius of a point. A point is labeled as a *core point* if the number of points within its *eps* neighborhood is at least *MinPts*. Based on the notion of density-based reachability, a cluster can be defined as the maximal set of *reachable* core points, i.e., such that each core point is within the *eps* neighborhood of at least one other core point in the cluster. Other (border) points that are with the neighborhood of core points are also added to the same cluster (ties are broken arbitrarily or in the order of visitation). Points that are not core and not reachable from a core are labeled as noise. The main advantages of DBSCAN are that is 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 also 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 [19,20] is a density based clustering algorithm based on kernel density estimation. DENCLUE models the impact of a data point within its neighborhood as an *influence function*. The influence function is defined in terms of the distance between the two points. The *density function* at a point in the data space is expressed in terms of the influence functions acting on that point. Clusters are determined by identifying *density attractors* which are local maximas of the density function. The density attractors are identified by performing a gradient ascent type algorithm over the space of influence functions. Both center-defined and arbitrary-shaped clusters can be identified by finding the set of points that are density attracted by a density attractor. 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 $O(n^2)$. The recent DENCLUE2.0 [18] method practically speeds up the time by adjusting the step size in the hill climbing approach. An extension [11] of DENCLUE, proposes a grid approximation to deal with large datasets.

The arbitrary shape clustering problem has also been modeled as a hierarchical clustering task. For example, Kaufman and Rousseeuw [26] proposed one of the earliest agglomerative method that can handle arbitrary shape clusters, which they termed as elongated clusters. They compute the similarity between two clusters A and B as the smallest distance between a pair of objects from A and B, respectively. This method is computationally very expensive due to the expensive similarity computations, with a complexity of $O(n^2 \log n)$. Moreover, presence of outlier points between the boundary region of two distinct clusters can cause wrong merging decisions. In a recent work [27], the authors propose a hierarchical clustering algorithm based on an approximate nearest neighbor search—*Locality-Sensitive Hashing*. This approach considerably improves the time complexity of the algorithm.

CURE [16] is another hierarchical agglomerative clustering algorithm that handles shape-based clusters. It follows the nearest neighbor distance to measure the similarity between two clusters as in [26], 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. To ensure that the representative points are not outlier points, the representatives are pulled in, by a predetermined factor, toward the mean of the cluster. CURE is still expensive with its quadratic complexity, and more importantly, the quality of clustering depends enormously on the sampling quality. In [24], the authors show several examples where CURE failed to obtain the desired shape-based clusters.

CHAMELEON [24] also formulates the shape-based clusters as a hierarchical clustering problem over 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 sub-clusters). The partitioned sub-graphs are then merged to obtain the desired number of final k clusters. CHAMELEON introduces two measures—*relative inter-connectivity* and *relative closeness*—that determine if a pair of clusters can be merged. Relative interconnectivity is defined as ratio of the total edge cut between the two sub-clusters and the mean *internal connectivity* of the sub-clusters. The internal connectivity is defined as the weight of the cut that divides a sub-cluster into equal parts. The relative interconnectivity measure ensures that sub-clusters having a small bridge connecting them are not merged together. Relative closeness is the ratio of the *absolute closeness* to the *internal closeness* of the two sub-clusters, where absolute closeness is the mean edge cut between the two clusters, and the internal closeness of a cluster is the average edge cut that splits it into two equal parts. Relative closeness ensures that the two merged sub-clusters have the same density. Moreover, this measure ensures that the distance between the two sub-clusters is comparable with their internal densities. 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. BIRCH [44] is another hierarchical clustering algorithm that can identify clusters with arbitrary shapes.

Proposed in the pattern recognition community, the spectral clustering approach of Shi and Malik [38] is also capable of handling arbitrary shaped clusters. They represent the data points as a weighted undirected graph, where the weights denote the similarities. 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*. The basic idea is to partition the similarity graph based on the second largest eigenvector of the Laplacian matrix. If the desired number of clusters are not obtained the subgraphs are further partitioned using the lower eigenvectors as approximations for the second eigenvector of the subgraphs. The intuitive reason of its success is its alternate similarity measure which is shape-insensitive. Meila and Shi [30] shows that the similarity between two data points in the normalized-cut framework is equivalent to their connectedness with respect to the random walks in graph, where the transition probability between nodes is inversely proportional to the distance between the pair of points. 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 [42], but all suffer from the poor scalability problem.

A wavelet based shape clustering method was also proposed by Sheikholeslami et al. [37]. In [9], the authors propose a two stage EM algorithm wherein a larger set of parameters l ($l > k$) is initialized, then subsequently pruned before a second round of EM is run on the remaining set. Their approach though different, comes close in essence to our proposed method. Note that our focus in this paper is on full-space shape-based clusters. However, when low dimension data with specific shapes are embedded in a high dimensional space as manifolds, subspace shape clustering can be useful. The current approaches to handle such cases is to unfold the manifold to reduce the dimensionality and then to apply clustering algorithms on the new representation [39]. We refer the reader to Lee and Verleysen [28] for more details on nonlinear dimensionality reduction.

Another line of research from the neural networks community has a distant resemblance with our work. In [2,29], the authors propose a method for representing the topology of a manifold by a grid network.

Our proposed method SPARCL is based on the well known family of Kmeans based algorithms, which are widely popular for their simplicity and efficiency [41]. 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. Kmeans based methods are characterized by $O(n\ d\ k\ e)$ time complexity, where e represents the number of iterations the algorithm runs before convergence. They are related to Voronoi tessellation, which leads to convex polytopes in metric spaces [33]. 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* [35] 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 [7]. In this work, we approximate the convex decomposition of an arbitrary shape cluster by the convex polytopes 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 polytope. One can immediately 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*

```
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.     $S(i,j)$ = compute_similarity($C_i, C_j$)
5. cluster_centers($C_{seed}, S, k$)
```

Fig. 1 The SPARCL algorithm

(*MCSC*) [15]. 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. In our case, S is the set of points and c the convex polytope 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.

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

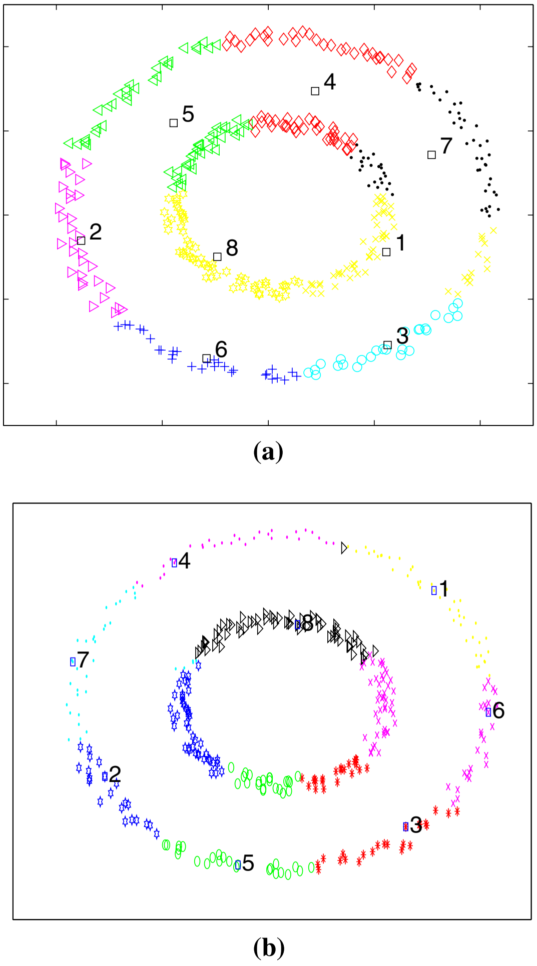


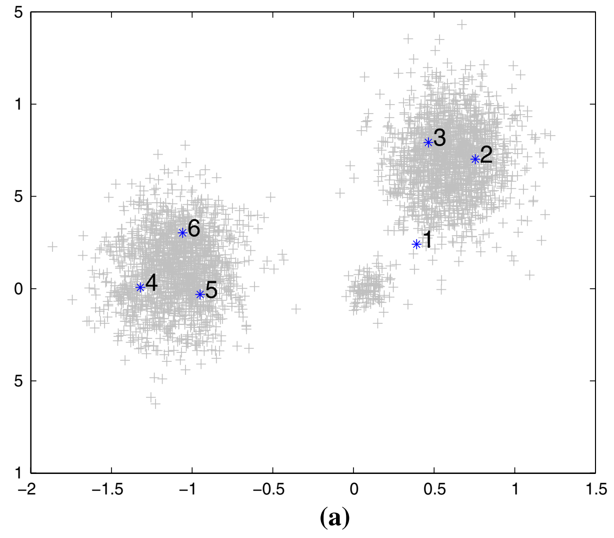
Fig. 2 Effect of choosing mean or actual data point. **a** Using mean point, **b** Using actual data point

get easily influenced by outliers. Figure 2a 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. 2b.

This phase starts by 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_{seeds} 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. Figure 3 shows an example of such a case. In Fig. 3a seed center 1 is almost an outlier point. As a result the members belonging to seed center 1 come from two



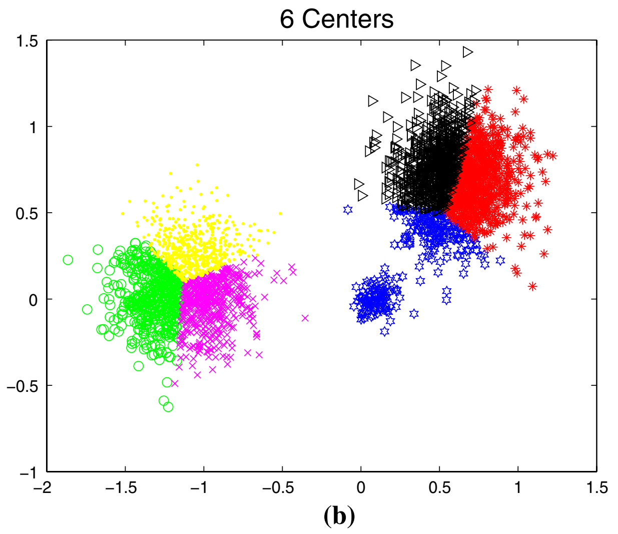


Fig. 3 Bad choice of cluster centers. **a** Randomly selected centers, **b** Natural cluster split by bad center assignment

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 our recently proposed outlier and density insensitive based selection of initial centers [17]. Let us take a quick look at other initialization methods before discussing our Local Outlier Factor based initialization technique.

Kmeans initialization methods Although there are numerous initialization methods, we briefly discuss some of the key ones. One of the first schemes of center initialization was proposed by Ball and Hall [4]. They suggested use of a user defined threshold, d, to ensure that the seed points are well apart from each others. The first point is chosen as a seed, and for any subsequent point considered, it is selected as a seed if it is at least d distance apart from already chosen seeds, until k seeds are found. With a right choice of the value of d, this approach can restrict the splitting of natural clusters, but guessing a right value of d is very difficult and the quality of seeds depends on the order in which the data points are considered.

Astrahan [1] suggested using two distance parameters, d_1 and d_2. The method first computes the *density* of each point in the dataset, which is given as the number of neighboring points within the distance d_1, and it then sorts the data points according to decreasing value of density. The highest density point is chosen as the first seed. Subsequent seed point are chosen in order of decreasing *density* subject to the condition that each new seed point be at least at a distance of d_2 from all other previously chosen seed points. This step is continued until no more seed points can be chosen. Finally, if more than k seeds are generated from the above approach, hierarchical clustering is used to group the seed points into the final k seeds. The main problem with this approach is that it is very sensitive to the values of d_1 and d_2. Furthermore, users have very little knowledge regarding the good choices of these parameters, and the method is computationally very expensive. A range search query needs to be made for every data point followed by a hierarchical clustering of a set of points. Small values of d_1 and d_2 may produces enormously large number of seeds, and hierarchical clustering of those seeds can be very expensive ($O(n^2 \log n)$ in the worst case). This method also performs poorly when there exist different clusters in the dataset with variable density and size.

Katsavounidis et al. [25] suggested a parameterless approach, which we call the KKZ method based on the initials of all the authors. KKZ chooses the first centers near the “edge” of the data, by choosing the vector with the highest norm as the first center. Then, it chooses the next center to be the point that is farthest from the nearest seed in the set chosen so far. This method is very inexpensive ($O(kn)$) and is easy to implement. It does not depend on the order of points and is deterministic by nature; as single run suffices to obtain the seeds. However, KKZ is sensitive to outliers, since the presence of noise at the edge of the dataset may cause a small set of outlier/noise points to make up a cluster.

Bradley and Fayyad [5] proposed an initialization method that is suitable for large datasets. We call their approach Subsample, since they take a small subsample (less than 5%) of the dataset and use k-means clustering on the subsample and record the cluster centers. This process is repeated and cluster centers from all the different iterations are accumulated in a dataset. Finally, a last round of k-means is performed on this dataset and the cluster centers of this round are returned as the initial seeds for the entire dataset. This method generally performs better than k-means and converges to the local optimal faster. However, it still depends on the random choice of the subsamples and hence, can obtain a poor clustering in an unlucky session.

More recently, Arthur and Vassilvitskii [3] proposed the k-means++ approach, which is similar to the KKZ method. However, when choosing the seeds, they do not choose the farthest point from the already chosen seeds, but choose a point with a probability proportional to its distance from the already chosen seeds.

Initialization using local outlier factor We chose the *local outlier factor* (*LOF*) criterion for selecting the initial set of cluster centers. LOF was proposed in [6] 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 mpth nearest neighbor of x. Thus $N(x, mp)$ contains at least mp points. The *density* of x is then computed as follows:

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

Essentially, the lower the distance between x and neighboring points, the higher the density of x. The *average relative density* (*ard*) of x, is then computed as the ratio of the density of x and the average density of its nearest neighbors, given as follows:

$$ard(x, mp) = \frac{density(x, mp)}{\left(\frac{\sum_{y \in N(x,mp)} density(y,mp)}{|N(x,mp)|} \right)}$$

Finally the LOF score of x is just the inverse of the average relative density of x:

$$\text{LOF}(x, mp) = ard(x, mp)^{-1}$$

If a point is in a low density neighborhood compared to all its neighbors, then its *ard* score is low and hence its LOF value is high. 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. That is, for a large range of values of *mp*, the outlier points will have LOF value well above 1, whereas points belonging to a cluster will assume an LOF value close to 1. (3) It leads to practically faster convergence of the Kmeans algorithm, i.e., fewer iterations.

As we reported in [17], to select the initial seeds, we use the following approach. Assume that i initial centers have been chosen. To choose the $i + 1$th center, we 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, we pick the first point that is not an outlier as the next seed. We repeat until K initial seeds have been chosen, and then run the Kmeans algorithm to converge with those initial starting centers, to obtain the final set of seed centers C_{seed}.

Complexity analysis of LOF based initialization The overall complexity of this approach can be analyzed in terms of the steps involved. Let us assume that $t \ll n$ is the number of outliers in the data. While choosing the $i + 1$th center, the minimum distance of each of the $n - i$ non-center points from the i centers is computed. Aggregated over the K centers, the total computational cost of this step amounts to $O(nKd)$, where d is the dimensionality of the data. Once the minimum distances are computed, the $i + 1$th center is chosen by examining points in descending order of minimum distance and selecting the point that has an LOF value close to 1. The linear-time *partition-based selection algorithm* [8] for computing the pth largest number can be used to find the points in descending order. In the worst case, the selection algorithm has to be invoked t times (with $p = 1, \dots, t$) for the $i + 1$th center selection. If t is a small constant, the selection based approach can be much more efficient as compared to the $O(n \log n)$ sorting based selection algorithm. The aggregated computational cost for the selection phase of K centers is given by $O(tnK)$. The LOF value for each examined point is computed during the selection stage. The cost of computing the LOF value of a point is given by $O(n^{1-\frac{1}{d}} \times mp)$. In the worst case, for each center t LOF computations need to

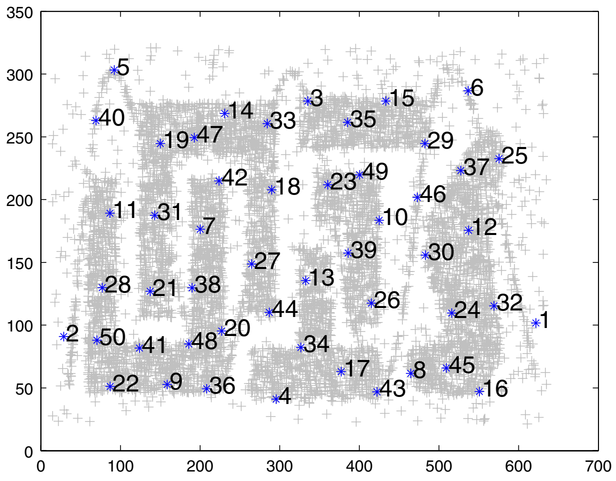


Fig. 4 Local outlier based center selection. Selected centers on D1 dataset

be performed. As the result, the LOF computation aggregated over K centers comes out to $O(n^{1-\frac{1}{d}} \times mp \times t \times K)$. Finally, adding up the costs of the above steps, the complexity of the entire process is given by $O(nKd + tnK + n^{1-\frac{1}{d}} \times mp \times t \times K)$. As seen from the previous expression, the overall time complexity is linear in the number of points in the data.

As an example of LOF-based seed selection, Fig. 4 shows the initial set of centers for one of the shape-based datasets. Section 4.2 provides an empirical comparison of LOF based initialization with other initialization methods.

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.

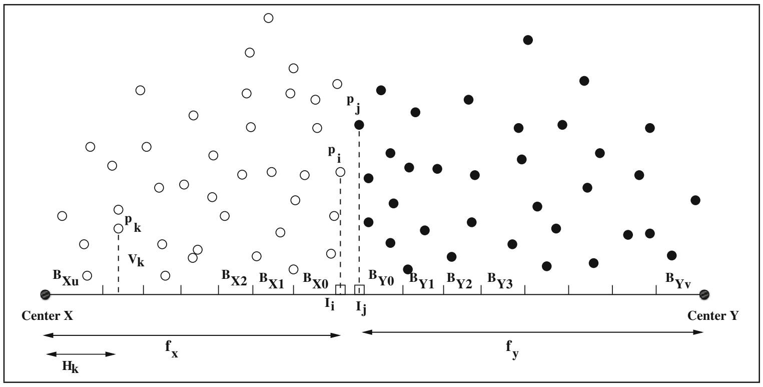


Fig. 5 Projection of points onto the vector connecting the centers

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. $\max\_\text{bin}_i = \arg\max_j d_{i_j}$ 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 $\frac{N[X_i]}{N[X_{\max\_\text{bin}_X}]}$ 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:

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

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:

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

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

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

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 [14], but the additional computational cost does not justify the improvement in the quality of the results.

Finally, once the similarity between pairs of seed has been computed, 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.

Our similarity metric $S(X, Y)$ can be shown to be a *kernel*. The following lemmas regarding kernels allow us to prove that the similarity function is a kernel.

Lemma 3.1 [36] *Let κ_1 and κ_2 be kernels over $\mathbf{X} \times \mathbf{X}$, $\mathbf{X} \subseteq \mathbb{R}^n$, and let $f(.)$ be 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 3.2 [36] *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 3.3 *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 3.1(ii). This makes $\exp(-dist\_sim(.,.))$ a kernel by Lemma 3.2. Product of $size\_sim$ and $\exp(-dist\_sim(.,.))$ is a kernel by Lemma 3.1(iii). And finally, $S(X, Y)$ is a kernel since the sum of kernels is also a kernel by Lemma 3.1(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 method, such as spectral clustering [32] or kernel k-means [10], 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 $O(mp\,K^2\,n\,t)$ [17], 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.

3.3 Estimating the value of K

As discussed in Sect. 3.1.2, neighboring clusters are merged to obtain the final set of k natural clusters. A "good" clustering is guaranteed if the following conditions are satisfied:

1. **Merging Condition:** Only the pseudo-centers belonging to a single natural cluster are merged together,
2. **Pseudo-center Condition:** No pseudo-center exists such that it is a representative for points belonging to more than one natural cluster.

Although, the effectiveness of the similarity measure and the merging process influence the *Merging Condition*, the similarity measure computation relies on the existence of the Pseudo-center Condition. Hence, satisfying the second condition is crucial for obtaining a good clustering result. The value of K can adversely influence the Pseudo-center condition. Underestimating K can result in points from two or more natural clusters being assigned to the same pseudo-center, since each center has to now account for a larger number of points. This is emphasized by our results (Sect. 4.3.5) in Fig. 13b, which shows a lower clustering quality score for smaller values of K. Hence, having a good estimate of K can considerably improve the clustering quality. At the same time, the clustering outcome is not sensitive to small changes in the value of K, which implies that a rough estimate suffices.

In many application domains the expert has an insight into approximate distances between natural clusters. For instance, cell biologists might have an estimate of the distance between nearby chromosomes; a radiologist might have an intuition regarding average distance between bones in an X-ray image; or distance between regions of interest on a weather forecast map might be known a priori. Let us assume that an expert can estimate the minimum distance between any true clusters, denoted by $cdist_{\min}$. Given $cdist_{\min}$, we can estimate the value of K such that the *Pseudo-center Condition* is satisfied. Figure 6a shows the true clusters for an illustrative dataset, with the noise points removed. The figure also shows the $cdist_{\min}$ for this dataset.

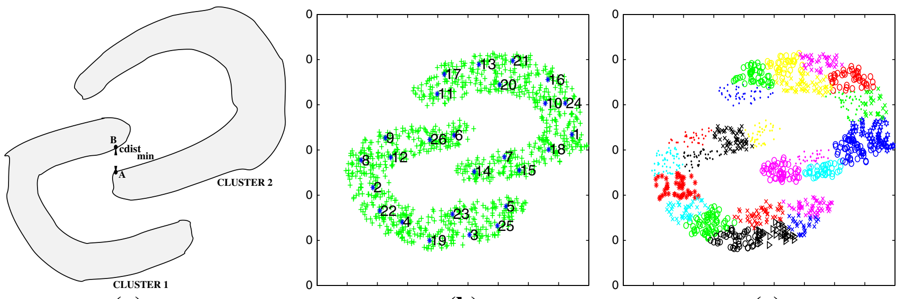


Fig. 6 Estimating K. **a** Estimating K from $cdist_{\min}$, **b** Seed centers with $cdist_{\min} = 20$, **c** Members of each seed cluster with $cdist_{\min} = 20$

```
seed_center_initialization($\mathcal{D}, cdist_{min}, mp$):
1.  Take any reference point, $r$ (origin suffices)
2.  Insert $r$ in $\mathcal{C}$
3.  do
4.      sort the points in $\mathcal{D}$ in decreasing order of
5.      minimum distance from points in $\mathcal{C}$
6.      for each $x$ in sorted order
7.          if (LOF$(x, mp) \approx 1$)
8.              insert $x$ in $\mathcal{C}$
9.              $min\_dist = dist_x$
10.             break
11.         endif
12.     endfor
13. while $min\_dist \geq cdist_{min}$
14. remove $r$ from $\mathcal{C}$
15. return $\mathcal{C}$
```

Fig. 7 Generating seed representatives with $cdist_{\min}$

Assume point A is selected as one of the pseudo-centers. In order to assign point B to a center other than A, there has to be another center closer than $cdist_{\min}$ to point B. Any point closer than $cdist_{\min}$ to B has to belong to Cluster 1. In other words, if the nearest center for each point is at a distance less than $cdist_{\min}$, condition two is satisfied. If the dataset is scattered over a 2-dimensional region with area (volume for higher dimensions) A, then the value of K is given by $\lceil \frac{A}{2\pi cdist_{\min}} \rceil$. The $2\pi cdist_{\min}$ expression is an approximation for the area of a convex polyhedron around a center point. The above expression can be similarly generalized for higher dimensional datasets to $\lceil \frac{Vol.\ occupied\ by\ cluster\ shape}{Vol.\ of\ polyhedron\ with\ radius\ cdist_{\min}} \rceil$. The area or volume occupied by the clusters does not have to be computed explicitly. Based on the above idea, the LOF based algorithm (Sect. 3.1.1) for obtaining initial seeds can be modified to automatically select the required number of seeds. Figure 7 shows the modified LOF initialization algorithm. The algorithm **seed_center_initialization** takes as input the dataset of points $\mathcal{D}$, $cdist_{\min}$ and a parameter for LOF computation. The algorithm returns the set of seed pseudo-centers. In the LOF based initialization, it can be shown that each subsequently selected seed representative results in a monotonic decrease in the minimum distance (min_dist). As a result, the condition on Line 13 is violated after a finite number of points. At the time the condition in the while loop is violated, the maximum distance of a point to its nearest seed representative is less than $cdist_{\min}$. As a result of which, no pseudo-cluster has points from more than one natural cluster. The *LOF* function on Line 7 computes the Local Outlier Factor for a point. Recall that the LOF value is an indicator of the degree to which a point is an outlier. A value close to 1 signifies a point is not an outlier.

We would like to point out that this is a *worst case analysis* which guarantees the existence of the *Pseudo-center Condition*. On the downside, for pathological datasets much larger number of seed representatives could be selected as compared to what might suffice for a good clustering. This can be seen in the results for dataset DS2 wherein a good clustering is obtained with $K = 60$ (as shown in Fig. 9b). Applying the algorithm in Fig. 7 generates $K = 287$ seed representatives with $cdist_{\min} = 20$. For the dataset in Fig. 6a, the seed centers and the points assigned to them are shown in Fig. 6b, c, respectively. As one can see in Fig. 6b, the LOF based initialization generates centers that are uniformly distributed. For regular shaped clusters, the Pseudo-center Condition can be preserved even with a non-uniform distribution of the centers as long as points such as A and B are not assigned to the same seed-cluster. This approach could result in a reduced number of seed centers and a faster overall computation time. In this work, we do not address the non-uniform selection of seed centers.

4 Experiments and results

Experiments were performed to compare the performance of our algorithm with Chameleon [24], DBSCAN [13] and spectral clustering [38]. The Chameleon code was obtained as a part of the CLUTO [45][1] package. The DBSCAN implementation in Weka was used for the sake of comparison. Similarly, for spectral clustering the *SpectraLIB* Matlab implementation[2] based on the Shi–Malik algorithm [38] was used initially. Since this implementation could not scale to larger datasets, we implemented the algorithm in C++ using the *GNU Scientific Library*[3] and *SVDLIBC*.[4] Even this implementation would not scale to very large datasets since the entire affinity matrix would not fit in memory. The results in Table 2 for spectral clustering are based on this implementation.

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 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 Fig. 9a–d, are those that have been used in previous studies including

[1] http://glaros.dtc.umn.edu/gkhome/cluto/.

[2] http://www.stat.washington.edu/spectral/.

[3] http://www.gnu.org/software/gsl/.

[4] http://tedlab.mit.edu/~dr/svdlibc/.

Chameleon and CURE. These are all 2d datasets with points ranging from 8000 to 100,000. The Swiss-roll dataset in Fig. 9 is the classic non-linear manifold using in non-linear dimensionality reduction [28]. We simply 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 the points which lie in our desired shape. All the shapes are generated with point (0, 0) as the origin. To get complex shapes, we combine rotated and translated basic shapes (circle, rectangle, ellipse, circular strip, etc.). Our 3d shape generation is built on the 2d 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 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 (1–2%) to the drawing frame. An example of a synthetic 3d dataset is shown in Fig. 12b. This 3d dataset has 100,000 points, and 10 clusters.

4.1.2 Real datasets

We used two real shape-based datasets: cancer images and protein structures. The first is a set of 2d images from benign breast cancer.[5] 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.

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 need to be centered such that the minimum point (atom) is above the (0, 0, 0) origin. We translate the proteins to get separated clusters. Once the translation is done, we add the noise points. Our protein dataset has 15,000 3d points obtained from the following proteins: 1A1T (865 atoms), 1B24 (619 atoms), 1DWK (11,843 atoms), 1B25 (1,342 atoms), and 331 noise points.

4.2 Comparison of Kmeans initialization methods

We compared different initialization methods on a set of synthetic datasets. These datasets contain regular shaped clusters of varying sizes and densities. The exact details of the datasets are omitted as they do not influence the comparative results. Since the clusters generated by Kmeans are hyperspheres, **distortion score** is used as the evaluation metric for comparing the initialization methods. *Distortion Score* is defined as the sum of the distance between each point and its closest center. Smaller value of distortion implies a better clustering.

Multiple runs are performed for algorithms that depend either on the order of objects in the dataset or on randomization. For those algorithms, the minimum and average distortion values are shown. Results for different dimensions (d) and number of natural clusters (k) are shown in Table 1. The optimal distortion measure is also shown since the datasets are synthetically generated. The bold number in each row of Table 1 indicates the lowest distortion score for that dataset. The results show that the LOF based initialization performs better on most of the datasets. Results in Fig. 8 highlight the robustness of LOF-based initialization as compared to random initialization. In Fig. 8a, the distortion score is computed for varying percentage of random noise in the dataset. Again, a lower distortion score indicates

[5] These were obtained from Prof. Bulet Yener at RPI.

Table 1 Comparison on synthetic datasets

d	k	Optimal	LOF	Random		Subsample		k-means++		KKZ
				Min	Avg	Min	Avg	Min	Avg	
8	10	7,738	**7,755**	7,904	8,421	7,887	8,092	8,008	8,508	9,204
	25	9,365	**9,382**	9,774	10,185	9,639	10,044	9,641	9,951	10,743
	50	8,694	**8,754**	9,244	9,565	9,136	9,407	9,289	9,598	17,042
16	10	16,865	16,882	17,406	18,496	17,356	18,314	**16,870**	17,951	19,346
	25	17,241	**17,261**	18,298	19,219	17,647	18,812	17,732	18,550	20,567
	50	17,580	**17,622**	18,866	19,507	18,469	19,084	18,974	19,661	21,632
24	10	26,149	**26,150**	26,706	28,733	**26,150**	28,340	26,755	28,860	29,413
	25	22,233	**22,261**	23,241	24,582	23,034	23,942	22,803	23,787	27,052
	50	21,453	**21,467**	22,838	23,818	22,477	23,387	23,003	23,762	26,599

The distortion scores are shown for each method

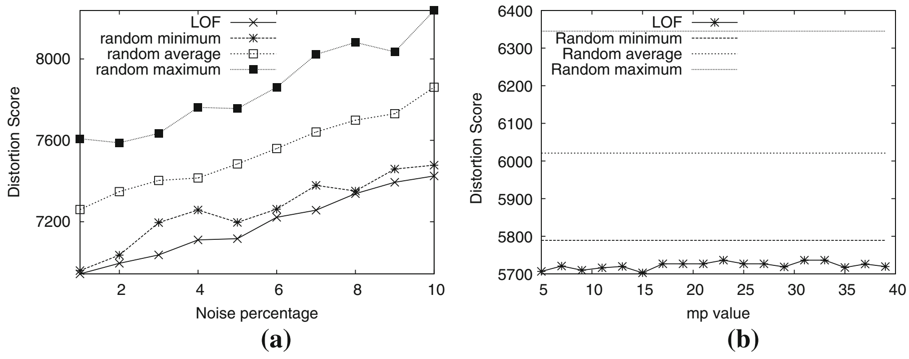


Fig. 8 Sensitivity comparison: LOF versus random. **a** Random noise, **b** $d = 8$, $k = 15$

robustness to random noise. Similarly, Fig. 8b shows that the LOF based initialization is robust to small changes in the parameter (mp) value.

4.3 Results on synthetic datasets

Results of SPARCL on the synthetic datasets are shown in Table 2, and in Fig. 9a–e. We refer the reader to Karypis et al. [24] for the clustering results of Chameleon and DBSCAN on datasets DS1-4. In essence, 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 had no difficulty in identifying the shape-based clusters in these datasets. However, SPARCL does make minor mistakes at the boundaries in the Swiss-roll dataset (Fig. 9). 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. In other words, it is a nonlinear subspace cluster. What is remarkable is that SPARCL can actually find a fairly good clustering even in this case.

Table 2 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

Table 2 Runtime performance on synthetic datasets

Name	$\|D\|(d)$	k	SPARCL (LOF/Random)	Chameleon	DBSCAN	Spectral
DS1	8,000(2)	6	5.74/1.277	4.02	14.16	199
DS2	10,000(2)	9	8.6/1.386	5.72	24.2	380
DS3	8,000(2)	8	6.88/1.388	4.24	14.52	239
DS4	100,000(2)	5	35.24/20.15	280.47		–
Swiss-roll	19,200(3)	4	23.92/17.89	19.38		–

All times are reported in seconds. "–" for Spectral method denotes the fact that it ran out of memory for all these cases

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 shows 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.

4.3.1 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. Figure 10 compares the runtime performance of Chameleon, DBScan and our approach for dataset sizes ranging from 100,000 points to 1 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 100 K points. Figure 10 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 with increase in the size of the dataset. For Chameleon, the parameters *agglofrom, 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.

Figure 11 shows the clusters obtained as a result of executing our algorithm, DBSCAN and Chameleon on the dataset DS5 of size 50 K 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. 12b. The 3d dataset consists of shapes in full 3d space

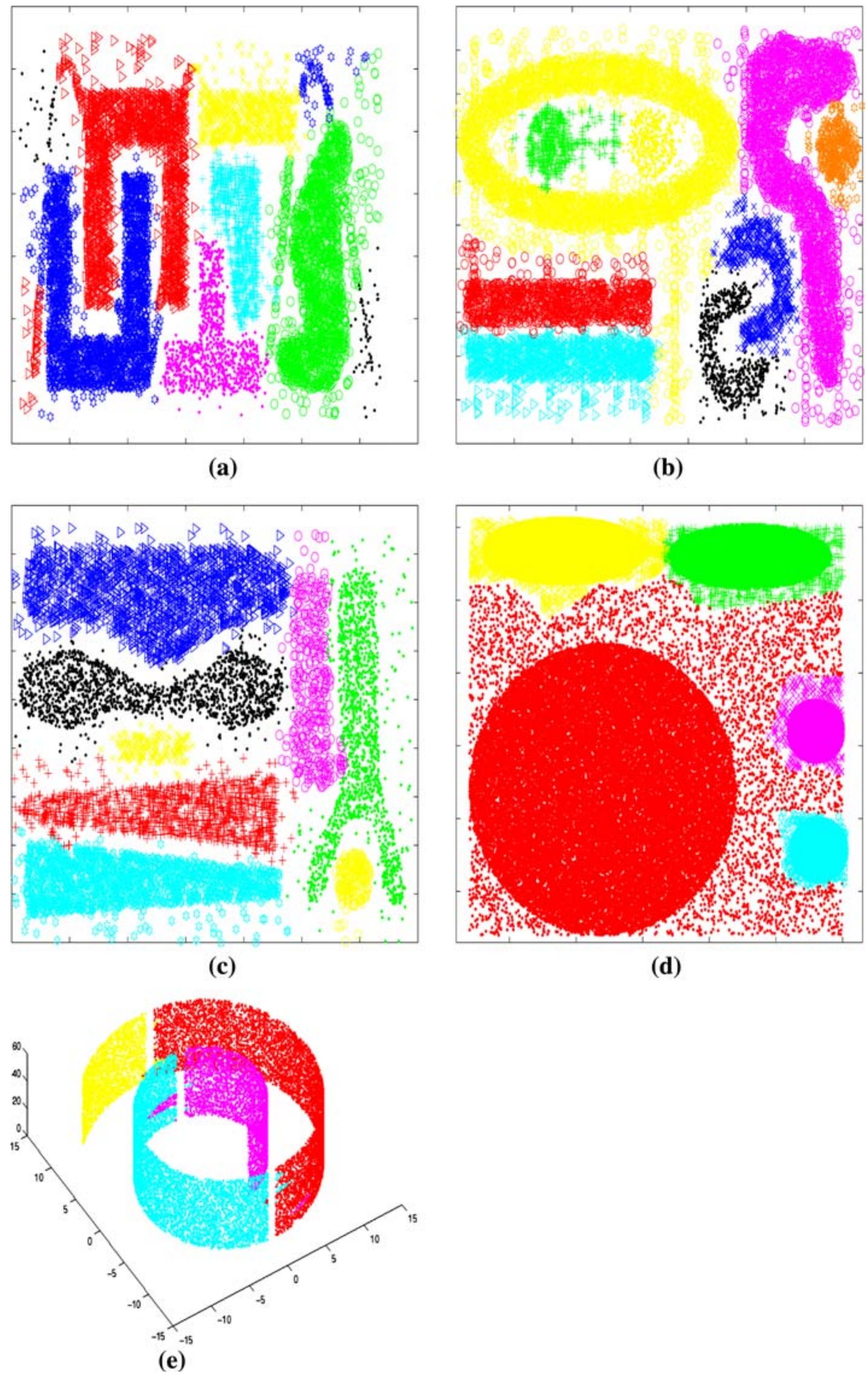


Fig. 9 **a** Results on DS1, **b** Results on DS2, **c** Results on DS3, **d** Results on DS4, **e** Results on Swiss-roll

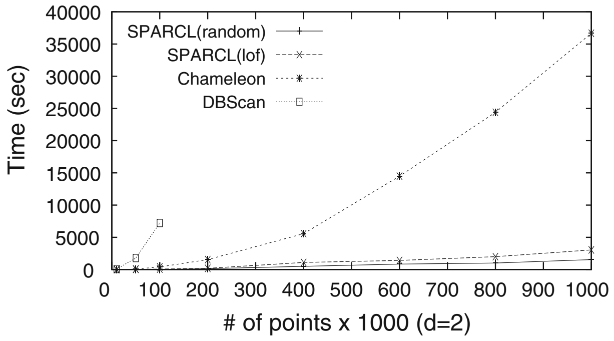


Fig. 10 Scalability results on dataset DS5

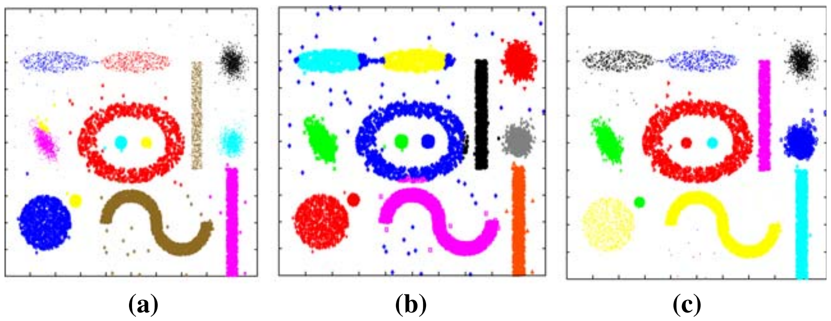


Fig. 11 Clustering quality on dataset DS5. **a** SPARCL ($K = 100$), **b** DBSCAN ($minPts = 150, eps = 0.05$), **c** Chameleon ($agglofrom = 30$, $sim = dist$, $clmethods = graph$)

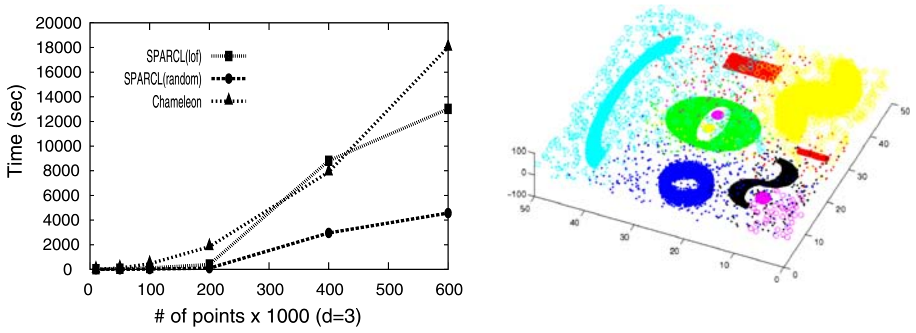


Fig. 12 Clustering results on 3D dataset

(and not 2d shapes embedded in 3d space). The dataset contained random noise too (2% of the dataset size). As seen in Fig. 12a, SPARCL (with random initialization) can be more than four times as fast as Chameleon.

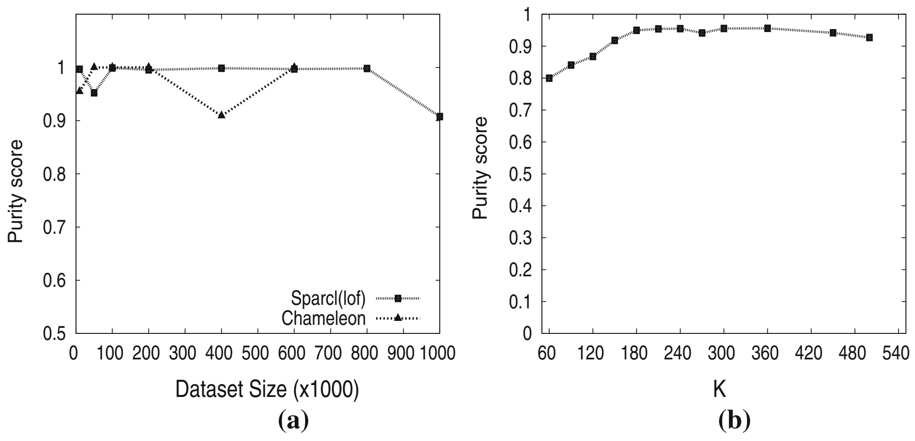


Fig. 13 Cluster quality. **a** Varying dataset size, **b** Varying # of seed-clusters

4.3.2 Clustering quality

Since two points in the same cluster can be very far apart, traditional metrics such as cluster diameter, Kmeans/Kmedoid 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 a priori, 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:

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

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. Figure 13a 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 600 K, 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, whereas SPARCL's purity score is more than 0.9.

4.3.3 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

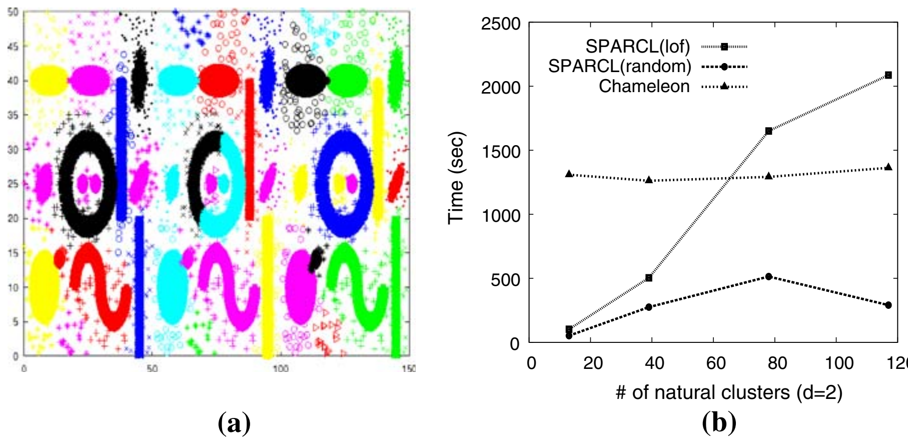


Fig. 14 Varying number of natural clusters. **a** 1 × 3 grid tiling. Results of SPARCL, **b** Varying number of natural clusters

(see Fig. 14a). The number of points are held constant at 180 K. 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., $K = 5\,\mathrm{K}$. 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. 14b).

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.

4.3.4 Varying number of dimensions

Synthetic data generator SynDECA [40] (http://cde.iiit.ac.in/~soujanya/syndeca/) was used to generate higher dimensional datasets. The number of points and clusters were set to 500 K and 10, respectively. 5% of 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. Figure 15b 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. Figure 15a shows the purity for higher dimensions. Both SPARCL and CHAMELEON perform well on this measure.

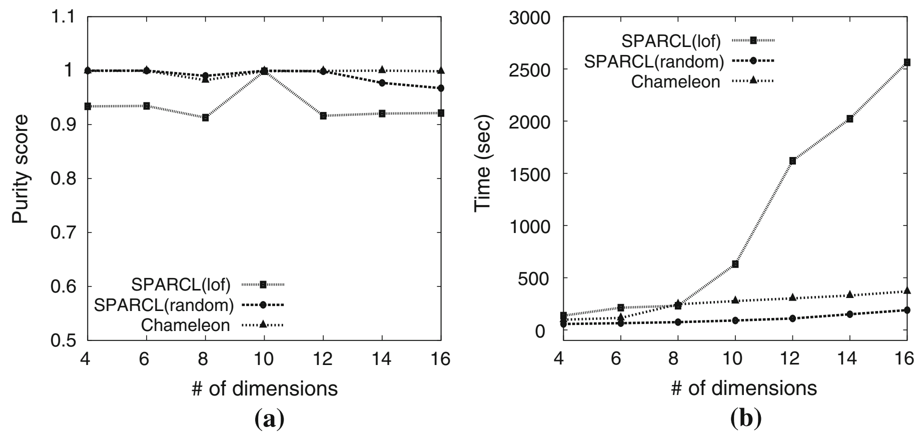


Fig. 15 Varying number of dimensions. **a** Purity, **b** Runtime

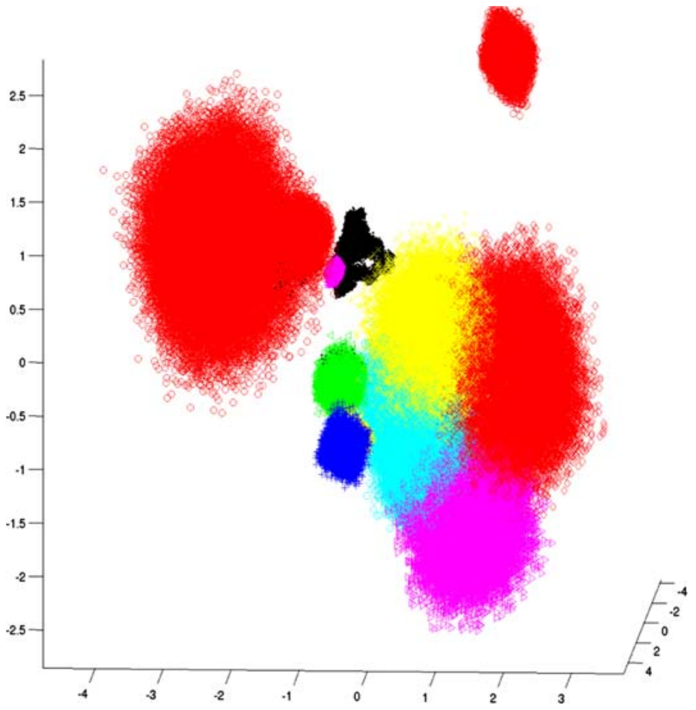
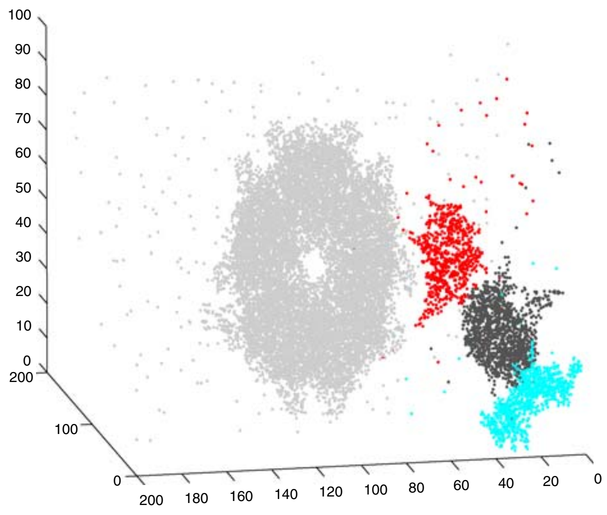


Fig. 16 Ten dimensional dataset (size = 500 K, k = 10) projected onto a 3D subspace

The quality of the clustering can also be visually inspected. The points in the high dimensional space can be projected onto a lower dimensional space. For our experiments, we use Principal Component Analysis [12] (PCA) as the dimensionality reduction technique. PCA has the distinction of being a linear transformation that is optimal for preserving the subspace with the largest variance in the data. Figure 16 shows the above dataset with 10 dimensions projected onto a 3-dimensional subspace. The noise points have been purposely suppressed

Fig. 17 Protein dataset



in order to view the projected clusters clearly. As seen in Fig. 16, the compact regions representing the clusters contain points from the same natural clusters. Projecting points on a lower dimensional sub-space, results in small overlap of some of the clusters.

4.3.5 Varying number of seed-clusters (K)

SPARCL has a single parameter, K. Figure 13b shows the effect of changing K on the quality of the clustering. The dataset used is same as in Fig. 14a, with 300 K 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.

4.4 Results on real datasets

We applied SPARCL on the protein dataset. As shown in Fig. 17 SPARCL is able to perfectly identify the four proteins. The largest doughnut-shaped is the 1DWK protein while the other smaller ones are amoeba like irregular shaped. The K value for the protein dataset is 30. On this dataset, Chameleon returns similar results. The results on the benign cancer datasets are shown in Fig. 18. Here too SPARCL successfully identifies the regions of interest. The K value for this dataset is 100. The distinct clusters in the cancer dataset represent the nuclei, whereas the surrounding region is the tissue. Clusters that are globular in shape correspond to healthy tissues whereas irregular shapes of the nuclei correspond to cancerous tissues. We do not show the time for the real datasets since the datasets are fairly small and both Chameleon and SPARCL perform similarly.

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

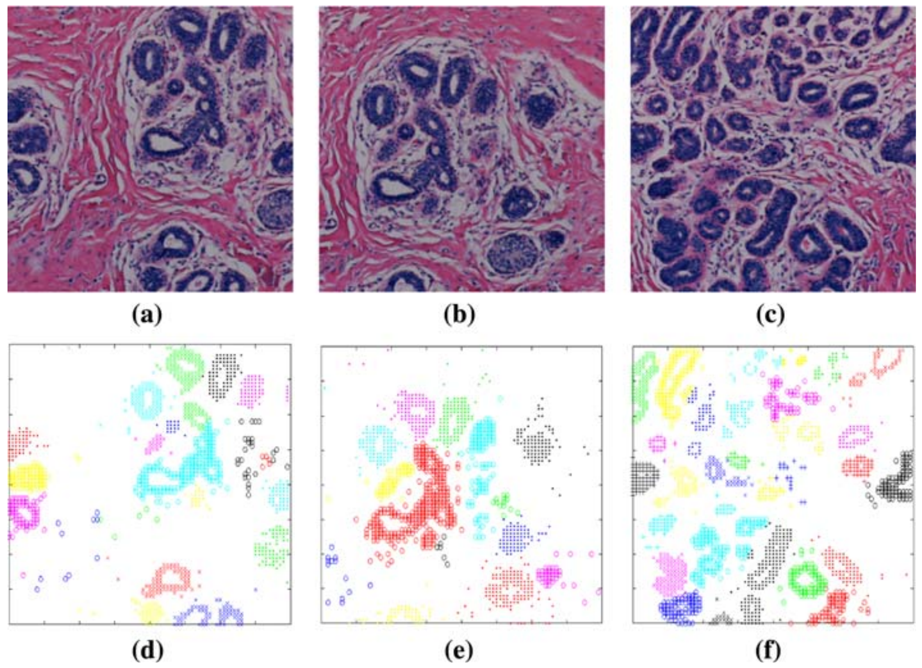


Fig. 18 Cancer dataset: **a**–**c** are the actual benign tissue images. **d**–**f** gives the clustering of the corresponding tissues by SPARCL

selecting a large enough number of seed clusters K, we are able to capture most of the dense areas in the dataset. Next, we compute pairwise similarities between the seed clusters to obtain a $K \times K$ symmetric similarity matrix. The similarity is designed to capture the extent to which the points from the two clusters come close to each other, namely close to the d-dimensional hyperplane that separates the two clusters. 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 bin the distances horizontally and compute a one-dimensional histogram to approximate the closeness of the points, which in turn gives the similarities between the clusters. 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 remarkably effective in finding arbitrary shaped-based clusters in a variety of 2d and 3d datasets. It has the same 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. SPARCL can also find clusters in the classic Swiss-roll dataset, effectively discovering the 2d manifold via Kmeans approximations. It does make some small errors on that dataset. In general SPARCL works well for full-space clusters, and is not yet tuned for subspace shape-based clusters. In fact, find arbitrary shaped-based subspace clusters is one avenue of future work for our method.

Acknowledgments We would like to thank George Karypis for providing the Chameleon datasets.

References

1. Astrahan MM (1970) Speech analysis by clustering, or the hyperphoneme method. Stanford A.I. Project Memo, Stanford University
2. Aupetit M (2003) Robust topology representing networks. In: Proceedings of the European symposium on artificial neural networks, Bruges, Belgium, pp 45–50
3. Arthur D, Vassilvitskii S (2007) k-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, New Orleans, Louisiana, 2007, pp 1027–1035
4. Ball GH, Hall DJ (1967) PROMENADE—an online pattern recognition system. Stanford Research Institute, Stanford University
5. Bradley PS, Fayyad UM (1998) Refining initial points for *k*-means clustering. In: Proceedings of the fifteenth international conference on machine learning Madison, Wisconsin, pp 91–99
6. Breunig MM, Kriegel H, Ng RT, Sander J (2000) LOF: identifying density-based local outliers. In: Proceedings of 2000 ACM-SIGMOD international conference on management of data, Dallas, Texas, 2000, pp 93–104
7. Chazelle B, Palios L (1994) Decomposition algorithms in geometry. In: Bajaj C (ed) Algebraic geometry and its applications. Springer, Berlin pp 419–447
8. Cormen TH, Leiserson CE, Rivest RL, Stein C (2005) Introduction to algorithms, 2nd edn. MIT Press, Cambridge
9. Dasgupta S, Schulman LJ (2000) A two-round variant of EM for Gaussian mixtures. In: Proceedings of the 16th conference on uncertainty in artificial intelligence, Stanford, CA, June 2000, pp 152–159
10. Dhillon IS, Guan Y, Julis B (2004) Kernel k-means, spectral clustering and normalized cuts. In: Proceedings of the tenth international conference on knowledge discovery and data mining, Seattle, WA, August 2004, pp 551–556
11. Domeniconi C, Gunopulos D (2004) An efficient density-based approach for data mining tasks. Knowl Inform Syst 6(6):750–770
12. Duda RO, Hart PE, Stork DG (2000) Pattern classification, 2nd edn. Wiley-Interscience, New York
13. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining, 1996, pp 226–231
14. García JA, Fdez-Valdivia J, Cortijo FJ, Molina R (1995) A dynamic approach for clustering data. Signal Process 44(2):181–196
15. Gao BJ, Ester M, Cai J-Y, Schulte O, Xiong H (2007) The minimum consistent subset cover problem and its applications in data mining. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining San Jose, California, USA, 2007, pp 310–319
16. Guha S, Rastogi R, Shim K (1998) CURE: an efficient clustering algorithm for large databases. In: ACM SIGMOD international conference on management of data, Seattle, WA, pp 73–84
17. Hasan MA, Chaoji V, Salem S, Zaki MJ (2008) Robust partitional clustering by outlier and density insensitive seeding. Technical Report 08-04, RPI Computer Science
18. Hinneburg A, Gabriel H-H (2007) DENCLUE 2.0: fast clustering based on kernel density estimation. In: International symposium on intelligent data analysis, pp 70–80
19. Hinneburg A, Keim DA (2003) A general approach to clustering in large databases with noise. Knowl Inform Syst 5(4):387–415
20. Hinneburg A, Keim DA (1998) An efficient approach to clustering in multimedia databases with noise. In: Proceedings of 4th international conference on knowledge discovery and data mining. AAAI Press, New York
21. Hu X, Pan Y (2007) Knowledge discovery in bioinformatics: techniques, methods, and applications (Wiley Series in Bioinformatics). Wiley, New York
22. Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall, Upper Saddle River
23. Klaus J (2003) Clustering intrusion detection alarms to support root cause analysis. ACM Trans Inform Syst Secur 6(4):443–471
24. Karypis G, Han E-H, Kumar V (1999) Chameleon: hierarchical clustering using dynamic modeling. Computer 32(8):68–75
25. Katsavounidis I, Kuo CCJ, Zhen Z (1994) A new initialization technique for generalized Lloyd iteration. IEEE Signal Process Lett 1(10):144–146
26. Kaufman, L Rousseeuw, PJ (1990) Finding groups in data: An introduction to cluster analysis. Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics, New York
27. Koga H, Ishibashi T, Watanabe T (2007) Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing. Knowl Inform Syst 12(1):25–53

28. Lee J, Verleysen M (2007) Nonlinear dimensionality reduction. Springer, Berlin
29. Martinetz T, Schulten K (1994) Topology representing networks. Neural Netw 7(3):507–522
30. Meila M, Shi J (2001) A random walks view of spectral segmentation. AI and Statistics (AISTATS)
31. Miller HJ, Han J (2001) Geographic data mining and knowledge discovery. Taylor & Francis, Bristol
32. Ng AY, Jordan M, Weiss Y (2001) On spectral clustering: analysis and an algorithm. In: Advances in neural information processing systems, vol 14
33. Okabe A, Boots B, Sugihara K (1992) Spatial tessellations: concepts and applications of voronoi diagrams. Wiley, New York
34. Punj G, Stewart DW (1983) Cluster analysis in marketing research: review and suggestions for application. J Mark Res 20(2):134–148
35. Sack JR, Urrutia J (2000) Handbook of computational geometry. North-Holland, Amsterdam
36. Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge
37. Sheikholeslami G, Chatterjee S, Zhang A (1998) WaveCluster: a multi-resolution clustering approach for very large spatial databases. In: Proceedings of 24th international conference on very large data bases, pp 428–439
38. Shi J, Malik J (2000) Normalized cuts and image segmentation. IEEE Trans Pattern Anal Mach Intell 22(8):888–905
39. Tenenbaum JB, de Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. Science 290(5500):2319–2323
40. Vennam JR, Vadapalli S (2005) SynDECA: a tool to generate synthetic datasets for evaluation of clustering algorithms. In: 11th international conference on management of data
41. Wu X, Kumar V, Ross Q, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu PS, Zhou Z-H, Steinbach M, Hand DJ, Steinberg D (2008) Top 10 algorithms in data mining. Knowl Inform Syst 14(1):1–37
42. Zelnik-Manor L, Perona P (2004) Self-tuning spectral clustering. In: Proceedings of 18th annual conference on neural information processing systems
43. Zeng H-J, He Q-C, Chen Z, Ma W-Y, Ma J (2004) Learning to cluster web search results. In: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, pp 210–217
44. Zhang T, Ramakrishnan R, Livny M (1997) BIRCH: a new data clustering algorithm and its applications. Data Min Knowl Discov 1(2):141–182
45. Zhao Y, Karypis G (2005) Hierarchical clustering algorithms for document datasets. Data Min Knowl Discov 10(2):141–168

Author Biographies

Vineet Chaoji is a PhD candidate in the Computer Science Department at Rensselaer Polytechnic Institute. He received a Masters degree from Rochester Institute of Technology and a Bachelors in Engineering from Pune University (India). His areas of interesting include data mining, applied machine learning, social networking analysis and scalable mining algorithms.

Mohammad Al Hasan is a PhD candidate, Computer Science at Rensselaer Polytechnic Institute, Troy, USA. He received his MS from the University of Minnesota, Minneapolis, USA and a BSc(Engg) from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh both in Computer Science. His broad research interests are in data mining, information retrieval, machine learning, social network analysis, and bioinformatics.

Saeed Salem is a PhD candidate in the Department of Computer Science at Rensselaer Polytechnic Institute. He obtained his MSc in Information Technology from the same institute in 2002. His research interests are in data mining, machine learning, and bioinformatics.

Mohammed J. Zaki is a Professor of Computer Science at RPI. He received his PhD degree in computer science from the University of Rochester in 1998. His research interests focus on developing novel data mining techniques, especially in bioinformatics. He has published over 175 papers on data mining and co-edited 15 books and proceedings (including “Data Mining in Bioinformatics, Springer-London, 2005). He is currently an associate editor for ACM Transactions on Knowledge Discovery in Data, Data Mining and Knowledge Discovery, and Statistical Analysis and Data Mining. He was the program co-chair for SDM’08 and SIGKDD’09. He received the National Science Foundation CAREER Award in 2001 and the Department of Energy Early Career Principal Investigator Award in 2002. He also received the ACM Recognition of Service Award in 2003, and an IEEE Certificate of Appreciation in 2005.