ABACUS: Mining Arbitrary Shaped Clusters from Large Datasets based on Backbone Identification

Vineet Chaoji* Geng Li† Hilmi Yildirim‡ Mohammed J. Zaki§

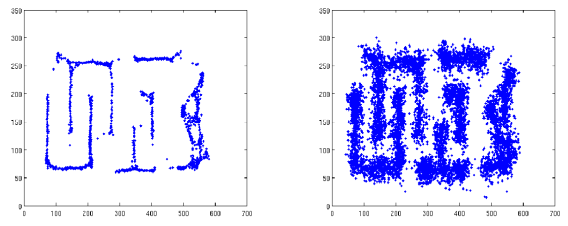
Abstract

A wide variety of clustering algorithms exist that cater to applications based on certain special characteristics of the data. Our focus is on methods that capture arbitrary shaped clusters in data, the so called *spatial clustering algorithms*. With the growing size of spatial datasets from diverse sources, the need for scalable algorithms is paramount. We propose a shape-based clustering algorithm, ABACUS, that scales to large datasets. ABACUS is based on the idea of identifying the *intrinsic structure* for each cluster, which we also refer to as the *backbone* of that cluster. The backbone comprises of a much smaller set of points, thus giving this method the desired ability to scale to larger datasets. ABACUS operates in two stages. In the first stage, we identify the backbone of each cluster via an iterative process made up of globbing (or point merging) and point movement operations. The backbone enables easy identification of the true clusters in a subsequent stage. Experiments on a range of real (images from geospatial satellites, etc.) and synthetic datasets demonstrate the efficiency and effectiveness of our approach. In particular, ABACUS is over an order of magnitude faster than existing shape-based clustering methods, yet it provides a comparable or better clustering quality.

1 Introduction

Clustering has been a prominent area of research within the data mining, machine learning and statistical learning communities. The choice of a clustering algorithm is strongly motivated by the data or domain characteristics, such as the data type (binary, categorical or numerical features). For instance, if the clusters are expected to span lower-dimensional subspaces, then projective or subspace clustering algorithms would give the better results. Similarly, when the clusters are non-convex, shape-based (or spatial clustering) methods that identify clusters with arbitrary shapes, sizes and densities are called for.

Spatial clustering has been applied to data from astronomy, meteorology, epidemiology, seismology, geospatial imagery, biomedicine, location-based services, and so on. The large size of many of the spatial datasets still poses scalability issues for existing shape-based clustering methods. Furthermore, these methods also vary in terms of robustness to noise in the dataset. We propose a simple, yet effective, robust, and scalable spatial clustering algorithm. Our approach is based on the hypothesis that a spatial cluster can be generated from a set of core points within the cluster that form the *backbone* or the *intrinsic shape* of the cluster. To elaborate, consider the intrinsic shape of a set of clusters, as shown in Figure 1(a). Given these core points in the backbone, a dataset can be obtained through the following hypothetical generative process. Assume that each backbone point has two parameters associated with it. The *weight* parameter $w_i \geq 1$, for a backbone point p_i, indicates the number of points that can be generated from p_i. The second parameter, *spread*, indicates the region around p_i within which w_i points can be generated. The spread parameter can be expressed in terms of a covariance matrix Σ_i, for a d-dimensional input space. For the sake of simplicity, we assume that the covariance matrix is a diagonal matrix with the variance σ_i along each dimension. Now, assume that a Gaussian process generates $m_i < w_i$ points at random, with mean at p_i and the covariance matrix Σ_i dictating the distribution of these points. The weight w_i of the backbone point is redistributed amongst the generated m_i points, either uniformly or as a function of the distance of the point from p_i. The covariance matrix for each of the m_i points is obtained by updating Σ_i (of p_i) such that the variance σ_i for each of the m_i points is decreased in proportion to weights assigned to them. The entire collection of m_i points resulting from each backbone point forms a new generation of points for the next step. This generative pro-



(a) Cluster Backbone (b) Recovered Clusters

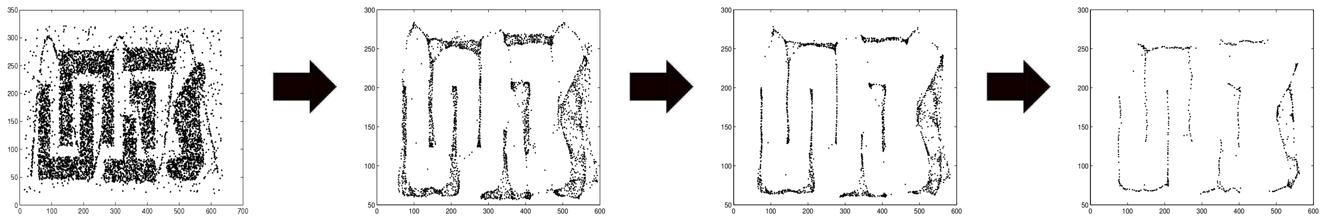
Figure 1: Clusters obtained through the generative process.

*Yahoo! Labs, Bangalore, India.

†Computer Science Dept, Rensselaer Polytechnic Institute, Troy, NY, USA.

‡Computer Science Dept, Rensselaer Polytechnic Institute, Troy, NY, USA.

§Computer Science Dept, Rensselaer Polytechnic Institute, Troy, NY, USA.



(a) Initial Dataset (b) After 3 Iterations (c) After 6 Iterations (d) After 8 Iterations

Figure 2: Initial dataset (a) after 3 (b) and 6 (c) iterations, and the final *backbone* after 8 iterations of ABACUS. The dataset (DS1) initially has 8000 points, whereas the backbone only has 838 points.

cess is repeated until the weight assigned to each new point has reduced to one. Figure 1(b) shows the clusters obtained from the backbone, via this generative process. For the sake of comparison, the original dataset from which the backbone was derived is shown in Figure 2(a). Note that the number of points generated is solely controlled by the w_i parameter assigned to each core point. In the real-world, we obviously do not have access to the cluster backbones, but rather we have to find the intrinsic shapes given the original datasets. Our new approach to scalable shape-based clustering, called ABACUS (anagram of the bold letters in **A**rbitrary **S**hape **C**l**U**stering via **BA**ckbones), is motivated by this generative approach. Given a spatial dataset, ABACUS aims to recover the intrinsic shape or backbone of the clusters, by intuitively following the generative process in reverse. The basic idea is to recursively collapse a set of points into a single representative point via an operation we call *globbing*. Furthermore, points also move under the influence of neighboring points. Over a few iterations, the dataset is repeatedly summarized until the backbone emerges. This process is briefly illustrated in Figure 2, which shows the putative intrinsic shape after three, six, and eight iterations, starting from the initial dataset. Once the backbone is identified, it is relatively straightforward to determine the final set of clusters. This is especially true since the iterative globbing and movement operations automatically confer two main benefits, namely i) removal of noise points from the dataset, and ii) reduction in the size of the dataset. Both these effects help in reducing the computational cost and memory requirements, as well as in making the method robust to noise, resulting in improved quality of the clustering. In the experimental section, we present extensive results on both synthetic and real datasets, which show how scalable and effective our method is compared to state of the art methods; ABACUS has comparable quality of clustering, but it can outperform existing methods in terms of runtime by an order of magnitude or more.

2 Related Work

Traditionally, spatial clustering methods have been divided into the following categories – *partitional*, *hierarchical*, *density-based* and *spectral*.

CLARANS [10] was one of the pioneering works in spatial clustering, but it is rather slow since it is a medoid based partitioning approach. The SNN (shared nearest-neighbor) algorithm [5] is an example of partitional clustering. SSN computes a graph based on the shared nearest neighbors between every two points. The connected components of the graph are the final clusters after some threshold-based edge removal.

The density based approach is exemplified by DBSCAN [6] and DENCLUE [8], both of which can find arbitrary shaped clusterings. However, they can be quite sensitive to the parameter values, and are computationally expensive ($O(N^2)$ for high dimensional data, otherwise $O(N \log N)$ with R*-tree index structure). DENCLUE's density estimation identifies local maxima (termed *density attractors*) in the data. Although the notion of density attractors is similar to the points on the backbone in ABACUS they do not necessarily preserve the structural shape of the clusters. Another non-parametric algorithm – *mean shift clustering* [4] – is closely related to DENCLUE.

CURE [7] is a hierarchical agglomerative clustering algorithm that handles shape-based clusters, but it is still expensive with its quadratic complexity, and more importantly, the quality of clustering depends enormously on the sampling quality. In general, *sampling based methods* suffer when the clusters are of varying sizes and densities [1]. Another hierarchical approach, CHAMELEON [9], formulates the shape-based clustering as a graph partitioning algorithm. However, it requires a number of thresholds to be set for effective clustering.

The spectral clustering approach [12] is also capable of handling arbitrary shaped clusters. They formulate the arbitrary shape clustering problem as a *normalized min-cut* problem. Unfortunately the spectral approach is not really scalable, requiring $O(N^3)$ time. Recently, a scalable approximate spectral method was introduced [14], which first selects representative points on which the full spectral clustering is performed.

Many efforts have focused on scaling spatial clustering. For scaling hierarchical clustering further, Breunig et al. [2] propose compressing the data using representatives called *Data Bubbles*. Clustering is then performed on the compressed representation. The approach proposed in [13] shrinks clusters into dense compact regions by altering the position of data points under "gravitational force" exerted by other neighboring points. Unlike [13], ABACUS repeatedly globs points resulting in faster convergence of the algorithm. SPARCL [3] is one of the latest approaches for shape-based clusters. It works in two phases. In the first phase a large number of representatives or *pseudo-centers*, and in the second step it merges them to obtain clusters. It is linear in the number of points in the dataset, but is sensitive to the quality of pseudo-centers.

Most algorithms described above are unable to scale to large datasets. Due to lack of space we are unable to describe some of the other algorithms.

3 The ABACUS Approach

Our approach to shape-based clustering is motivated by the notion that a cluster possesses an *intrinsic shape* or a *core shape*. Intuitively, for a 2-dimensional Gaussian cluster, points around the mean of the cluster could be considered as points forming the core shape of the cluster. For an arbitrary shaped cluster, such as shown in Figure 2(a), the intrinsic shape of the cluster is captured by the *backbone* of the cluster (Figure 2(d)). ABACUS needs two parameters – the number of nearest neighbors, denoted k, to be considered for each point, and the final number of clusters desired, denoted C. Unlike some of the methods that need an absolute radius as a parameter, the number of nearest neighbors k is independent of the density of a cluster. This parameter thus makes ABACUS relatively robust to clusters with varying densities.

The ABACUS clustering approach has two phases, detailed below. In the first phase we identify the intrinsic shape of the clusters. In the following phase, the individual clusters are identified.

3.1 Preliminaries Consider a dataset $\mathcal{D}$ of N points in d-dimensional Euclidean space. The distance between points i and j is represented by d_{ij}. The k-nearest neighbors (kNN) of a data point i are given by the set $R_k(i)$. The nearest neighbors for all points are captured in a matrix A, where each entry $A(i,j)$ is given as

$$A(i,j) = \begin{cases} 1 & \text{if } j \in R_k(i) \\ 0 & \text{if } j \notin R_k(i) \end{cases}$$

The term *kNN matrix* is used for A henceforth.

Figure 3 (left) shows a sample dataset and Figure 4(a) shows its corresponding kNN matrix. Figure 3 (right) shows the sample dataset after one iteration, while Figure 4(b) shows the corresponding updated kNN matrix. In this example, k is set to 2.

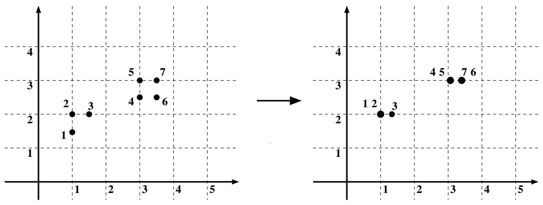


Figure 3: Globbing and Movement ($k = 2$)

$$A_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

(a) Initial kNN Matrix

$$A_1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

(b) Updated kNN Matrix

Figure 4: kNN matrices for sample dataset (k=2)

3.2 Backbone Identification As opposed to the generative model described earlier, our goal is to identify points belonging to the backbone given the original dataset. In essence, ABACUS follows the generative model in the reverse order, starting with the original dataset and culminating in the identification of the backbone, as illustrated in Figure 2(d). The backbone identification phase consists of two simple operations:

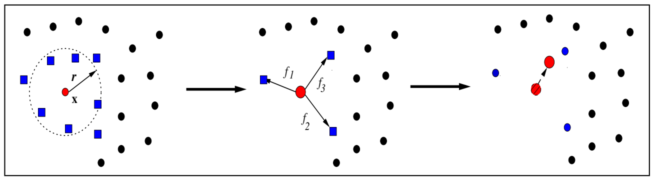


Figure 5: Globbing and object movement: x (red) has 8 nearest neighbors (blue); only 5 lie within radius r. These 5 are globbed by x, and the 3 other points move x based on the force vectors f_i.

Globbing: Globbing involves assigning a representative to a group of points. All points that lie within a d-dimensional ball of radius r, around a representative x are globbed by x. The globbed points are removed from the dataset and their representative (point) is retained. Each point in the dataset has a *weight* w assigned to it. Initially, the weight of each point is set to 1. As points are globbed by a representative, the weight of the representative is updated to reflect the number of points globbed by it. Note that a representative can be globbed by another representative point. As discussed later, the value of the radius r is directly estimated from the dataset. Note that r represents the *spread* parameter in the generative model.

Object Movement: In our model, each point experiences a force of attraction from its neighboring points. Under the influence of these forces a point can change its position. The magnitude of the movement is proportional to the forces exerted on the point and the direction of movement is the weighted sum of the force vectors. In the context of the generative model, a point moves towards the most likely component that is responsible for generating the point.

The backbone identification phase involves repeated application of the above two steps, as illustrated in Figure 5. In the first step, objects are globbed starting at the dense regions of the dataset, since that results in faster convergence of the algorithm. Moreover, starting at the dense regions ensures that noise points do not distort the intrinsic shape of the clusters by globbing points from true clusters. The dense regions are identified by smaller values for the kNN distance. In the following second step, the representative points move under the influence of forces exerted by neighboring points. Figure 3 shows a sample dataset consisting of 7 points and the effect of one iteration (globbing followed by movement) on the dataset. Similarly, Figures 4(a) and 4(b) show the initial kNN matrix A_0 and the updated kNN matrix A_1 after one iteration, respectively. On convergence of the iterative process, A_n represents the intrinsic shape of the clusters. Figure 2(d) shows the backbone of the dataset in Figure 2(a), on convergence. Note that the two steps outlined in the algorithm are essentially simulating the generative model in reverse. The ABACUS algorithm is outlined in Figure 6. It

```
ABACUS($\mathcal{D}_0, k, C$):
1. Initialize $w_i = 1, \forall i \in \mathcal{D}_0$
2. $j = 0$
3. $\mathcal{K}$ = compute_kNN($\mathcal{D}_0$)
4. $r$ = estimate_knn_radius($\mathcal{D}_0, k, \mathcal{K}$)

5. repeat
6.     $j = j + 1$
7.     glob_objects($\mathcal{D}_{j-1}, r, k$)
8.     $\mathcal{D}_j$ = move_objects($\mathcal{D}_{j-1}, r, k$)
9.     $m_j$ = number of points moved in iteration j
10.    $\mathcal{K}$ = update_kNN($\mathcal{D}_j$)
11. until $\frac{m_j}{m_{j-1}} < \frac{m_{j-1}}{m_{j-2}}$

12. $\mathcal{C}$ = identify_clusters($\mathcal{D}_{j-1}, C$)
```

Figure 6: The ABACUS Clustering Algorithm

takes three inputs – the dataset $\mathcal{D}$ of d-dimensional points, the number of nearest neighbors k, and the number of final clusters C. **estimate_knn_radius** computes an estimate for the trimmed average distance to the k^{th} nearest neighbor for objects in the dataset. The radius is estimated by first obtaining the distance to the k^{th} nearest neighbor over a random sample from the dataset. The average of the top 95% percentile of these distances (arranged in ascending order) is used as the globbing radius r. Note that we discard the top 5% distances to make r robust to outliers.

During **glob_objects** all points within a radius r of a point x are marked as being "globbed" or represented by x. Note that not all points within the kNN of x are within globbing radius r, thus the use of r in the globbing step ensures that only points in the close proximity of x can be represented by x. Such selective globbing also ensures that outlier or noise points do not glob points belonging to dense cluster regions. Globbing modifies the dataset by removing the globbed points and by updating the weight w_x of the representative point to include the weights of all the globbed points (i.e., $w_x = \sum_{\forall p \ s.t. \ dist(p,x)<r} w_p$). An estimate for r based on sampling is preferred for the following reasons. First, an arbitrarily small value for r can degrade the convergence of the backbone identification approach. On the other hand, an arbitrarily large value for r can result in points from more than one cluster being globbed by a representative point.

In the **move_objects** step, a point y in d-dimensional space is displaced under the influence of its nearest neighbors' force of attraction. Out of the k nearest neighbors, only those that have not been globbed by y participate in displacing y. The force exerted by an object z on object y is proportional to w_z and inversely proportional to $dist(y, z)$, where $dist()$ is some distance function. The updated position of y in dimension i is given by Equation 3.1, where y_i is the i^{th} dimension of y.

(3.1)
$$y_i^{new} = \frac{y_i \cdot w_y + \sum_{z \in R_k(y) \wedge d(y,z)>r} z_i \cdot w_z \cdot \frac{1}{dist(y,z)}}{w_y + \sum_{z \in R_k(y) \wedge d(y,z)>r} w_z \cdot \frac{1}{dist(y,z)}}$$

Figure 5 elaborates the globbing and movement steps. The dataset $(D)_{j-1}$ before violation of the stopping condition is used for extracting the final clusters.

3.3 Stopping Condition for ABACUS One can extrapolate that the above two steps, of globbing and object movement, repeated without a suitable stopping condition would result in a dataset with a single point which globs all the points in the dataset. Let $\mathcal{D}_i$ be the dataset after iteration i. Let $\mathcal{D} = \mathcal{D}_0$ be the initial dataset, and let $\mathcal{D}_{final}$ be the final globbed dataset obtained after Line 11 of Figure 6. Clustering quality is poor if $\mathcal{D}_{final}$ has points that represent globbed points from more than one natural cluster. At the same time, if $\mathcal{D}_{final}$ has most of the points in $\mathcal{D}_0$, then the algorithm has not achieved a substantial reduction in the dataset size. Hence, a "good" stopping condition needs to balance the reduction in the dataset size and the degree to which $\mathcal{D}_i$ captures the shape-based clusters of $\mathcal{D}_0$. To express the similarity between $\mathcal{D}_i$ and $\mathcal{D}_0$ we compare their corresponding kNN matrices. Since the sizes of the two kNN matrices are not the same, an estimated kNN matrix for the initial set of points is reconstructed from the kNN matrix for $\mathcal{D}_i$.

To formalize this notion, let A_i be the kNN matrix after iteration i. The initial kNN matrix for the dataset is A (or A_0) as shown in Figure 4(a). Let the size of A_i be $N_i \times N_i$, where N_i is the number of points in the dataset at the end of iteration i, and $N_0 = N$ is the number of points in the initial database. Consider an onto function $f_i : \mathbb{R}^d \rightarrow \mathbb{R}^d$ for iteration i. Function f_i maps a point a in the original dataset $\mathcal{D}_0$ to a point in $\mathcal{D}_i$ that has globbed a.

We would like to compute the probability that a point b is in the kNN set of another point b, after iteration i. There are two cases to consider: 1) Both a and b are globbed by the same representative x, and 2) a and b are globbed by different representatives, x and y respectively. Consider the first case: Given that $f_i(a) = f_i(b) = x$, i.e., both a and b are globbed by the same point $x \in \mathcal{D}_i$, the probability that b is a kNN of a can be approximated by:

$$\mathbf{Pr}[b \in R_k(a)] \propto \frac{\binom{w_x-2}{k-2}}{\binom{w_x-1}{k-1}} \tag{3.2}$$

where w_x, the weight of x, is the number of points globbed by x. The numerator in the above equation corresponds to the number of ways of choosing remaining $k-2$ points from $w_x - 2$, since we assume that a and b have already been chosen. The denominator corresponds to the number of sets (of points) that include point a.

In the alternate scenario, when a and b are globbed by different representatives, namely $x = f_i(a) \neq f_i(b) = y$, the probability of $b \in R_k(a)$ is given by the expression

$$\mathbf{Pr}[b \in R_k(a)] \propto \frac{1}{d(x,y)} \cdot \frac{\binom{w_x+w_y-2}{k-2}}{\binom{w_x+w_y-1}{k-1}} \tag{3.3}$$

Here, the numerator gives the number of ways of choosing $k-2$ points from the glob set of x and y, i.e., the number of possible neighborhood containing both a and b. The denominator gives the number of ways of choosing the neighborhood for a. The probability in Equation 3.3 depends on two factors: 1) the number of points globbed by the representatives of a and b in $\mathcal{D}_n$, and 2) the distance between the representatives x and y. The larger this distance, the smaller the probability of b belonging to $R_k(a)$. Similarly, the probability in Equation 3.3 is less than that in Equation 3.2. This resonates with the intuition that nearby points should have higher probability. Note that although the kNN relation is not symmetric, the above probabilities are symmetric, i.e., $\mathbf{Pr}[b \in R_k(a)] = \mathbf{Pr}[a \in R_k(b)]$. Note also that for Equations 3.2 and 3.3 to represent true probabilities, the right hand side should be normalized by dividing by the term $\mathcal{Z}_a = \sum_b \mathbf{Pr}[b \in R_k(a)]$.

Let M_i denote the $N \times N$ matrix with the entry $M_i[x,y]$ representing $\mathbf{Pr}[y \in R_k(x)]$, i.e., M_i is the reconstructed matrix for the probabilistic kNN relationship from $\mathcal{D}_i$.

3.3.1 MDL Based Stopping Condition

Given the above description, the stopping condition for ABACUS, can be ideally formulated in terms of the Minimum Description Length (MDL) principle [11], that takes an information theoretic approach towards selecting a model.

The MDL principle suggests selecting the model h_i that minimizes $L(h_i) + L(\mathcal{D} \mid h_i)$, where $L(h_i)$ is the number of bits required to represent the model and $L(\mathcal{D} \mid h_i)$ is the number of bits to encode the data given the model. Thus, the MDL principle balances the generality and the specificity in model selection for the data. A simple model requires fewer number of bits corresponding to the $L(h_i)$ term, but it results in a larger number of bits to represent the data $L(\mathcal{D} \mid h_i)$. On the contrary, a complex model would exhibit just the opposite effect.

In the context of ABACUS, the set of hypotheses/models is represented by $\mathcal{D}_i$ ($\forall i > 0$), i.e., the set of globbed points after each iteration. The simplest model $\mathcal{D}_1$ requires the largest number of bits, but requires fewest number of bits to encode $\mathcal{D}_0$. Stated another way, the simplest model has the smallest error when it comes to *reconstruction* of the original data. This is often called as the *reconstruction error.* For subsequent hypotheses, as $L(\mathcal{D}_i)$ decreases, the additional information required to represent $\mathcal{D}_0$ (given by $L(\mathcal{D}_0 \mid \mathcal{D}_i), i > 0$) increases. $L(\mathcal{D}_0 \mid \mathcal{D}_i)$ can be interpreted as the error introduced in reconstructing $\mathcal{D}_0$ from $\mathcal{D}_i$.

As seen before, A_i represents the kNN matrix for $\mathcal{D}_i$ and M_i represent the kNN matrix "reconstructed" from $\mathcal{D}_i$ using Equations 3.2 and 3.3. The probability that the reconstructed kNN matrix M_i faithfully captures A_0 is given by $\mathbf{Pr}(A_0 \mid M_i)$. Since each element in A_0 can be considered to be independent, $\mathbf{Pr}(A_0 \mid M_i)$ can be expressed as

$$\mathbf{Pr}(A_0 \mid M_i) = \prod_{m=1}^{N_0} \prod_{n=1}^{N_0} \mathbf{Pr}\left(A_0(m,n) \mid M_i(m,n)\right) \tag{3.4}$$

Since A_0 is a binary matrix, we have the expression

$$\mathbf{Pr}\left(A_0(m,n) \mid M_i(m,n)\right) = \begin{cases} M_i(m,n) & \text{if } A_0(m,n) = 1 \\ 1-M_i(m,n) & \text{if } A_0(m,n) = 0 \end{cases} \tag{3.5}$$

A high value for $M_i(m,n)$ when $A_0(m,n) = 1$ indicates that $M_i(m,n)$ can successfully represent the neighborhood relationship between m and n. If $M_i(m,n)$ is small when $A_0(m,n) = 0$, then $1 - M_i(m,n)$ gets a high value, thus capturing the absence of neighborhood relationship between m and n. . The term $L(\mathcal{D}_0 \mid \mathcal{D}_i)$ normalized by ${N_0}^2$, gives the average number of bits per entry in the matrix. The number of bits required to represent the total reconstruction error is captured by

$$\begin{aligned} L(\mathcal{D}_0 \mid \mathcal{D}_i) &= -\log \mathbf{Pr}(A_0 \mid M_i) \\ &= -\sum_{m=1}^{N_0} \sum_{n=1}^{N_0} \log \mathbf{Pr}\left(A_0(m,n) \mid M_i(m,n)\right) \end{aligned} \tag{3.6}$$

The number of bits to represent the model depends on the relative size of $\mathcal{D}_i$, given by the expression

$$L(\mathcal{D}_i) = -\log\left(\frac{\mid \mathcal{D}_i \mid}{\mid \mathcal{D}_0 \mid}\right) \quad (3.7)$$

Hence the trade-off at the end of any iteration i is between the average reconstruction error given by $\frac{1}{N_0^2}L(\mathcal{D}_0 \mid \mathcal{D}_i)$ and the size of the model $L(\mathcal{D}_i)$. Generating the reconstruction matrix entails computing the error at each entry $M_i(m,n)$. The computation cost for each iteration is thus $O(N^2)$, where N is the number of points in the original dataset. This approach is infeasible for large datasets, both in terms of computation and in terms of the memory requirement. To bypass this computational cost we present a simpler alternative that tries to capture the same trade-off between the reconstruction error and the dataset size.

3.3.2 Practical Stopping Condition Given that the pure MDL-based stopping condition discussed above is computationally expensive, we use a more practical stopping condition for ABACUS, that nevertheless, is intuitively related to the MDL based formulation.

Notice that if points are only globbed (without object movement), it results in the sparsification (reduction) of the data. To complement the globbing, moving the points enables further globbing in subsequent iterations. If g_i is the number of points globbed in an iteration and m_i is the number of points that are moved in an iteration then we have $g_i \propto m_{i-1}$. That is, the number of points globbed in iteration i, is directly proportional to the number of points that move in the previous iteration $i-1$. This observation is shown in Figure 7.

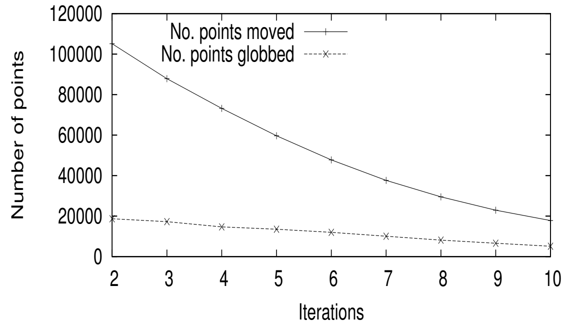


Figure 7: The number of points moved and globbed per iteration for a dataset with 1000K points.

Intuitively, as more points are globbed across the iterations, the reconstruction error obviously increases. Let $E_i = L(D_0|D_i)$ be the reconstruction error at the end of iteration i and let the error difference between two consecutive iterations be $\Delta E_i = E_i - E_{i-1}$. The difference between the errors is proportional to the number of points globbed, i.e., $\Delta E_i \propto \frac{g_i}{g_{i-1}}$. Combining this with the previous observation ($g_i \propto m_{i-1}$) yields $\Delta E_i \propto \frac{m_{i-1}}{m_{i-2}}$. As fewer points move in subsequent iterations ($m_i < m_{i-1}$), it reflects the decline in the size of the dataset, i.e., $N_i < N_{i-1}$. The ratio $\frac{m_i}{m_{i-1}}(< 1)$ captures the relative rate of this decline.

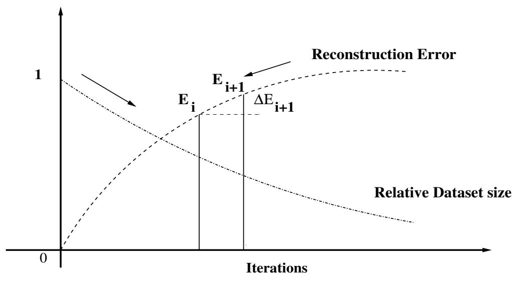


Figure 8: Balancing the two contradicting influences in the clustering formulation.

Our stopping condition is based on these observations. If the expression

$$\frac{m_{i-1}}{m_{i-2}} < \frac{m_i}{m_{i-1}} \quad (3.8)$$

is false then the iterative process is halted, else continued (see line 11 in Figure 6). In other words, if the rate of decline in the current iteration i is less than that in iteration $i-1$, we stop the globbing and movement operations.

Let us look at Figure 8 to understand this stopping condition. Figure 8 shows the two contradicting influences – dataset size and reconstruction error. The condition in Equation 3.8 favors an increase in the ratio $\frac{m_i}{m_{i-1}}$, implying that the stopping condition encourages a rapid decrease in the size of the dataset, by the relation $m_i \propto N_i$. The downward sloping arrow along the 'Relative Dataset Size' curve in Figure 8 represents this effect. In short, the stopping condition ensures that the algorithm progresses as long as the dataset size continues to shrink progressively.

$\frac{\Delta E_{i+1}}{\Delta E_i} < 1$, if expression 3.8 is not true. As long as the stopping condition in Equation 3.8 holds, the rate of change of relative error difference ($\frac{\Delta E_{i+1}}{\Delta E_i}$) is positive, i.e. numerical value of the fraction $\frac{\Delta E_{i+1}}{\Delta E_i}$ is increasing. Hence, the condition in Equation 3.8, does not favor a decline in the relative error difference, which happens during later iterations. In the context of Figure 8, this tendency to oppose a decline in the relative error difference is depicted by the downward sloping arrow along the 'Reconstruction Error' curve. At the iteration at which the stopping condition in Equation 3.8 is violated, both the above effects (increasing relative reconstruction error and the rate at which the dataset size is decreasing) are balanced. We chose to stop at this iteration. This is indicated by the intersection point of the two curves in Figure 8.

At the end of the iterative process a much smaller dataset $\mathcal{D}_{final}$, as compared to the original dataset $\mathcal{D}_0 = \mathcal{D}$, is obtained.

3.4 Cluster Identification Once the intrinsic shape or backbone of the clusters is identified, the task remains to isolate the individual clusters. ABACUS currently assumes that the desired number of clusters C has been pre-specified. Below, we also discuss how one can determine the number of clusters automatically.

3.4.1 Number of clusters specified Given that the first phase helps drastically reduce the noise and the size of the dataset, the cluster identification step is relatively straightforward if the desired number of clusters C is given. Due to the size reduction any suitable clustering algorithm can be applied to $\mathcal{D}_{final}$ in line 12 in Figure 6. In our experiments, we applied both DBSCAN [6] and CHAMELEON [9] during the cluster identification phase of ABACUS. Both these algorithms are able to effectively capture the clusters and are relatively robust to noise, though the latter is more efficient.

3.4.2 Number of clusters unspecified When the desired number of clusters C is not specified, we propose the following two-step approach to identify the final set of clusters. In the first step, one can run a connected components algorithm on $\mathcal{D}_{final}$ to obtain a set of preliminary clusters $\mathcal{C}$. In the second step, the clusters in $\mathcal{C}$ can be merged to obtain the final clusters.

The cluster merging process is based on two similarity measures. Let $B(C_i, C_j)$ be the points in cluster C_i that have a point from C_j in their kNN set, i.e., $B(C_i, C_j) = \{p_i \in C_i : \exists p_j \in R_k(p_i) \wedge p_j \in C_j\}$. We call $B(C_i, C_j)$ the *border points* in cluster C_i with respect to cluster C_j. Note that $B(C_i, C_j)$ need not be the same as $B(C_j, C_i)$. Let $E(C_i, C_j)$ be the total number of occurrences of points in C_j in the k-neighborhood of points in C_i, i.e., $E(C_i, C_j) = \sum_{p_i \in C_i} \left|\{p_j : p_j \in R_k(p_i) \wedge p_j \in C_j\}\right|$. Note that $E(C_i, C_j)$ may count a point multiple times if it belongs to the neighborhood of multiple points p_i. Let $B(C_i)$ be the set of all border points in cluster C_i, i.e., $B(C_i) = \bigcup_{\forall C_j \neq C_i} B(C_i, C_j)$.

The first similarity metric S_1 is given as

$$S_1(C_i, C_j) = \frac{E(C_i, C_j)}{\mid B(C_i, C_j) \mid} > \alpha \tag{3.9}$$

The higher the value of the ratio in Equation 3.9, the greater the similarity between the clusters. A high value for $S_1(C_i, C_j)$ indicates that the points in C_j are close to the border points in C_i. This similarity metric captures the degree of closeness, measured in terms of local neighborhood of border points, between a cluster pair.

The second similarity measure, S_2, is given as

$$S_2(C_i, C_j) = \frac{\mid B(C_i, C_j) \mid}{\mid B(C_i) \mid} > \beta \tag{3.10}$$

S_2 ensures that two clusters can be merged only if the interaction "face" (fraction of border points) between the two clusters is above the β threshold.

Cluster pairs are iteratively merged, starting with the pair with highest similarity. For two clusters C_i and C_j to be merged both the conditions, given by Equations 3.9 and 3.10, must be satisfied. Since the true number of clusters are not specified, we need to provide lower-bound thresholds (α and β) for the similarity criteria to continue merging of clusters.

3.5 Complexity Analysis Let us assume that ABACUS converges after t iterations. The number of points at the end of each iteration is given by N_0, N_1, ..., N_t. Initially ABACUS performs a kNN computation on all the points in the dataset – $O(N^2)$ in the worst case for high-dimensional data, whereas $O(N \log N)$ for spatial datasets that are typically in just 2 or 3 dimensions.

Let us now consider the time for the subsequent iterations. For each point p (in each iteration i) that globs its nearest neighbors, we have to update the kNN only for all the affected points (line 10 in Figure 6). Since we use a kd-Tree to store the points, a kNN search takes $O(N_i^{1-\frac{1}{d}})$ time. Let $g_1, g_2, ..., g_t$ be the number of points that have globbed other points, in each of the iteration. The total complexity of the kNN searches is given by $O(\sum_{i=1}^{t} g_i \cdot N_i^{1-\frac{1}{d}})$. Moving the points involves computing the new location based on the kNN. If $m_1, m_2, ..., m_t$ represents the number of points that move in each iteration, the total cost of moving across all iterations is given by $O(k \cdot \sum_{i=1}^{t} m_i)$. When CLUTO [1] is applied in Phase 2 to the set of points after the iterative process, the computational cost is $O(N_t \log N_t)$. Hence the total computational cost is the sum of the above terms. Let us assume that a constant fraction of points are globbed and moved in each iteration, i.e., $g_i = m_i = O(1)$ (note that this is worst case behavior for us, since the more the points are globbed, the faster is the convergence). Also let us assume that in the worst case number of points in each iteration is $N_i = O(N)$. Then, in the worst case, the runtime complexity over all the iterations is $O(tN + kt + N \log N)$, where t is the number of iterations and k is the number of nearest neighbors.

In summary, the complexity of ABACUS is $O(N^2)$ when the number of dimensions is large, since in that case the time for computing the initial kNN for each point would dominate. However, for the typical 2D or 3D datasets common in spatial clustering, ABACUS takes $O(tN + N \log N) = O(N \log N)$ time. Practically, ABACUS is much faster as compared to other algorithms as shown in the following section. This is because the number of kNN searches drops significantly in each iteration.

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

Name	$\|D\|(d)$	C	k	ABACUS CHAMELEON / DBSCAN	SPARCL (Random)	SPARCL (LOF)	CHAMELEON	KASP
DS1	8000 (2)	6	70	1.7s / 2.6s	1.8s	4.1s	4.3s	19s
DS2	8000 (2)	6	70	1.3s / 2.2s	1.5s	4.0s	4.2s	13s
DS3	10000 (2)	9	55	1.9s / 3.0s	2.5s	5.5s	5.9s	33s
DS4	8000 (2)	8	20	1.7s / 3.4s	1.8s	4.2s	4.3s	24s
Swiss-roll	19386 (3)	4	70	4.4s / 5.7s	4.9s	19.6s	19.8s	43s

Table 1: Runtime Performance on Synthetic Datasets. All times are reported in seconds.

4 Experimental Evaluation

All our experiments are conducted on a Mac G5 machine with a 2.66 GHz processor, running the Mac 10.4 OS X. ABACUS is written in C++, using the Approximate Nearest Neighbor Library (ANN)[2]. We compare the performance of ABACUS with a range of clustering algorithms, namely CHAMELEON [9] as implemented in the CLUTO package, SPARCL [3], DBSCAN [6] implemented in C++ using R*-tree index, and finally, the K-Means based Fast Spectral Approximation (KASP) [14] obtained from its authors [3] (written in R). Parameters were tuned for each method for best results. Also note that unless mentioned otherwise, we used the standard clustering parameters *-clmethod=graph, -sim=dist, -agglofrom=30)* for CHAMELEON. For DBSCAN we used $minPts = 15$ and $eps = 0.7$. For the cluster identification phase of ABACUS we used these same parameters. Finally, for KASP we used $\gamma = 8, \sigma = 100$, and for SPARCL we use the parameters $K = 30, minPts = 15$.

4.1 Datasets A wide range of datasets were used to evaluate ABACUS. For the scalability experiments, we use the dataset DS-SCAL, from SPARCL [3], which consists of 13 arbitrary shaped clusters in 2D with varying densities and number of points (up to 1 million points). DS1 – DS4, shown in Figure 11 and 12, are datasets that have been used by previous methods like CURE, CHAMELEON and SPARCL. The real datasets consist of proteins of varying densities (PROT; see Figure 10(c)), natural images (NATIMG; see Figure 13), and geospatial satellite images (GEOIMG; see Figure 14).

4.2 Results on Synthetic Datasets Table 1 shows runtime performance of ABACUS and other algorithms on some popular datasets in the literature. The runtime for ABACUS with both CHAMELEON and DBSCAN in phase 2 is shown in Column 5. ABACUS is considerably more efficient as compared to KASP or CHAMELEON. For these relatively small datasets ABACUS is comparable to SPARCL(random), and has an advantage over SPARCL (LOF) and CHAMELEON in terms of the execution time. KASP is about 10 times slower. Figures 11, 12 and 10(a)-(b) show the backbone and final clusterings for these datasets. The figures also show the number of points in the original dataset, and those in the resulting backbone. The 3D datasets (including the 3D protein dataset in Figure 10 (c)-(d)) exhibit a predominant sparsification effect as compared to a skeletonization effect. The k parameter used by ABACUS is shown in Column 4 of Table 1.

4.3 Scalability Results To study the scalability of ABACUS, we used the DS-SCAL dataset, with varying number of points. The number of noise points in this dataset are set constant at 5% of the total dataset size. The dimensionality of the dataset is $d = 2$ and the number of clusters are fixed at 13. For each dataset k is set at 70. Further, for these results, we used CHAMELEON (with standard parameters) for the cluster identification phase.

A	B	C	D	E
10K	0.5s	4	0.4s	4.41%
50K	3.0s	4	1.1s	4.07%
100K	5.6s	4	1.6s	5.2%
200K	12.2s	4	7.7s	5.98%
400K	26.5s	4	25.1s	6.94%
600K	40.9s	4	58.7s	6.88%
800K	57.5s	4	109.9s	7.49%
1000K	113.9s	10	10.5s	1.78%

Table 2: ABACUS Scalability Results. The size of the dataset is varied keeping the noise at 5% of the dataset size ($d = 2, k = 70, C = 30$). A: Dataset size (no. of points), B: Time for *t* iterations, C: Number of iterations (*t*), D: Time for Phase 2, E: Dataset size after *t* iterations (% of initial size).

The first column in Table 2 specifies the size of the dataset, the largest being a dataset with 1 million points. The table breaks down the total execution time of ABACUS into the time taken by the backbone phase (Column 2) and the cluster identification phase (Column 4). The number of iterations t, and the size of the final dataset (as a percentage of the initial dataset) after t iterations are shown in Columns 3 and 5, respectively. We can observe that the time for backbone identification increases with increasing size of the

[2]http://www.cs.umd.edu/∼mount/ANN/

[3]http://www.cs.berkeley.edu/∼jordan/fasp.html

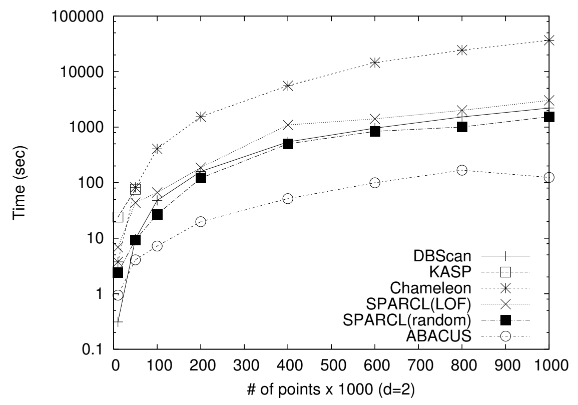
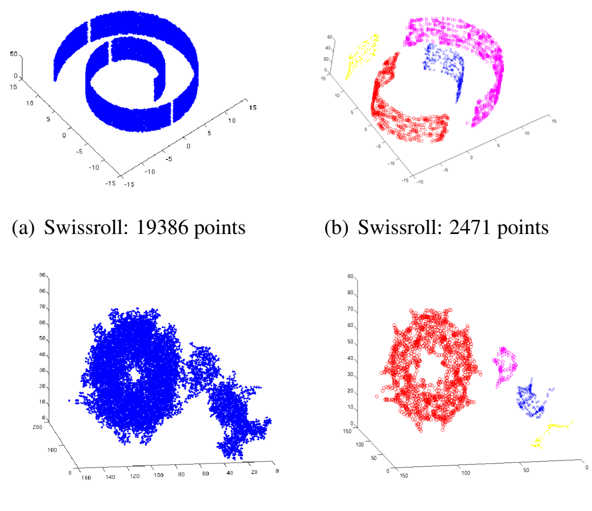


Figure 9: Scalability: ABACUS versus other algorithms. Note the log scale on the y-axis.

datasets. Also, different datasets exhibit varying degrees of dataset reduction. The time taken by the cluster identification phase is proportional to the dataset reduction achieved. This is evident from the observation that the time taken by Phase 2 on the 1000K dataset is ten times less than that for the 800K dataset. This reduction is purely a factor of the density of the points and also the relative position of the points. Figure 9 compares the execution time of ABACUS

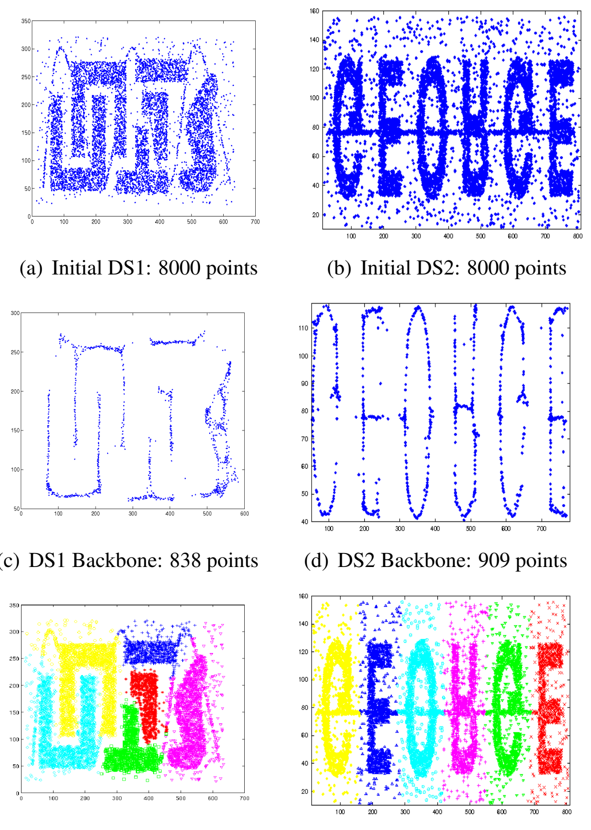


(a) Swissroll: 19386 points (b) Swissroll: 2471 points

(c) Proteins: 14669 points (d) Proteins: 2023 points

Figure 10: ABACUS on 3D datasets (Initial Data, and Backbone/Clusters)

with other competing algorithms. For ABACUS the time reported is the total execution time, i.e., time for the iterative backbone step, and the cluster finding step. We can clearly observe that as the dataset size increases, ABACUS get progressively better. ABACUS is about two orders of magnitude faster than CHAMELEON, and an order of magnitude faster than SPARCL(random/LOF; run with $K = 100$ and $minPts = 15$) and DBSCAN. KASP was too slow to be run on more than 100K points.



(a) Initial DS1: 8000 points (b) Initial DS2: 8000 points

(c) DS1 Backbone: 838 points (d) DS2 Backbone: 909 points

(e) DS1 Clusters (f) DS2 Clusters

Figure 11: ABACUS Results: Initial Dataset, Backbone, and Final Clustering on DS1 and DS2.

4.4 Results on Real Datasets We applied ABACUS to several image datasets, containing natural images (NATIMG), as well as satellite images (GEOIMG). For the natural image results shown in Figure 13, we first applied a preprocessing step, whereby the RGB (Red-Green-Blue) values for each pixel in the image are obtained. ABACUS is then run on the RGB 3D data. For each row in Figure 13, the original image is followed by the clustering results from ABACUS and K-Means. It is quite clear that ABACUS yields a better segmentation/clustering of these images. K-Means results in clusters that have granularities within them, whereas ABACUS yields more uniform clusters, i.e., it has a smoothening effect on the objects, resulting in objects having uniform color. For instance, the entire pyramid has the same color using ABACUS (Fig. 13(h)), in contrast to K-Means results (Fig. 13(i)), where it appears somewhat patchy. Results for CHAMELEON and KASP are omitted due to space considerations.

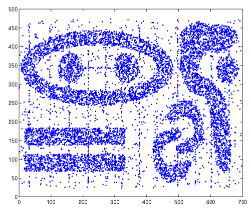
Table 3 shows comparative running time among the

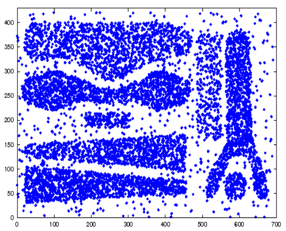
Name	$\lvert D\rvert(d)$	C	ABACUS (CHAMELEON)	K-Means	CHAMELEON	SPARCL (LOF)	KASP
Horse	154401 (3)	5	31.2s	4.5s	868.6s	41.8	1325s
Mushroom	154401 (3)	15	29.3s	18.6s	797.3s	-	1589s
Pyramid	154401 (3)	5	11.3s	2.1s	743.2s	-	1441s
Road	154401 (3)	4	14.9s	1.9s	779.4s	-	1369s

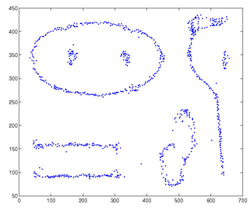
Table 3: Runtime Performance on NATIMG Datasets. K-Means implemented in Matlab.

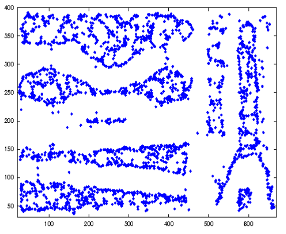
Name	$\lvert D\rvert(d)$	C	ABACUS (CHAMELEON)	CHAMELEON	SPARCL (LOF/Random)	KASP
GEOIMG1	37876 (2)	3	7.6s	42.7s	4.4s / 3.6s	103s
GEOIMG2	62417 (2)	10	35.1s	100.5s	21.5s / 5.7s	310s
GEOIMG3	143269 (2)	4	136.3s	519.3s	70.4s / 17.9s	-

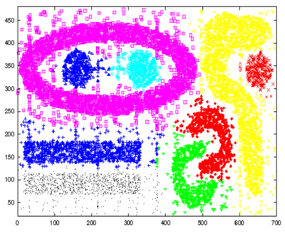
Table 4: Runtime Performance on GEOIMG Datasets.


(a) Initial DS3: 10000 points


(b) Initial DS4: 8000 points


(c) DS3 Backbone: 1077 points


(d) DS4 Backbone: 2211 points


(e) DS3 Clusters

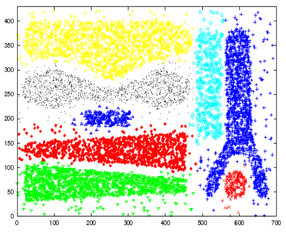
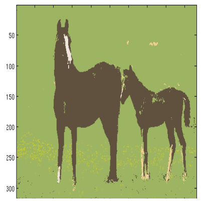
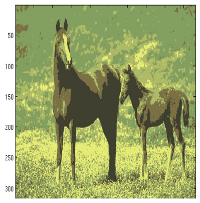

(f) DS4 Clusters

Figure 12: ABACUS Results: Initial Dataset, Backbone, and Final Clustering on DS3, and DS4.

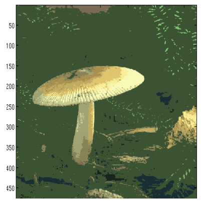
competing algorithms on the NATIMG images. The number of pixels in these images is also shown (the images are 481×321 in size, giving a total of 154401 pixels). Due to its simplicity, K-Means is much faster than ABACUS, but at the same time, K-Means is sensitive to small variations in the color space, resulting in inferior clustering quality. DBSCAN had to be forcefully terminated since it did not finish even after 6hrs for all the datasets in NATIMG. For the Mushroom, Pyramid, and Road datasets SPARCL (LOF)

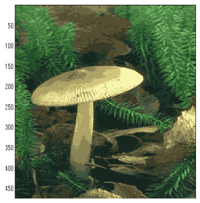
(a) Horse


(b) ABACUS

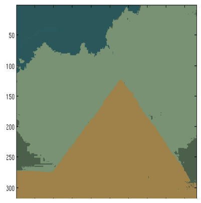

(c) K-Means

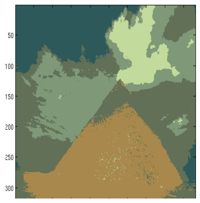
(d) Mushroom

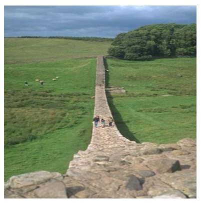

(e) ABACUS

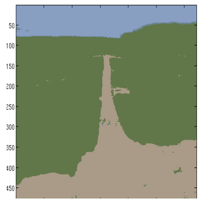

(f) K-Means

(g) Pyramid


(h) ABACUS


(i) K-Means


(j) Road


(k) ABACUS

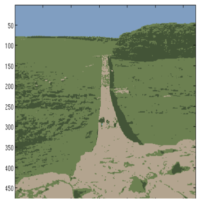

(l) K-Means

Figure 13: ABACUS and K-Means on NATIMG: Horse ($C = 5$), Mushroom ($C = 15$), Pyramid ($C = 5$), and Road ($C = 4$).

had to be manually terminated, since it failed to terminate even after a 30 minutes (probably due to some bug in the code). We also applied ABACUS to geospatial satellite imagery. For instance, Figure 14(a) shows the original image of Baghdad. The image was pre-processed using Sobel edge mask before a half-toning filter is applied. The half-toned image is shown in Figure 14(b). Note that the pre-processing results in clearer half-toning, but does not aid in the clustering directly. ABACUS with CHAMELEON for phase two results in the clusters shown in Figure 14(c). As seen, the clusters correspond to the land masses separated by the Tigris river. Although SPARCL took less time, the clustering quality was far inferior, as shown in Figure 14(d). Figure 14 shows two other examples of applying ABACUS to geospatial data taken from Earth-as-Art site [4] (Landsat-7 Satellite). Figure 14(e) is a satellite image of the Netherlands delta region, whereas Figure 14(f) is an image of Himalayan Snow-capped peaks in China.

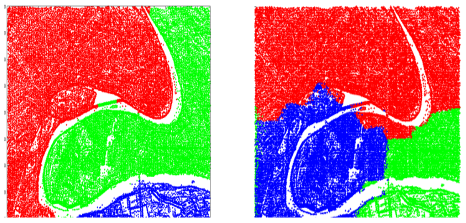
Table 4 summarizes the runtime for each algorithm. The parameters used for SPARCL (LOF) were $K = 30, minPts = 15, C = 3$. The reduction in the size of the original GEOIMG1 dataset at the end of phase 1 was 83.6%. For GEOIMG2 and GEOIMG3 again, SPARCL is more efficient as compared to ABACUS. The reduction obtained at the end of phase 1 of ABACUS for GEOIMG2 and GEOIMG3 is 55.44% and 56.43%, respectively. This explains the larger time taken by ABACUS. For all GEOIMG experiments, we used $k = 30$. Note that KASP ran out of memory for GEOIMG3 (we used $\gamma = 15, \sigma = 100$).

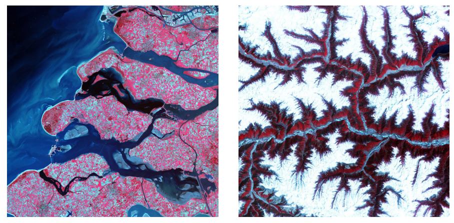
Name	ABACUS (Chameleon)	SPARCL (LOF)	CHAME-LEON	KASP
10K	0.91/0.97	0.94/0.96	1.0/1.0	0.43/0.55
50K	0.95/0.97	0.94/0.96	0.99/0.99	0.44/0.56
100K	0.95/0.965	0.91/0.96	0.99/0.99	0.43/0.55
200K	0.95/0.974	0.91/0.95	0.99/0.98	-
400K	0.95/0.974	0.95/0.98	0.99/0.99	-
600K	0.95/0.974	0.91/0.96	0.99/0.99	-
800K	0.99/0.99	0.95/0.98	0.99/0.99	-
1000K	0.95/0.97	0.91/0.95	0.98/0.99	-

Table 5: Clustering quality results on synthetic datasets. Each entry shows the Purity/NMI Score

4.5 Clustering Quality Results Since arbitrary shaped clusters do not respect similarity measures in the metric sense, internal clustering quality measures such as sum-of-squared error with respect to the cluster mean are essentially meaningless. As a result, we utilize external quality measures to evaluate the performance of ABACUS. External quality measures evaluate the clustering quality as compared to the ground truth clustering. For evaluating the clustering quality of ABACUS, we use two external criteria – *purity score* and *Normalized Mutual Information* (NMI). Purity is given by $purity(\mathcal{C}_a, \mathcal{C}_{gt}) = \frac{1}{N}\sum_{i=0}^{i=k} \max_j \|c_a^i \cap c_{gt}^j\|$, where $\mathcal{C}_a$ and $\mathcal{C}_{gt}$ denote the clusterings obtained from ABACUS and the ground truth, respectively, and c_a^i denotes the i^{th} cluster from $\mathcal{C}_a$. Although purity is a simple measure with an easy interpretation, it tends to be biased towards a clustering with higher number of clusters. Normalized mutual information (NMI) overcomes this drawback. NMI is given by $NMI(\mathcal{C}_a, \mathcal{C}_{gt}) = \frac{I(\mathcal{C}_a;\mathcal{C}_{gt})}{\frac{H(\mathcal{C}_a)+H(\mathcal{C}_{gt})}{2}}$ where I denotes the mutual information: $I(\mathcal{C}_a;\mathcal{C}_{gt}) = \sum_k \sum_j \frac{\|c_a^i \cap c_{gt}^j\|}{N} \log \frac{N\|c_a^i \cap c_{gt}^j\|}{\|c_a^i\|\|c_{gt}^j\|}$ and H denotes the Entropy: $H(\mathcal{C}_m) = -\sum_j \frac{\|c_m^j\|}{N} \log \frac{\|c_m^j\|}{N}, m \in \{a, gt\}$. Both purity and NMI scores lie in the range $[0, 1]$.

(a) GEOIMG-1: Baghdad (b) Sobel filter/halftoning


(c) ABACUS: 3 clusters (d) SPARCL: 3 clusters


(e) GEOIMG-2: Netherlands (f) GEOIMG-3: Himalaya

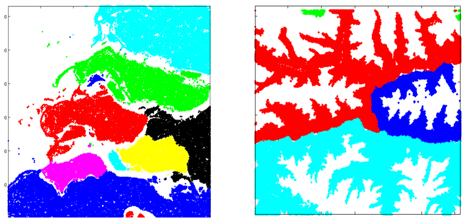

(g) ABACUS: 10 Clusters (h) ABACUS: 4 clusters

Figure 14: Clusters in the GEOIMG datasets

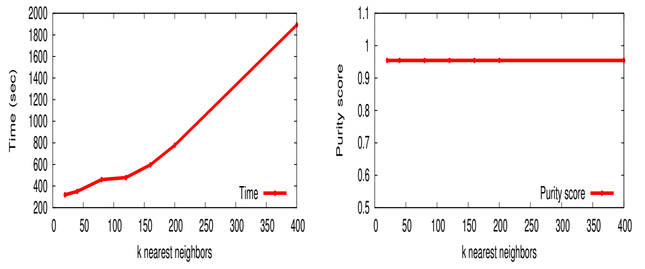
Before measuring the clustering quality, we eliminate the noise points, since different algorithms deal with noise points differently. Note that since we do not know the ground truth for the real datasets (NATIMG and GEOIMG), and we also do not know it for the synthetic datasets DS1-DS4, we cannot use the external quality measures for these datasets.

To evaluate the clustering quality of the different meth-

[4] http://earthasart.gsfc.nasa.gov/

ods, Table 5 shows the Purity and NMI scores for the synthetic datasets we used earlier in the scalability experiments (DS-SCAL), where we know the ground truth (i.e., there are $C = 13$ true clusters). We can see that the purity score and NMI are both fairly stable across the methods. The quality scores for ABACUS are comparable or better to those for SPARCL. On the other hand, the clustering quality obtained from CHAMELEON is the best. Note that the memory requirement for KASP makes it inoperable beyond datasets of size 100K. Also its purity and NMI scores were rather poor.

4.6 Parameter Sensitivity Results We performed experiments to test the sensitivity of ABACUS to the input parameter k (number of nearest neighbors). For a given dataset, we alter k and record the clustering quality. We selected the synthetic dataset with 800K points (2D with 13 clusters) for this experiment. Figure 15 shows the execution time and purity



(a) Execution time (b) Purity score

Figure 15: ABACUS: Sensitivity to k, Execution time and Purity

as the number of nearest neighbors k is increased. For the range of k shown, clusters remain intact at the end of phase 1. As a result the purity score remains almost the same. Although this might not be true for much higher values of k for which true clusters can get split. From Figure 15(a) one can see that the execution time increases linearly as the k parameter is gradually increased, since the time to compute the kNN increases linearly with k.

5 Conclusion

In this paper we proposed a scalable and robust algorithm ABACUS for clustering large spatial point datasets. The algorithm is based on the notion that each spatial cluster can be represented by its intrinsic shape, commonly known as the skeleton within the image processing community and as the backbone in this work. To identify the backbone, ABACUS performs two steps (globbing and movement) iteratively, resulting in a substantially reduced dataset that still captures the shape of the clusters. Finding clusters in the backbone amounts to identifying clusters in the original dataset. From the experimental evaluation we see that the algorithm is more scalable as compared to contemporary arbitrary shaped clustering algorithm.

Eliminating the dependency of the second phase of ABACUS on the number of true clusters is a task for the future.

6 Acknowledgements

This work was supported in part by NSF grant EMT-0829835, and NIH grant 1R01EB0080161-01A1.

References

[1] P.S. Bradley, U.M. Fayyad and C. Reina, *Scaling Clustering Algorithms to Large Databases*, Knowledge Discovery and Data Mining, 1998, pp. 9–15.

[2] M.M. Breunig, H.P. Kriegel, P.Kröger, Peer and J. Sander, *Data bubbles: quality preserving performance boosting for hierarchical clustering*, Proceedings of ACM SIGMOD International Conference on Management of Data, 2001, pp. 79–90.

[3] V. Chaoji, M.A. Hasan, S. Salem and M.J. Zaki, *SPARCL: Efficient and Effective Shape-Based Clustering*, Eighth IEEE International Conference on Data Mining, Dec. 2008, pp. 93–102.

[4] Y. Cheng, *Mean Shift, Mode Seeking, and Clustering*, IEEE Trans. Pattern Anal. Mach. Intell., 17:8 (1995), pp. 790–799.

[5] L. Ertöz, M. Steinbach and V. Kumar, *Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data*, Proceedings of Second SIAM International Conference on Data Mining, 2003.

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

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

[8] A. Hinneburg and D.A. Keim, *An Efficient Approach to Clustering in Multimedia Databases with Noise*, 4th Int'l Conf. on Knowledge Discovery and Data Mining, 1999, pp. 58–65.

[9] G. Karypis, E.-H. Han and V. Kumar, *Chameleon: Hierarchical Clustering Using Dynamic Modeling*, Computer, 32:8 (1999), pp. 68–75.

[10] R.T. Ng and J. Han, *CLARANS: A Method for Clustering Objects for Spatial Data Mining*, IEEE Trans. on Knowl. and Data Eng., 14:5 (2002), pp. 1003–1016.

[11] J. Rissanen, *Modeling By Shortest Data Description*, Automatica, 14 (1978), pp. 465–471.

[12] J. Shi and J. Malik, *Normalized Cuts and Image Segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22:8 (2000), pp. 888–905.

[13] Y. Shi, Y. Song and A. Zhang, *A shrinking-based approach for multi-dimensional data analysis*, Proceedings of the 29th International Conference on Very Large Data Bases, VLDB Endowment, 2003, pp. 440–451.

[14] D. Yan, L. Huang and M.I. Jordan, *Fast approximate spectral clustering*, Proceedings of the 15th ACM SIGKDD international Conference on Knowledge Discovery and Data mining, 2009, pp. 907–916.