Using the Fractal Dimension to Cluster Datasets

Daniel Barbará *Ping Chen*
George Mason University [*]
Information and Software Engineering Department
Fairfax, VA 22303
`{dbarbara,pchen}@gmu.edu`

October 19, 1999

Paper 145

Abstract

Clustering is a widely used knowledge discovery technique. It helps uncovering structures in data that were not previously known. Clustering of large datasets has received a lot of attention in recent years. However, clustering is a still a challenging task since many published algorithms fail to do well in scaling with the size of the dataset and the number of dimensions that describe the points, or in finding arbitrary shapes of clusters, or dealing effectively with the presence of noise. In this paper, we present a new clustering algorithm, based in the fractal properties of the datasets. The new algorithm, which we call Fractal Clustering (FC) places points incrementally in the cluster for which the change in the fractal dimension after adding the point is the least. This is a very natural way of clustering points, since points in the same cluster have a great degree of self-similarity among them (and much less self-similarity with respect to points in other clusters). FC requires one scan of the data, is suspendable at will, providing the best answer possible at that point, and is incremental. We show via experiments that FC effectively deals with large datasets, high-dimensionality and noise and is capable of recognizing clusters of arbitrary shape.

[*]This work has been supported by NSF grant IIS-9732113

1 Introduction

Clustering is one of the most widely used techniques in data mining. It is used to reveal structure in data that can be extremely useful to the analyst. The problem of clustering is to partition a dataset consisting of n points embedded in a d-dimensional space into k sets or clusters, in such a way that the data points within a cluster are more similar among them than to data points in other clusters. A precise definition of clusters does not exist. Rather, a set of functional definitions have been adopted. A cluster has been defined [1] as a set of entities which are alike (and different from entities in other clusters), an aggregation of points such that the distance between any point in the cluster is less than the distance to points in other clusters, and as a connected region with a relatively high density of points. Our method adopts the first definition (likeness of points) and use a fractal property to define similarity between points.

The area of clustering has received an enormous attention as of late in the database community. The latest techniques try to address pitfalls in the traditional clustering algorithms (for a good coverage of traditional algorithms see [14]). These pitfalls range from the fact that traditional algorithms favor clusters with spherical shapes (as in the case of the clustering techniques that use centroid-based approaches), are very sensitive to outliers (as in the case of all-points approach to clustering, where all the points within a cluster are used as representative of the cluster), or are not scalable to large datasets (as is the case with all traditional approaches).

New approaches need to satisfy the data mining desiderata [4]:

- Require at most one scan of the data.
- Have on-line behavior: provide the best answer possible at any given time and be suspendable at will.
- Be incremental by incorporating additional data efficiently.

In this paper we propose a clustering algorithm that follows this desiderata, while providing a very natural way of defining clusters that is not restricted to spherical shapes (or any other type of shape). This algorithm is based on fractal properties (namely, the fractal dimension) and clusters points in such a way that data points in the same cluster are more *self-affine* among themselves than to points in other clusters.

This paper is organized as follows. Section 2 offers a brief introduction to the fractal concepts we need to explain the algorithm. Section 3 describes our technique. Section 4 summarizes experimental results that we have obtained using our technique. Finally, Section 5 offers conclusions and guidelines for future work.

2 Fractal dimension

Nature is filled with examples of phenomena that exhibit seemingly chaotic behavior, such as air turbulence, forest fires and the like. However, under this behavior it is almost always possible to find *self-similarity*, i.e. an invariance with respect to the scale used. The structures that appear as a consequence of self-similarity are known as *fractals* [17].

Fractals have been used in numerous disciplines (for a good coverage of the topic of fractals and their applications see [22]). In the database arena, fractals have been successfully used to analyze R-trees [7], Quadtrees [6], model distributions of data [8] and selectivity estimation [2].

Fractal sets are characterized by their fractal dimension. In truth, there exists an infinite family of fractal dimensions. By embedding the dataset in an n-dimensional grid which cells have sides of size r, we can compute the frequency with which data points fall into the i-th cell, p_i, and compute D_q, the generalized fractal dimension [10, 11], as shown in Equation 1.

$$D_q = \frac{1}{q-1} \frac{\partial \log \sum_i p_i^q}{\partial \log r} \tag{1}$$

Among the dimensions described by Equation 1, the *Hausdorff fractal dimension* ($q = 0$), the *Information Dimension* ($\lim_{q \to 1} D_q$), and the *Correlation dimension* ($q = 2$) are widely used. The Information and Correlation dimensions are particularly useful for data mining, since the numerator of D_1 is Shannon's entropy, and D_2 measures the probability that two points chosen at random will be within a certain distance of each other. Changes in the Information dimension mean changes in the entropy and therefore point to changes in trends. Equally, changes in the Correlation dimension mean changes in the distribution of points in the dataset.

The traditional way to compute fractal dimensions is by means of the box-counting plot. For a set of N points, each of D dimensions, one divides the space in grid cells of size r (hypercubes of dimension D). If $N(r)$ is the number of cells occupied by points in the dataset, the plot of $N(r)$ versus r in log-log scales is called the *box-counting plot*. The negative value of the slope of that plot corresponds to the Hausdorff fractal dimension D_0. Similar procedures are followed to compute other dimensions, as described in [16].

To clarify the concept of box-counting, let us consider the famous example of George Cantor's dust, constructed in the following manner. Starting with the closed unit interval [0,1] (a straight-line segment of length 1), we erase the open middle third interval $(\frac{1}{3}, \frac{2}{3})$ and repeat the process on the remaining two segments, recursively. Figure 1 illustrates the procedure. The "dust" has a length measure of zero and yet contains an uncountable number of points. The Hausdorff dimension can be computed the following way: it is easy to see that after for the set obtained after n iterations, we are left with $N = 2^n$ pieces, each of length $r = (\frac{1}{3})^n$. So, using a unidimensional box size with $r = (\frac{1}{3})^n$, we find 2^n of the boxes populated with points. If, instead, we use a box size twice as big, i.e., $r = 2(\frac{1}{3})^n$, we get 2^{n-1} populated boxes and so on. The log-log plot of box population vs. r renders a line with slope $D_0 = -log2/log3 = -0.63....$ The value 0.63 is precisely the fractal dimension of the Cantor's dust dataset.

3 Incremental clustering using the fractal dimension

Incremental clustering using the fractal dimension (abbreviated as Fractal Clustering for short), is a form of grid-based clustering (where the space is divided in cells by a grid; other techniques that use grid-based clustering are STING [25], WaveCluster [24] and Hierarchical Grid Clustering [21]). The main idea behind FC is to group points in a cluster in such a way

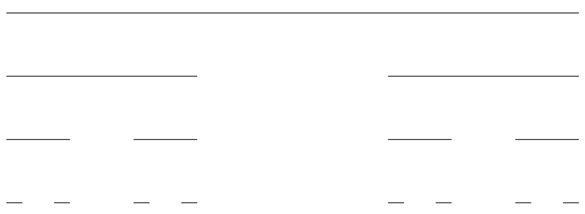

Figure 1: The construction of the Cantor dust. The final set has fractal (Hausdorff) dimension 0.63.

that none of the points in the cluster changes the the cluster's fractal dimension radically. FC also combines connectness, closeness and data points position information to pursue high clustering quality.

Our algorithm takes a first step of initializing a set of clusters, and then, incrementally adds points to that set. In what follows, we describe the initialization and incremental steps.

3.1 FC initialization step

In clustering algorithms the quality of initial clusters is extremely important, and has direct effect on the final clustering quality. Fortunately, the process of initialization is made easy by the fact that we are able to convert a problem of clustering a set of multidimensional data points (which is a subset of the original dataset) into a much simpler problem of clustering 1-dimensional points. Moreover, the problem is further simplified by the fact that the set of data points that we use for the initialization step fits in memory, Figure 2 shows the pseudo-code of the initialization step. Notice that lines 3 and 4 of the code map the points of the initial set into unidimensional values, by computing the effect that each point has in the fractal dimension of the rest of the set (we could have computed the difference between the fractal dimension of S and that of S minus a point, but the result would have been the same). Line 5 of the code deserves further explanation: in order to cluster the set of Fd_i values, we can use any known algorithm. For instance, we could feed the fractal dimension values Fd_i, and a value k to a K-means implementation [23, 9]. Alternatively, we can let a hierarchical clustering algorithm (e.g., CURE [12]) cluster the sequence of Fd_i values.

Although, in principle, any of the dimensions in the family described by Equation 1 can be used in line 4 of the initialization step, we have found that the best results are achieved by using D_2, i.e., the correlation dimension.

3.2 Incremental step

After we get the initial clusters, we can proceed to cluster the rest of the data set. Each cluster found by the initialization step is represented by a set of boxes (cells in a grid). Each box in the set records its population of points. Let k be the number of clusters found in the initialization step, and $C = \{C_1, C_2, \ldots, C_k\}$ where C_i is the set of boxes that represent cluster i. Let $F_d(C_i$ be the fractal dimension of cluster i.

```
1.Given an initial set $S$ of points $\{p_1, \cdots, p_M\}$ that fit in main memory
(obtained by sampling the dataset).
        2. For each $i = 1, \cdots, M$
                3. Define group $G_i = S - \{p_i\}$
                4. Calculate the fractal dimension of the set $G_i$, $Fd_i$.
        5. Cluster the set of $Fd_i$ values,
        (The resulting clusters are the initial clusters.)
```

Figure 2: The initialization step for FC.

```
1.Given a batch $S$ of points brought to main memory:
        2.For each point $p \in S$:
            3.For $i = 1, \cdots, k$:
                4. Let $C_i' = C_i \bigcup \{p\}$
                5. Compute $F_d(C_i')$
                6.Find $\hat{i} = min_i(|F_d(C_i' - F_d(C_i)|)$
                7.If $|F_d(C_{\hat{i}}') - F_d(C_{\hat{i}})| > \tau$
                    8.Discard $p$ as noise
                9.else
                    10. Place $p$ in cluster $C_{\hat{i}}$
```

Figure 2: The incremental step for FC.

The incremental step brings a new set of points to main memory and proceeds to take each point and add it to each cluster, computing its new fractal dimension. The pseudo-code of this step is shown in Figure 3. Line 5 computes the fractal dimension for each modified cluster (adding the point to it). Line 6 finds the proper cluster to place the point (the one for which the change in fractal dimension is minimal). Line 7 is used to discriminate "noise." If a point causes a change in the fractal dimension (of its best choice for a cluster) which is bigger than a threshold τ, then the point is simply rejected as noise (Line 8). Otherwise, it is included in that cluster. Again, we choose to use the correlation dimension, D_2, for the fractal dimension computation of Line 5 in the incremental step.

To compute the fractal dimension of the clusters every time a new point is added to them, we keep the cluster information using a series of grid representations, or layers. In each layer, boxes (i.e., grids) have a size that is smaller than in the previous layer. The sizes of the boxes are computed in the following way. For the first layer (largest boxes), we divide the cardinality of each dimension in the dataset by 2, for the next layer, we divide the cardinality of each dimension by 4 and so on. Accordingly, we get $2^D, 2^{2D}, \cdots, 2^{LD}$ D-dimensional boxes in each layer, where D is the dimensionality of the dataset, and L the maximum layer we will store. Then, the information kept is not the actual location of points in the boxes, but rather, the number of points in each box. It is important to remark that the number of boxes in layer L can grow considerably, specially for high-dimensionality datasets. However, we need only to save boxes for which there is any population of points, i.e., empty boxes are not needed. The number of populated boxes at that level is, in practical datasets, considerably smaller (that is precisely why clusters are formed, in the first place). Let us denote by B the number of populated boxes in level L. Notice that, B is likely to remain very stable throughout passes over the incremental step.

Every time a point is assigned to a cluster, we register that fact in a table, adding a row that maps the cluster membership to the point identifier (rows of this table are periodically saved to disk, freeing the space for new rows). The array of layers is used to drive the computation of the fractal dimension of the cluster, using a box-counting algorithm. In particular, we chose to use FD3 [20], an implementation of a box counting algorithm based on the ideas described in [16].

3.3 Reshaping clusters in mid-flight

It is possible that the number and form of the clusters may change after having processed a set of data points using the step of Figure 3. This may occur because the data used in the initialization step does not accurately reflect the true distribution of the overall dataset or because we are clustering an incoming stream of data, whose distribution changes over time. In this case, we want to revise the composition of each cluster and see if we need to split any of them into two or more clusters. A good indication that a cluster may need reshaping is given by how much the fractal dimension of the cluster has changed since its inception during the initialization step. (This information is easy to keep and does not occupy much space.) A large change may indicate that the points inside the cluster do not belong together. (Notice that these points were included in that cluster because it was the *best choice* at the time, i.e., it was the cluster for which the points caused the least amount of change on the fractal dimension; but this does not mean this cluster is an ideal choice for the points.)

Once the decision of revising a cluster has been made, the actual procedure is simple. Since we know the points that belong to this cluster, we can bring a sample of them to main memory and use this sample to run the initialization step shown in Figure 2. That will define how many clusters (if more than one) are needed to represent the set of points. The rest of the points in the original cluster can be incorporated in the new set of clusters by means of applying the incremental step of Figure 3 to them.

It is worth noticing at this point that we do not consider merging clusters, simply because the nature of our algorithm guarantees that points that have been placed in different clusters are sufficiently dissimilar to remain that way throughout the clustering procedure (if a point had been placed in a different cluster, it would have caused a greater change in that cluster's fractal dimension than it did for the cluster where it was placed).

3.4 Complexity of the algorithm

We assume that the cost of computing the fractal dimension of a set of n points is $O(n\log(n))$, as it is the case for the software (FD3 [20]) that we have chosen for our experiments.

For the initialization step, the complexity is $O(M^2\log(M))$, where M is the size of the sample of points. This follows from the fact that for each point in the sample, we need to compute the fractal dimension of the rest of the sample set (minus the point), incurring a cost of $O(M\log(M))$ per point. The incremental step is executed $O(N)$ times, where N is the size of the dataset. The complexity of the incremental step is $O(n\log(n))$ where n is the number of points involved in the computation of the fractal dimension. Now, since we do not use the point information, but rather the box population to drive the computation of the fractal dimension, we can claim that n is $O(B)$ (the number of populated boxes in the highest layer). Now, since $B << N$, it follows that the incremental part of FC will take time linear with respect to the size of the dataset.

For small datasets, the initialization time becomes dominant in FC. However, for large datasets, i.e., when $M << N$, the cost of the incremental step dominates, making FC linear in the size of the dataset.

3.5 Memory management

Our algorithm is very space-efficient, by the virtue of requiring memory just to hold the boxes population at any given time during its execution. This fact makes FC scale very well with the size of the set. Notice that if the initialization sample is a good representative of the rest of the data, the initial clusters are going to remain intact (just containing large populations in the boxes). In that case, the memory used during the entire clustering task remains stable.

However, there are cases in which we will have demands beyond the available memory. Mainly, there are two cases where this can happen. If the sample is not a good representative (or the data changes with time in an incoming stream) we will be forced to change the number and structure of the clusters (as explained in Section 3.3), possibly requiring more space. The other case arises when we deal with high dimensional sets, where the number of boxes needed to describe the space may exceed the available memory.

For these cases, we have devised a series of memory reduction techniques that aim to achieve reasonable trade-offs between the memory used and the performance of the algorithm, both in terms of its running time and the quality of the uncovered clusters.

3.5.1 Memory reduction technique 1:

A box in level l will be divided onto 2^D boxes in the next level $l+1$. In order to store the population of all these boxes, we need space proportional to $(1+2^D)$. However, the number of points in the bigger box is the sum of the numbers of all the 2^D smaller boxes, so space can be storing only the population for 2^D boxes. Notice that the quality of the clusters remains unaltered.

3.5.2 Memory reduction technique 2:

If we only keep the numbers of points in the smallest boxes in memory, we can compute the population of bigger boxes by aggregation. Again, this method only affects the running time, but not the quality of the clusters.

3.5.3 Memory reduction technique 3:

In this technique, we cache boxes in memory, while keeping others swapped out to the disk, replacing the ones in memory on demand. Our experience shows that the boxes of smallest size consume 75% of all memory. So, we share the cache only amongst the smallest boxes, keeping the other layers always in memory. Of course, we cluster the boxes in pages, and use the pages as a caching unit. This reduction technique affects the running time but not the clustering quality.

3.5.4 Memory reduction technique 4:

A way of requiring less memory is to ignore boxes with very few points. While this method can, in principle, affect the quality of clusters, it may actually be a good way to eliminate noise from the data set.

4 Experimental results

In this section we will show the results of using FC to cluster a series of datasets. Each dataset aims to test how well FC does in each of the issues we have discussed in the Section 3. For each one of the experiments we have used a value of $\tau = 0.03$ (the threshold used to decide if a point is noise or it really belongs to a cluster). We performed the experiments in a Sun Ultra2 with 500 Mb. of RAM, running Solaris 2.5. The initialization step was performed as indicated in Section 3.1, using K-means to cluster the unidimensional vector of effects. Then, the algorithm proceeded to incrementally cluster the rest of the points using the technique described in 3.2. In each of the experiments, the points are distributed equally among the clusters (i.e., each cluster has the same number of points). After we run FC, for each cluster found, we count the number of points that were placed in that cluster and that also belonged there. The accuracy of FC is then measured for each cluster as the percentage of points correctly placed there. (We know, for each dataset, the membership of each point; in one of the datasets we spread the space with outliers: in that case, the outliers are considered as belonging to an extra "cluster.")

4.1 Scalability

In this subsection we show experimental results of running time and cluster quality using a range of datasets of increasing sizes and a high-dimensional dataset.

First, we use datasets whose distribution follows the one shown in Figure 4 for scalability experiments. We use a complex set of clusters in this experiment, in order to show how FC can deal with arbitrarily-shaped clusters. (Not only we have a square-shaped cluster, but also one

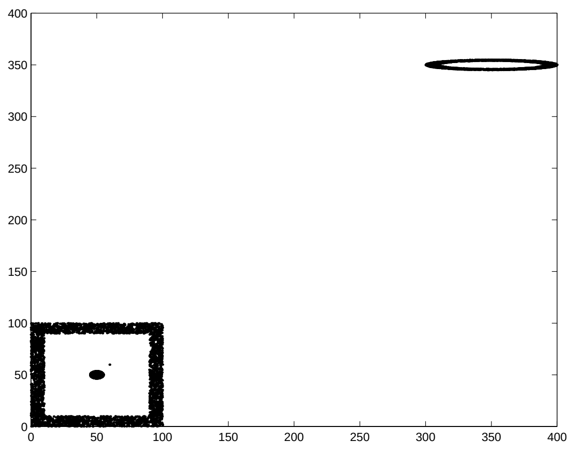

Figure 4: Three-cluster dataset used for scalability experiments.

of the clusters resides inside of another one.) We vary the total number of points in the dataset to measure the performance of our clustering algorithm. In every case, we pick a sample of 600 points to run the initialization step. The results are summarized in Figure 5. As it can be seen, the running time increases with the size of the dataset in a almost linear fashion, going from 53 seconds (48 seconds for initialization and 5 seconds for the incremental step) for the 30,000 points dataset to 5,028 seconds for the set with 30 million points. The incremental step grows linearly from 5 seconds in the 30,000 points case to 4,980 seconds in the 30 million point dataset. The figure shows that the memory taken by our algorithm is constant for the entire range of datasets (16 Mbytes). Finally, the figure shows the composition of the clusters found by the algorithm, indicating the number of points and their procedence: whether they actually belong to cluster1, cluster2 or cluster3. (Since we know to which one of the rings of Figure 4 each point really belongs). These figures are a measure of the quality of the clusters found by FC.

Figure 6 shows the results of our algorithm running on a 10 dimension data set, with two clusters. (We obviously cannot show the figure here.) The results show that our FC algorithm performs extremely well with this high-dimensional set.

4.2 Quality of the clusters

To show how FC is capable of cluster arbitrary shapes, we used the algorithm in a dataset of 10,000 points shown in Figure 7. The dataset is organized in two clusters: a tree-like structure and a ring cluster. We used a sample of 1,000 points to initialize the clusters, and the initialization step took 136 seconds. (Which is the dominating factor, in this case, for the running time of 140 sec.) Figure 8, shows how FC places every point in the correct cluster. (The time reported is the sum of the initialization step and that taken by the incremental clustering of the rest of the points).

N	time	memory	clusters found	assigned to	Coming from cluster1	cluster2	cluster3	accuracy
30,000	53s	16 Kb.	1	10,326	9,972	0	354	99.72 %
			2	11,751	0	10,000	1,751	100 %
			3	7,923	28	0	7,895	78.95 %
300,000	91s	16 Kb.	1	103,331	99,868	0	3,463	99.86%
			2	117,297	0	100,000	17,297	100.00%
			3	79,372	132	0	79,240	79.24 %
3,000,000	526s	16 Kb.	1	1,033,795	998,632	0	35,163	99.86%
			2	1,172,895	0	999,999	173,896	99.99%
			3	793,310	1,368	0	791,942	79.19%
30,000,000	5,028s.	16 Kb.	1	10,335,024	9,986,110	22	348,897	99.86%
			2	11,722,887	0	9,999,970	1,722,917	99.99%
			3	7,942,084	13,890	8	7,928,186	79.28%

Figure 5: Results of using FC in a dataset (of several sizes) whose composition is shown in Figure 4. The table shows the dataset size (N), the running time for FC (time), the main memory utilization (memory), and the composition for each cluster found in terms of points assigned to the cluster (points in cluster) and their provenance, i.e., whether they actually belong to cluster1, cluster2 or cluster3. Finally, the accuracy column shows the percentage of points that were correctly put in each cluster.

N	time	memory	clusters found	points in cluster	Coming from cluster1	cluster2	accuracy
200000	206 s.	2 Mb.	1	94,333	94,333	0	94.33%
			2	105,667	5,667	100,000	100 %

Figure 6: Results of using FC in a dataset of 10 dimensions and two clusters.

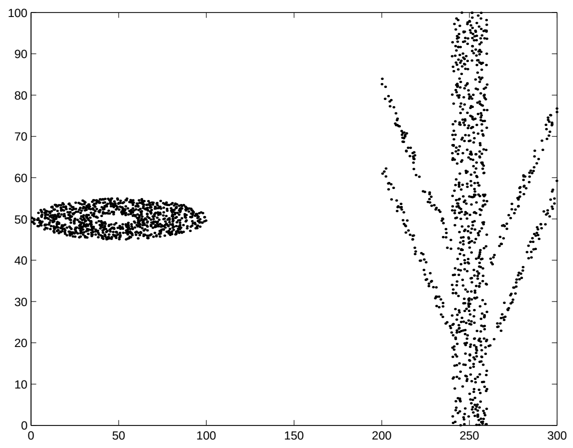

Figure 7: A complex dataset used for measuring the quality of clusters obtained by the fractal dimension algorithm.

N	time	memory	clusters found	points in cluster	Coming from cluster1	Coming from cluster2	accuracy
10,000	52s.	16 Kb.	1	10,000	10,000	0	100%
			2	10,000	0	10,000	100%

Figure 8: Results of using FC in the dataset of Figure 7.

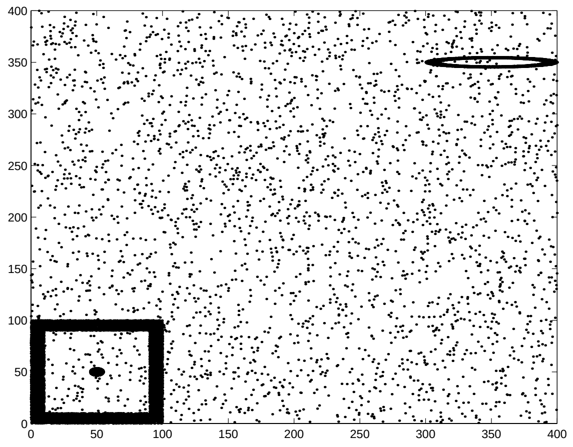

Figure 9: A noisy dataset with three clusters and 5 % of noise.

N	time	memory	clusters found	points in cluster	Coming from cluster1	cluster2	cluster3	outliers	accuracy
300,000	96s.	16 Kb.	1	98,911	94,598	0	3,276	1,037	99.57%
			2	107,485	0	95,000	12,280	205	100%
			3	79,846	402	0	79,444	0	83.62 %
			outliers	13,758	0	0	0	13,758	91.72%

Figure 10: Results of using FC on the noisy dataset of Figure 9.

4.3 Resistance to noise

For this experiment, we used the dataset shown in Figure 9, similar to that shown in Figure 4, but with a set of points that act as noise. The data set used has 300,000 points with 5% "noisy" points, i.e., 285,000 points belong into the clusters, while 15,000 points are outliers. Figure 10 shows the results of this experiment. FC places 286,242 points into the clusters (i.e., 1,242 outliers are incorrectly placed into clusters) and discards 13,758 points (91.72 % of the noise).

4.4 Evaluation of memory reduction techniques

In this section we evaluate our memory reduction techniques, using the dataset shown in Figure 4. The chosen dataset had 300,000 points. The results of running FC without any memory reduction technique can be found among those reported in Figure 5.

N	time	memory	clusters found	assigned to	Coming from			accuracy
					cluster1	cluster2	cluster3	
300,000	105s.	13 Kb.	1	103,331	99,868	0	3,463	99.86%
			2	117,297	0	100,000	17,297	100.00%
			3	79,372	132	0	79,240	79.24 %

Figure 11: Results of applying memory reduction technique 1 to the dataset of Figure 4.

N	time	memory	clusters found	assigned to	Coming from			accuracy
					cluster1	cluster2	cluster3	
300,000	436s.	12 Kb.	1	103,331	99,868	0	3,463	99.86%
			2	117,297	0	100,000	17,297	100.00%
			3	79,372	132	0	79,240	79.24 %

Figure 12: Results of applying memory reduction technique 2 to the dataset of Figure 4.

4.4.1 Technique 1

Figure 11 shows the results of technique 1 (using the sum constraint to save the space for one box) being applied to the dataset in Figure 4. The clusters composition remains unaltered (as expected), while the running time is essentially the same as took by when not using memory management. The savings in memory are 18.75 %.

4.4.2 Technique 2

Figure 12 shows the results of applying technique 2. As expected, the clusters' composition remained unchanged, and the running time increased substantially with respect to that of obtained without memory reduction, due to the need of computing the populations of boxes other than those in the highest layer. There is a savings of 25 % in the memory needed to store the boxes, with respect to the 16 Mb needed when no memory reduction is applied.

4.4.3 Technique 3

Figure 13 shows the results of applying technique 3. As expected, the clusters' composition remained unchanged, and the running time increased substantially (4-fold) with respect to the running time without memory reduction, due to the swapping of boxes in and out of main memory. The main memory utilization was kept at 4Mb, a reduction of 75 % with respect to the memory needed without memory reduction.

4.4.4 Technique 4

Figure 14 shows the results of applying technique 4 to the dataset of Figure 4, with 300,000 points. We performed the experiment with three different policies: discarding boxes with less

N	time	memory	clusters found	assigned to	Coming from cluster1	cluster2	cluster3	accuracy
300,000	404s.	4 Kb.	1	103,331	99,868	0	3,463	99.86%
			2	117,297	0	100,000	17,297	100.00%
			3	79,372	132	0	79,240	79.24 %

Figure 13: Results of applying memory reduction technique 3 to the dataset of Figure 4.

N	boxes discarded	time	memory	clusters found	assigned to	Coming from cluster1	cluster2	cluster3	accuracy
300,000	$<$ 2 pts.	91s.	7.5 Kb.	1	125,112	99,820	0	25,292	99.82%
				2	100,091	180	99,901	10	99.90%
				3	74,794	0	99	74,698	74.69 %
300,000	$<$ 5 pts.	100s.	7.4 Kb.	1	125,112	99,820	0	25,292	99.82%
				2	100,091	180	99,901	10	99.90%
				3	74,794	0	99	74,698	74.69 %
300,000	$<$ 6 pts.	94s.	7.0 Kb.	1	125,112	99,820	0	25,292	99.82%
				2	100,091	180	99,901	10	99.90%
				3	74,794	0	99	74,698	74.69 %

Figure 14: Results of applying memory reduction technique 4 to the dataset of Figure 4. The column "boxes discarded" indicates the policy used. (E.g., $<$ 5 pts. indicates all the boxes with less than five points were discarded.)

than 2, 5 and 6 points respectively. The memory reduction spans from 53.1 % to to 56.25 % with respect to the experiment without memory reduction. The running times are essentially constant. The quality of the clusters, which remains the same throughout the 3 policies, is slightly diminished with respect to that obtained without memory reduction: the accuracy of cluster1 goes to 99.82 % from 99.86 %, that of cluster2 goes to 99.90 % from 100 %, and the accuracy of cluster3 goes down to 74.69% from 79.24%. Policies that discarded boxes with higher number of points resulted in a much more marked decrease of cluster quality.

4.5 Splitting clusters

Using the dataset of Figure 4 containing 30,000 points we performed the following experiment. For the initial sample, we selected points belonging only to two of the three clusters. Then, we monitored when the change of the fractal dimension for the clusters with respect to the original fractal dimension (of the initial clusters) exceeded 0.1. At that point we took a sample of each cluster and ran the initialization algorithm again. The results obtained were similar in quality to those reported in Figure 5. The running time increased from 56 seconds to 120 seconds.

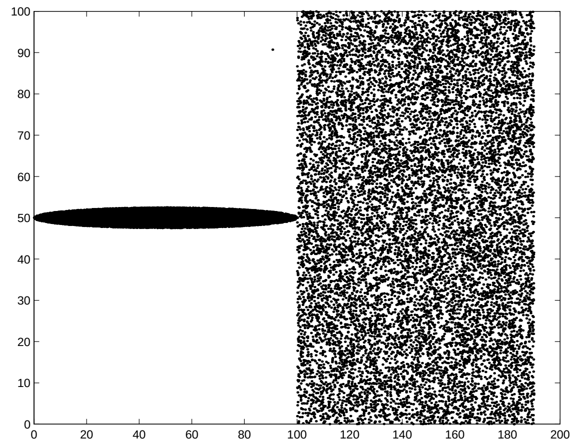

Figure 15: A dataset used to compare FC and CURE.

N	Algorithm	time	clusters found	points in cluster	Coming from cluster1	Coming from cluster2	accuracy
300	CURE	0.26s.	1	69	69	0	46 %
			2	127	0	127	84.6%
			outliers	104	-	-	
	FC	49	1	150	150	0	100%
			2	150	0	150	100 %
3,000	CURE	22s.	1	1,626	1,146	480	76.4 %
			2	349	0	349	23.26%
			outliers	1,025	-	-	
	FC	49s.	1	1,486	1,481	5	98.7%
			2	1,514	19	1,496	99.7 %

Figure 16: Results of using FC and CURE in the dataset of Figure 15.

4.6 Comparison of FC with CURE

We used FC and CURE to cluster the data set in Figure 15. Figure 16 shows how CURE and FC perform for this dataset for two different sizes (300 and 3,000 points). CURE declares 104 (34.6 %) and 1,025 (34.16 %) of the points as outliers, while FC is able to place all the points correctly. The running time of CURE is, for these datasets, smaller than that for FC (simply because the later is dominated by the initialization time). But as the datasets grow in size, we noticed that CURE's running time increases more rapidly than that of FC.

Finally, we also tried the dataset of Figure 7 with 20,000 points on CURE, to compare its results with those reported in Figure 8. Figure 17 shows the comparison. CURE declares 6,226 (31.13 %) points as outliers, placing the rest in the right clusters, while FC places all the points correctly. For this larger set, FC's running time clearly outperforms CURE's.

Algorithm	time	clusters found	points in cluster	Coming from		accuracy
				cluster1	cluster2	
CURE	2,520s	1	4,310	4,310	0	43.10 %
		2	9,464	0	9,464	94.64 %
		outliers	6,266	-	-	
FC	52s.	1	10,000	10,000	0	100 %
		2	10,000	0	10,000	100 %

Figure 17: Results of using FC and CURE in the dataset of Figure 7.

5 Related Work

The topic of clustering has received ample attention in the research community throughout the years. A recent tutorial [13], classifies clustering methods in three varieties:

5.1 Model- and Optimization-Based Approaches

Most of the initial approaches devised for clustering belong to this class. The methods usually start with an initial partition and use an iterative strategy to optimize a function.

The K-Means algorithm [23, 9] uses k initial prototypes of center of gravity of the clusters, and then iteratively assigns the data points to the nearest prototype and shifts the prototypes towards the mean of the clusters obtained. K-means was initially devised as an in-memory algorithm, but variants of it have been recently devised for large detasets that do not fit in main memory [3].

The Expectation Maximization algorithm [15] follows a similar approach to K-Means, by iterating on the process of estimating the parameters of k Gaussian distributions, optimizing the probability that the mixture of Gaussian distributions fits the data.

CLARANS [18], is the first method designed for large data sets. CLARANS k medoids, assigns points to the nearest medoid and follows an iterative procedure to optimize a distance function. CLARANS has, however, a large computational complexity [25] ($O(kN^2)$, where N is the size of the dataset) and due to its randomized approach in selecting medoids, the quality of the results cannot be guaranteed. Moreover CLARANS does not perform well for complex cluster structures.

5.2 Density-based Methods

DBSCAN [5], uses a density-based notion of clusters: for each cluster, the neighborhood of a given radius must contain at least a minimum number of points (in other words, the density of the cluster must exceed a threshold). DBSCAN can discover clusters of arbitrary shape, uses an R^*-tree to achieve better performance and has complexity $O(NlogN)$.

STING (Statistical INformation Grid-based method) [25], divides the space in rectangular cells using a hierarchical structure and proceeds to store the statistical parameters (mean, variance, type of distribution). Then the clusters are determined as the density-connected components of the grid. The complexity of the method is $O(N)$, but it fails to find some arbitrary shaped clusters.

Hierarchical Grid Clustering [21], also organizes the space as a grid, sorting the cells according to their density and scanning and merging adjacent cells to form a hierarchy.

CURE [12] is also a hierarchical clustering algorithm that tries to overcome the problems found in traditional hierarchical algorithms when they attempt to merge clusters: at one extreme the clusters are represented by a single point (or centroid), at the other by the complete set of points that are members of the cluster. Both extremes fail to deal well with non-spherical clusters. CURE adopts a middle ground approach, choosing a fixed-size set of points to represent clusters.

WaveCluster [24], is also a grid-based approach with complexity $O(N)$, that treats points as multidimensional feature vectors, quantizes the feature space, and then assigns the points to the corresponding quantized units. WaveCluster proceeds to apply wavelet transform on the feature space, find the connected components in the subbands of the transformed space, calling them clusters and finally assign the points to their corresponding clusters.

5.3 Hybrid Methods

BIRCH [26], builds a hierarchical data structure, the CF-tree -a height-balanced tree-, to incrementally cluster incoming points. BIRCH tries to come up with the best clusters with available main memory, while minimizing the amount of I/O required. Results can be improved by allowing several passes over the dataset, but in principle one pass suffices to get a clustering, so the complexity of the algorithm is $O(N)$. Since each node in the tree has predefined limited capacity, the clusters do not always correspond to natural shapes. (In [24], it is reported that BIRCH does not perform well for non-spherical clusters.) The algorithm is also sensitive to the ordering of the data.

6 Conclusions

In this paper we presented a new clustering algorithm based on the usage of the fractal dimension. This algorithm clusters points according to the effect they have on the fractal dimension of the clusters that have been found so far. The algorithm is, by design, incremental and its complexity is $O(N)$, where N is the size of the dataset (thus, it requires only one pass over the data).

Our experiments have proven that the algorithm has very desirable properties. It is resistant to noise, capable of finding clusters of arbitrary shape and capable of dealing with points of high dimensionality.

7 Acknowledgments

We like to thank Vipin Kumar and Eui-Hong (Sam) Han for lending us their implementation of CURE.

References

[1] E. Backer. *Computer-Assisted Reasoning in Cluster Analysis.* Prentice Hall, 1995.

[2] A. Belussi and C. Faloutsos. Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension. In *Proceedings of the International Conference on Very Large Data Bases*, pages 299–310, September 1995.

[3] P.S. Bradley, U. Fayyad, and C. Reina. Scaling Clustering Algorithms to Large Databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, New York City*, August 1998.

[4] P.S. Bradley, U. Fayyad, and C. Reina. Scaling Clustering Algorithms to Large Databases (Extended Abstract). In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, June 1998.

[5] M. Ester, J.P. Kriegel, J. Sander, and X. Su. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[6] C. Faloutsos and V. Gaede. Analysis of the Z-ordering Method Using the hausdorff Fractal Dimension. In *Proceedings of the International Conference on Very Large Data Bases*, pages 40–50, September 1996.

[7] C. Faloutsos and I. Kamel. Relaxing the Uniformity and Independence Assumptions, Using the Concept of Fractal Dimensions. *Journal of Computer and System Sciences*, 55(2):229–240, 1997.

[8] C. Faloutsos, Y. Matias, and A. Silberschatz. Modeling Skewed Distributions Using Multifractals and the '80-20 law'. In *Proceedings of the International Conference on Very Large Data Bases*, pages 307–317, September 1996.

[9] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, San Diego, California, 1990.

[10] P. Grassberger. Generalized Dimensions of Strange Attractors. *Physics Letters*, 97A:227–230, 1983.

[11] P. Grassberger and I. Procaccia. Characterization of Strange Attractors. *Physical Review Letters*, 50(5):346–349, 1983.

[12] S. Guha, R. Rastogi, and K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data, Seattle, Washington*, pages 73–84, 1998.

[13] A. Hinneburg and D. Keim. Clustering Techniques for Large Data Sets: From the Past to the Future. Tutorial Notes for ACM SIGKDD International Conference on Knowledge Discovery and Data MIning, 1999.

[14] A. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[15] S.L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, pages 101–201, 1995.

[16] L.S. Liebovitch and T. Toth. A Fast Algorithm to Determine Fractal Dimensions by Box Countig. *Physics Letters*, 141A(8), 1989.

[17] B.B. Mandelbrot. *The Fractal Geometry of Nature.* W.H. Freeman, New York, 1983.

[18] R.T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of the 20th Very Large Data Bases Conference*, pages 144–155, 1994.

[19] R. Rojas. *Neural Networks - A Systematic Introduction.* Springer, Springer, 1996.

[20] John Sarraille and P. DiFalco. FD3. http://tori.postech.ac.kr/softwares/.

[21] E. Schikuta. Grid clustering: An efficient hierarchical method for very large data sets. In *Proceedings of the 13th Conference on Pattern Recognition, IEEE Computer Society Press*, pages 101–105, 1996.

[22] M. Schroeder. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise.* W.H. Freeman, New York, 1991.

[23] S.Z. Selim and M.A. Ismail. K-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1), 1984.

[24] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In *Proceedings of the 24th Very Large Data Bases Conference*, pages 428–439, 1998.

[25] W. Wang, J. Yand, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd Very Large Data Bases Conference*, pages 186–195, 1997.

[26] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data, Montreal, Canada*, pages 103–114, 1996.