

# Using the Fractal Dimension to Cluster Datasets \*

Daniel Barbará  
George Mason University  
ISE Dept., MSN 4A4  
Fairfax, VA 22030 USA  
dbarbara@gmu.edu

Ping Chen  
George Mason University  
ISE Dept., MSN 4A4  
Fairfax, VA 22030 USA  
pchen@gmu.edu

## ABSTRACT

Clustering is a widely used knowledge discovery technique. It helps uncovering structures in data that were not previously known. The clustering of large data sets 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 data set 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 data sets. 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 data sets, high-dimensionality and noise and is capable of recognizing clusters of arbitrary shape.

## Categories and Subject Descriptors

I.5.3 [Computing Methodologies]: Pattern Recognition—Clustering

## General Terms

Fractals

## 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 data set 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 uses 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 [5]). 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 data sets (as is the case with all traditional approaches).

In this paper we propose a clustering algorithm that provides a very natural way of defining clusters that is not restricted to any particular cluster shape. This algorithm is based on the use of 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. (Notice that this does not mean the clusters have to be fractal data sets themselves.)

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* [7]. Fractals have been used in numerous disciplines (for a good coverage of the topic of fractals and their applications see [10]). Fractal sets are characterized by their fractal dimension. (In truth, there exists an infinite family of fractal dimensions.) By embedding the data set in an  $d$ -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 the fractal dimension [3, 4]. The traditional way to

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

compute fractal dimensions is by means of the box-counting plot. For a set of  $n$  points, each of  $d$  dimensions, the space is divided in grid cells of size  $r$  (hypercubes of dimension  $d$ ). If  $N(r)$  is the number of cells occupied by points in the data set, 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. Similar procedures are followed to compute other dimensions, as described in [6].

This paper is organized as follows. Section 2 describes our technique. Section 3 summarizes experimental results that we have obtained using our technique. Finally, Section 4 offers conclusions and guidelines for future work.

## 2. OUR METHOD

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 [12], WaveCluster [11] and Hierarchical Grid Clustering [9]). The main idea behind FC is to group points in a cluster in such a way that none of the points in the cluster changes the cluster's fractal dimension radically. After initializing a set of clusters, our algorithm incrementally adds points to that set. In what follows, we describe the initialization and incremental steps.

### 2.1 FC initialization step

Obviously, before we can apply the main concept of our technique, i.e., adding points incrementally to existing clusters, based on how they affect the clusters' fractal dimension, some initial clusters are needed. In other words, we need to "bootstrap" our algorithm via an initialization procedure that finds a set of clusters, each with sufficient points so its fractal dimension can be computed. We present in this section one initialization algorithm. (Although we have tried several algorithms for this step, space constraints limit us to showing one of the choices here.) The initialization algorithm uses a traditional distance-based procedure. We try to cluster the initial sample of points by taking points at random and finding recursively points that are close to them. When no more close points can be found, a new cluster is initialized, choosing another point at random out of the set of points that have not been clustered yet. Figure 1 presents the pseudo code for this algorithm.

Given a sample of points (which fits in main memory) and a distance threshold  $\kappa$  (Line 1), the algorithm proceeds to build clusters by picking a random yet unclustered point and recursively finding the nearest neighbor in such a way that the distance between the point and the neighbor is less than  $\kappa$ . The neighbor is then included in the cluster, and the search for nearest neighbors continues in depth-first fashion, until no more points can be added to clusters. Notice that we do not try to restrict the clusters by constraining it, as it is customarily done, to have a diameter less than or equal to a certain threshold, since we want to be able to find arbitrarily shaped clusters.

The threshold,  $\kappa$  is the product of a predefined, static parameter  $\kappa_0$ , and  $\hat{d}$ , the average distance between pairs of points already in the cluster. Thus,  $\kappa$  is a dynamic thresh-

- 
1. Given an initial sample  $S$  of points  $\{p_1, \dots, p_M\}$  that fit in main memory, and a distance threshold  $\kappa$ . (Initially  $\kappa = \kappa_0$ .)
  2. Mark all points as unclustered, and make  $k = 0$
  3. Randomly choose a point  $P$  out of the set of unclustered points
    4. Mark  $P$  as belonging to cluster  $C_k$
    5. Starting at  $P$  and in a recursive, depth-first fashion,
      - call  $P' = NEAR(P, \kappa)$
    6. If  $P'$  is not NULL
      7. Put  $P'$  in cluster  $C_k$
    8. else
      9. backtrack to the previous point in the search.
    10. Update  $\hat{d}$ , the average distance between pairs of points in  $C_k$
    11. Make  $\kappa = \kappa_0 \times \hat{d}$
  12. If there are still unclustered points, make  $k = k + 1$  and go to 3.

*NEAR*( $P, \kappa$ )  
 Find the nearest neighbor of  $P$  such that  $dist(P', P) \leq \kappa$ .  
 If no such  $P'$  can be found return NULL  
 Otherwise return  $P'$ .

---

**Figure 1: Initialization algorithm.**

---

old that gets updated as points are included in the cluster. Intuitively, we are trying to restrict the membership to an existing cluster to points whose minimum distance to a member of the cluster is similar to the average distance between points already in the cluster.

### 2.2 Incremental step

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  $C = \{C_1, C_2, \dots, C_k\}$  where  $C_i$  is the set of boxes that represent cluster  $i$ . Let  $F_d(C_i)$  be the fractal dimension of cluster  $i$ .

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 2. 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 by computing the minimal *fractal impact*, i.e., the minimal change in fractal dimension experienced by any of the current clusters when the new point is added. 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  $\tau$ , then the point is simply rejected as noise (Line 8). Otherwise, it is included in that cluster. We choose to use the Hausdorff dimension,  $D_0$ , for the frac-

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

**Figure 2: The incremental step for FC.**

tal dimension computation of Line 5 in the incremental step. We chose  $D_0$  since it can be computed faster than the other dimensions and it proves robust enough for the task.

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 data set by 2, for the next layer, we divide the cardinality of each dimension by 4 and so on. Accordingly, we get  $2^d, 2^{2d}, \dots, 2^{Ld}$   $d$ -dimensional boxes in each layer, where  $d$  is the dimensionality of the data set, 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 data sets. 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 data sets, 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, each cluster into a file, 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 [8], an implementation of a box counting algorithm based on the ideas described in [6].

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 2. This may occur because the data used in the initialization step does not accurately reflect the true distribution of the overall data set or because we are clustering an incoming stream of data. To address this issue, we

have devised two operations: splitting a cluster and merging two or more clusters into one. Further details can be found in [2].

### 3. EXPERIMENTAL RESULTS

In this section we will show the results of some experiments using FC to cluster a series of data sets. (Further experiments can be found in [2].) Each data set aims to test how well FC does in each of the issues we have discussed in the Section 2. 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. 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 data set, the membership of each point; in one of the data sets we spread the space with outliers: in that case, the outliers are considered as belonging to an extra "cluster.")

In one experiment, we used data sets whose distribution follows the one shown in Figure 3. We varied the total number of points in the data set 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 4. The number in the column "time" is the running time. The running time goes from 12 sec. (7 seconds for initialization and 5 for the incremental step) for the 30,000 points data set to 4,987 sec. (4,980 sec. for the incremental step and 7 seconds for initialization) in the case of 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 data set. The figure shows that the memory taken by our algorithm is constant for the entire range of data sets (64 Kbytes). Finally, the figure shows the composition of the clusters found by the algorithm, indicating the number of points and their precedence: whether they actually belong to cluster1, cluster2 or cluster3. (Since we know to which one of the rings of Figure 3 each point really belongs). Because both of cluster 2 and 3 are rings, they have close fractal dimension, and some points of cluster 3 are misplaced. These figures are a measure of the quality of the clusters found by FC. The quality of the clusters found by using the two initialization algorithms is very similar, so we only report it once.

Some other experiments, whose results are reported in [2], can be briefly described as follows. An experiment that used the data set shown in Figure 3 augmented with 5% of noise (points that lie outside the clusters) shows that FC is extremely resistant to noise, since it correctly identified 91.72 % of the noisy points as outliers. Experiments with data sets such as the one shown in Figure 5 demonstrated that FC is capable of identifying very arbitrarily-shaped clusters.

We also wanted to compare FC with two previously published algorithms: CURE and BIRCH. We show (for reasons of space) only one of the comparisons we performed. (Further comparisons can be found in [2].) We tried the data set

$N$	time	memory	clusters found	assigned to	Coming from			accuracy
					cluster1	cluster2	cluster3	
30,000	12s.	64 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	56s.	64 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	485s.	64 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	4,987s.	64 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 4: Results of using FC in a data set (of several sizes) whose composition is shown in Figure 3. The table shows the data set size ( $N$ ), the running time for FC (time), the main memory utilization (memory), and the composition for each cluster 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.

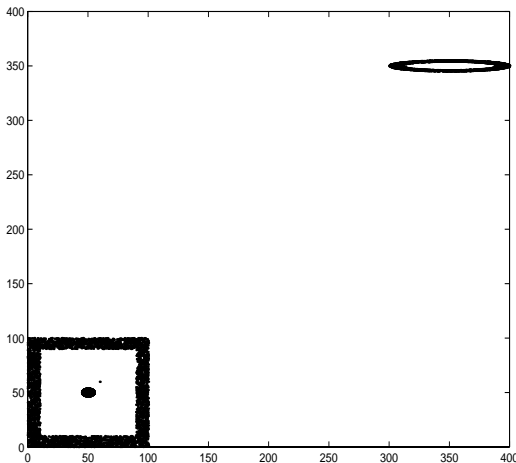


Figure 3: Three-cluster data set used for scalability experiments.

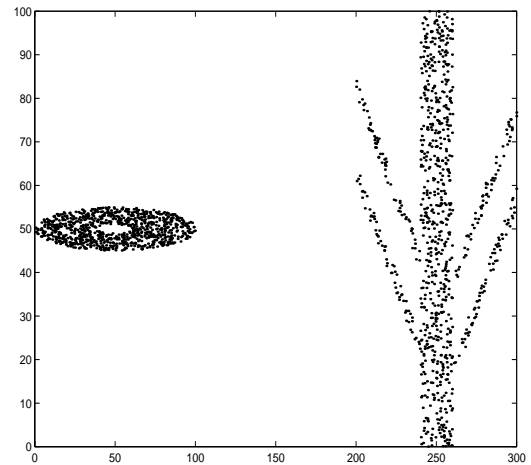


Figure 5: A complex data set .

of Figure 5 with 20,000 points for a comparison experiment whose results are shown in Figure 6. CURE declares 6,226 (31.13 %) points as outliers, placing the rest in the right clusters, while FC places all the points correctly. BIRCH declared a modest number of points (359) as outliers, placing the rest of the points correctly. For this larger set, FC's running time clearly outperforms CURE's and BIRCH's.

#### 4. 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	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	14s.	1	10,000	10,000	0	100 %
		2	10,000	0	10,000	100 %
BIRCH	3.89 s.	1	9,654	9,654	0	96.5 %
		2	9,987	0	9,987	98.7%
		outliers	359	-	-	-

Figure 6: Comparison of FC, BIRCH, and CURE.

algorithm is, by design, incremental and its complexity is linear on the size of the data set (thus, it requires only one pass over the data, with the exception of cases in which splitting of clusters needs to be done).

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.

We have also shown that our FC algorithm compares favorably with other algorithms such as CURE and BIRCH, obtaining better clustering quality. As the data sets get bigger, FC is able to outperform BIRCH and CURE in running time as well.

Although not shown here, we have successfully performed experiments with high-dimensional data sets to test FC's ability to scale with dimensions [2]. One point worth mentioning is that as the dimensionality grows, the memory demands of FC can grow beyond the available memory. To cope with this, we have devised two memory reduction techniques [2]. The first one is based on caching boxes in memory (we have discovered that the most efficient way to cache is to keep the last layer of boxes (higher granularity boxes) in memory and to bring the rest of the layers on demand). The second one is based on discarding low-populated boxes. Both deal very effectively with the memory growth: while they keep the memory usage at a reasonable level, their impact on quality and performance is low. (The first memory reduction technique has no impact on quality, but it does impact the performance of the algorithm, while the second memory technique impact is mainly on the quality of the clusters found.)

## 5. ACKNOWLEDGMENTS

We like to thank Vipin Kumar and Eui-Hong (Sam) Han for lending us their implementation of CURE. We also like to thank Raghu Ramakrishnan and Venkatesh Ganti for their BIRCH code, and for spending long hours helping us in the use of BIRCH for our experiments.

## 6. REFERENCES

- [1] E. Backer. *Computer-Assisted Reasoning in Cluster Analysis*. Prentice Hall, 1995.
- [2] D. Barbará and P. Chen. Using the Fractal Dimension to Cluster Datasets. Technical Report ISE-TR-99-08, George Mason University, Information and Software Engineering Department, Oct. 1999.
- [3] P. Grassberger. Generalized Dimensions of Strange Attractors. *Physics Letters*, 97A:227–230, 1983.
- [4] P. Grassberger and I. Procaccia. Characterization of Strange Attractors. *Physical Review Letters*, 50(5):346–349, 1983.
- [5] A. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.
- [6] L. Liebovitch and T. Toth. A Fast Algorithm to Determine Fractal Dimensions by Box Counting. *Physics Letters*, 141A(8), 1989.
- [7] B. Mandelbrot. *The Fractal Geometry of Nature*. W.H. Freeman, New York, 1983.
- [8] J. Sarraile and P. DiFalco. FD3. <http://tori.postech.ac.kr/software/>.
- [9] 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.
- [10] M. Schroeder. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise*. W.H. Freeman, New York, 1991.
- [11] 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.
- [12] 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.