CRYSTAL - A new density-based fast and efficient clustering algorithm

Priyadarshi Bhattacharya and Marina L. Gavrilova
Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, AB, Canada T2N1N4
{pbhattac, marina}@cpsc.ucalgary.ca

Abstract

In this paper, we present a fast $O(nlogn)$ clustering algorithm based on Delaunay Triangulation for identifying clusters of different shapes, not necessarily convex. The clustering result is similar to human perception of clusters. The novelty of our method is the growth model we follow in the cluster formation that resembles the natural growth of a crystal. Our algorithm is able to identify dense as well as sparse clusters and also clusters connected by bridges. We demonstrate clustering results on several synthetic datasets and provide a comparison with popular K-Means based clustering methods. The clustering is based purely on proximity analysis in the Delaunay Triangulation and avoids usage of global parameters. It is robust in the presence of noise. Finally, we demonstrate the capability of our clustering algorithm in handling very large datasets.

1. Introduction

Today, most Geographic Information Systems (GIS) handle huge volumes of spatial data. This facilitates the development of fast and efficient clustering algorithms that can determine patterns in real-time, so that further processing of data can be limited to specific regions only. Identifying all natural clusters in the data is a crucial prerequisite for a successful analysis of the data. It is used in fields such as navigation, planning, pattern recognition, AI, computer graphics to name a few.

Many types of clustering algorithms have been proposed in literature. A comparative study of the performance of recent clustering approaches can be found in [3]. In this paper, we consider statistical and graph-based approaches to clustering since these are two widely used approaches. The most well-known statistical clustering algorithm is K-Means. It is a centroid-based clustering algorithm that minimizes the summation of the Euclidean distances of each data point from its cluster-center, which is the centroid of the data points in that cluster. A drawback of K-Means is the sensitivity of clustering result towards initial positioning of the cluster centroids which very often results in convergence to a local minimum. However, a more serious problem is the requirement of specifying the number of cluster centroids. Not only is it impractical but this implies that the clustering method fails to find natural clusters in the data. To remedy the convergence to local minima and in order to find more globally optimal solutions, several variations of K-Means have been proposed [5][6]. But the fundamental problem of requiring an a-priori knowledge of dataset remains.

Recently, several graph-based clustering approaches have been introduced which do not require the number of clusters to be prespecified and attempt to find natural clusters. Some of these algorithms [1][2][8] are triangulation based. However, as reported in [2], identifying sparse clusters in presence of dense ones or clusters connected by multiple bridges remains elusive. Usage of a global threshold to determine cluster density is tricky. If most clusters have a high density, the global density tends to be high and this precludes the possibility of identifying sparse clusters. Noise in data further aggravates the problem. In this paper, we present a novel clustering algorithm which we name CRYSTAL because the formation of a cluster closely resembles the natural growth of a crystal. The cluster starts growing from a data point, first encompassing first-order neighbors, then second-order neighbors and so on until the boundary of the cluster is reached. The approach is based on a local density analysis and avoids the use of global parameters. We utilize the Delaunay Triangulation as an ideal data structure for preserving proximity information. We present results that demonstrate that our algorithm can indeed identify sparse clusters in presence of dense ones, closely located high-density clusters and even clusters linked with multiple bridges. The clustering closely resembles human visual perception of clusters as demonstrated by our experimental results. We also observe that the algorithm is fast and practical for very large datasets.

2. Literature Review

Several variations of the K-Means have been proposed in literature to reach more globally optimal solutions and reduce the vulnerability of the method to initial positioning of cluster-centroids. The global K-Means algorithm proposed in [6] starts with only one cluster-center. It then finds an appropriate position to place a new cluster-center based on minimization of an objective function and the process continues. After addition of a new cluster-center, the K-Means algorithm is run to repartition the data. Although the method reaches a more optimal solution by solving the clustering problem on a larger scale, it invokes the K-Means repeatedly which makes it impractical for large datasets. The Greedy Elimination Method proposed in [5] starts with approximately twice the required number of cluster-centers and then adopts a greedy approach to determine which cluster-center to remove at each step. Although we found this algorithm to be faster than [6], it still is not fast enough and suffers from the same fundamental problems identified above that K-Means suffers from. The clusters reported are roughly spherical in shape and they fail to identify many interesting patterns in data. Also, arriving at a suitable value for the number of clusters requires invoking the basic K-Means many times and this makes these algorithms incapable of handling very large datasets efficiently.

Graph-based clustering algorithms are able to detect clusters of much more varied shapes and densities. Two recently introduced algorithms attracted our attention. In [1], the clustering is performed on a Reduced Delaunay Graph. The method is able to identify many interesting cluster shapes. But it is not mentioned how well the algorithm performs in case of co-existence of sparse and dense datasets or in the presence of noise. The clustering algorithm proposed in [2] attempts to fully automate the clustering process so that no a-priori knowledge of the dataset is required. It is also able to detect clusters of widely varying densities. However, for cluster formation, it adopts a strategy of first deleting short edges from the graph and then recuperating them, which in our opinion is a costly operation. We propose a new idea based on Delaunay Triangulation. In our approach, we use the proximity information in the Delaunay Triangulation to grow clusters from data points and a data point once assigned to a cluster is not moved again. Also, in contrast to [2] which uses local standard deviation, we perform our analysis based on the mean of the edge lengths in the Delaunay Triangulation and this significantly reduces the computation cost. This approach, coupled with our natural crystal-like cluster growing mechanism ensures fast and efficient cluster detection.

3. CRYSTAL - Algorithm Description

3.1. Clustering based on Delaunay Triangulation

First, we explored the possibility of using the Minimal Spanning Tree in cluster analysis. Clustering based on Minimal Spanning Tree can be found in [7]. However, we discovered that the minimum spanning tree retains considerably less proximity information than the Delaunay Triangulation, which is prohibitive in correctly identifying clusters in subtle cases such as clusters joined by multiple bridges. Hence, we based our clustering algorithm on the Delaunay Triangulation.

We construct the triangulation in $O(nlogn)$ using the incremental algorithm [10]. We use an auxiliary grid structure to speed up the point location problem in the Delaunay Triangulation. This reduces length of walk in the graph to locate the triangle containing the data point considerably. We implement the walk similar to the process mentioned in [9]. Except in case of highly skewed data distributions, we observe that this generates the triangulation in near linear time. We ensure that we are not exceeding $O(n)$ storage by not allowing the number of buckets in the grid to exceed $\sqrt{n}$ in both horizontal and vertical directions. We observe that the average length of adjacent edges for boundary vertices is greater than those that are inside a cluster. In order to grow the cluster uniformly, it is important that we start from an inner data point rather than one on the boundary. So we first sort the vertices in order of decreasing average edge length of adjacent edges. This also ensures that we identify the more dense clusters before the sparse ones. Apart from the generation of the Delaunay Triangulation, this is the only other $O(nlogn)$ step. All other operations are $O(n)$ where n is the number of data points.

3.2. Growcluster - algorithm description

For every vertex in the Delaunay Triangulation, which represents a data point, we maintain the following information:

1. Vertex index
2. Coordinates (X and Y)
3. Indices of vertices that are adjacent to it in the Delaunay Triangulation
4. Average length of all adjacent edges (*avgedglen*)
5. A flag (*incluster*) which indicates whether the vertex is assigned to a cluster

We scan the sorted vertex list and for each vertex (v_i), if it is not already assigned to a cluster, call *Growcluster* to grow a cluster from that data point. The vertex v_i gets added to the cluster and also to the queue which we require to maintain the list of vertices from which we attempt to grow the cluster. We set the average edge length in cluster ($avgclusteredglen$) to the average length of edges adjacent to v_i. Thereafter, everytime we add a new vertex to the cluster, we update the average cluster edge length ($avgclusteredglen$). To decide whether a data point is on the boundary, we check the average adjacent edge length of the vertex with the average cluster edge length. If it is greater than 1.6 times, we consider it a boundary vertex. If it is a boundary vertex, we add it to the cluster, but not to the queue so that our cluster cannot grow from that vertex. The value of 1.6 is derived empirically.

At the growing phase of the cluster, the value of $avgclusteredglen$ does not accurately reflect the local density and hence some vertices which rightfully belong to the cluster may be left out. This in particular happens when the edge length between the first two added vertices to the cluster is extremely small. To resolve this, after the initial cluster formation is over, we re-check whether there is any vertex adjacent to any of the vertices already present in the cluster for which edge length is $\leq 1.6 * avgclusteredglen$ If so, we add it to the cluster and set *incluster* for the vertex to true. After the cluster formation is over, we delete any trivial clusters based on cluster size. This can also be done based on cluster density indicated by $avgclusteredglen$.

Since a data point once assigned to a cluster is not considered again, the entire clustering is done in $O(n)$ time, where n is the number of data points.

CRYSTAL - Clustering Algorithm(D, S)

Input: The Delaunay Triangulation (**D**) of the data points
Output: Collection of all clusters (**S**)

Sort the vertices of D in order of decreasing average edge length of adjacent edges
for each vertex $v_i \epsilon D$ **do**
  if $v_i[incluster] =$ false **then**
    Call *Growcluster(v_i, D, C)*
    if C is non-trivial
      $S \leftarrow S \cup C$
    end-if
  end-if
end for

Growcluster(v, D, C)

Input: **v** - The vertex for which cluster is to be determined; **D** - Delaunay Triangulation
Output: Newly formed cluster **C** with **v** as one of the cluster members

$Q \leftarrow \phi$ {*Q is a queue*}
$C \leftarrow \phi$
$sumdist \leftarrow 0$
$C \leftarrow v$
$Q \leftarrow v$
$v[incluster] \leftarrow$ true;
$avgclusteredglen \leftarrow v[avgedglen]$

while $Q \neq \phi$ **do**
  $v \leftarrow$ Head[Q]
  for each vertex v_j adjacent to v *(in order of increasing edge length)* **do**
    if $v_j[incluster] =$ false **then**
      dist $\leftarrow$ Edge length between v and v_j
      if ($v_j[avgedglen] \leq 1.6 * avgclusteredglen$) *OR* (dist $\leq 1.6 * avgclusteredglen$) **then**
        $v_j[incluster] \leftarrow$ *true*
        if ($v_j[avgedglen] \leq 1.6 * avgclusteredglen$) **then**
          $Q \leftarrow v_j$ {*not a boundary vertex*}
        end-if
        $C \leftarrow v_j$
        $sumdist \leftarrow sumdist + dist$
        $avgclusteredglen \leftarrow sumdist / (Size(C) - 1)$
      end-if
    end-if
  end for
  Dequeue(Q)
end while

for each vertex $v_i \epsilon C$ **do**
  for each vertex v_j adjacent to v_i with $v_j[incluster] = false$ **do**
    dist $\leftarrow$ Edge length between v_i and v_j
    if dist $\leq 1.6 * avgclusteredglen$ **then**
      $C \leftarrow v_j$
      $v_j[incluster] \leftarrow$ *true*
    end if
  end for
end for

3.3. Treatment of noise in data

Noise in the data may be in the form of isolated data points or scattered throughout the data. In the former case, clusters based at these data points will not be able to grow and will be eventually eliminated as their size will be very small. However, if the noise is scattered uniformly throughout the data, our algorithm identifies it as sparse clusters. As average edge length can be stored for each cluster without any additional computation cost using our method, we can simply get rid of noise by eliminating the clusters with large value for average edge length. This is again an $O(n)$ operation.

4 Experimental Results

We recorded the performance of our algorithm on a number of datasets. We discuss the results next. We implemented the Global K-Means [6] and Greedy Elimination Method [5] locally. In the figures, we attempt to visually differentiate between the clusters using different symbols for data points and by changing the grey scale.

We compared the clustering result of our algorithm with the K-Means based approaches. Subfigures 1(a) to 1(f) illustrate the results. The K-Means based algorithms attempt to minimize the Euclidean distance of all data points to their respective cluster-centers and as a result, the clusters are roughly circular in nature. We set the number of clusters to 5 for the K-Means based approaches. Only CRYSTAL was able to detect all the clusters correctly.

Our algorithm identifies clusters of different shapes. In subfigures 2(a)-2(b), CRYSTAL is able to identify all 5 clusters. We next add noise to the data. CRYSTAL correctly identifies the noise as separate clusters (subfigures 2(c)-2(d)) . The noise can be easily removed subsequently by deleting the cluster having the largest average edge length (least density). The number of data points in a cluster can also be used in deciding whether a cluster is trivial or not. As there are no global parameters involved, our algorithm can automatically identify sparse clusters in presence of dense ones (subfigures 3(a)-3(b)). Subfigures 4(a)-4(b) show the clustering result on two closely placed dense clusters. The growth mechanism correctly stops at the boundary between the two clusters without merging them into one.

In the next series of experiments, we evaluated the capability of our algorithm to identify clusters connected by multiple bridges. In subfigures 5(a)-5(b), CRYSTAL is able to correctly identify clusters connected by multiple bridges. Subfigures 6(a)-6(d) shows the clustering results on a dataset that visually represents co-centric circles. For the centroid-based clusters, we also display the centroid of the cluster for better understanding. As evident from the figures, the output of CRYSTAL closely resembles human perception of the clusters present in the data.

Figure 8 is the clustering result of our algorithm on $t7.10k$ dataset, originally used in [4]. It was kindly provided to us by Dr. Osmar R. Zaïane, University of Alberta. For noisy datasets such as this one, we observed that reducing the threshold value of 1.6 in $dist \leq 1.6 * avgclusteredglen$ (see pseudo code) yields good results. For Figure 8, we used a threshold of 0.7. This ensures that we do not add noise to our clusters. In fact, as evident from the figure, all of the noise is successfully removed by simply eliminating clusters that have very few elements (less than a threshold). For this dataset, our algorithm is able to identify all clusters except one which gets merged (8 clusters reported). In general, we observe that a value of this threshold between 0.5 and 1.0 is good for noisy datasets. Performance of other recently proposed clustering algorithms on this dataset can be found in [3].

We experimented with our algorithm on large datasets. Figure 7 illustrates the result. The X-axis is the number of clusters in 1000 and Y-axis is the time consumed in milliseconds. The time includes the construction of the Delaunay Triangulation. The program is implemented in Java and run on a Pentium-4 3 GHz processor with 512 MB RAM. As evident from Figure 7, our algorithm is able to find clusters in a dataset of size $70,000$ in less than a minute. This compares favorably with the run-times in [2].

5 Conclusion

In this paper, we propose an $O(nlogn)$ fast and efficient clustering algorithm for two-dimensional spatial databases based on the Delaunay Triangulation. The algorithm is robust in the presence of noise and is able to successfully detect clusters of widely varying shapes and densities. In the future we would like to extend our algorithm to 3D and handle clustering in the presence of physical constraints.

6 Acknowledgements

Authors would like to acknowledge GEOIDE for continuous support of this project.

References

[1] G. Papari, N. Petkov, "Algorithm That Mimics Human Perceptual Grouping of Dot Patterns", Lecture Notes in Computer Science, 3704, 2005, pp. 497-506.

[2] Vladimir Estivill-Castro, Ickjai Lee, "AUTOCLUST: Automatic Clustering via Boundary Extraction for Mining Massive Point-Data Sets", Fifth International Conference on Geocomputation, 2000.

[3] Osmar R. Zaïane, Andrew Foss, Chi-Hoon Lee, Weinan Wang, "On Data Clustering Analysis: Scalability, Constraints and Validation", Advances in Knowledge Discovery and Data Mining, Springer-Verlag, 2002.

[4] George Karypis, Eui-Hong Han, Vipin Kumar, "CHAMELEON: A Hierarchical Clustering Algorithm using dynamic modeling", IEEE Computer, 32(8), pp. 68-75.

[5] Z.S.H. Chan, N. Kasabov, "Efficient global clustering using the Greedy Elimination Method", Electronics Letters, 40(25), 2004.

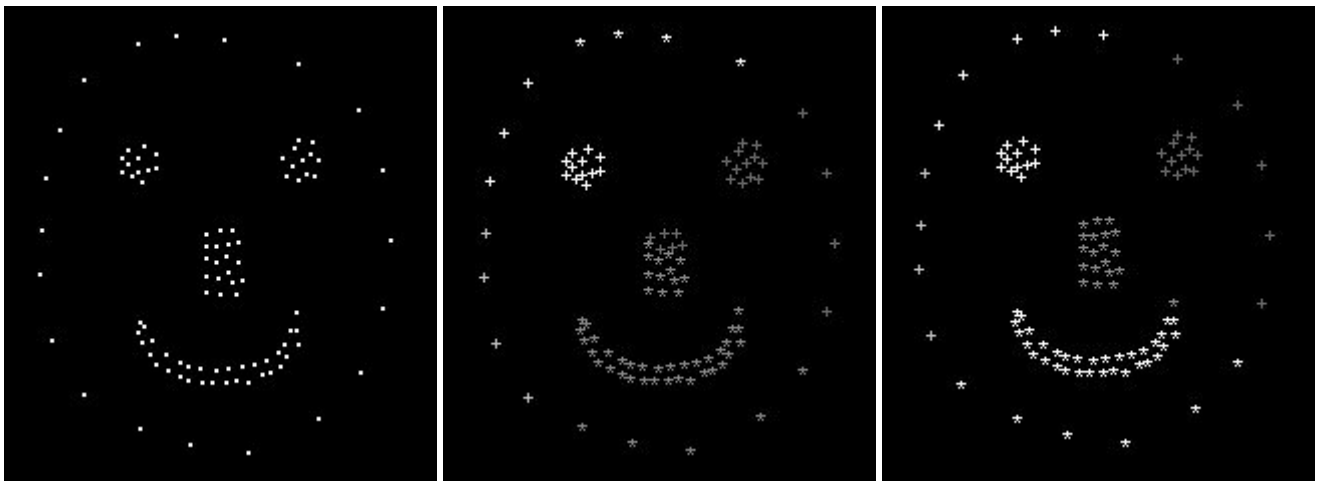
[6] Aristidis Likas, Nikos Vlassis, Jakob J. Verbeek, "The global k-means clustering algorithm", Pattern Recognition, 36(2), 2003, pp. 451-461.

[7] Ying Xu, Victor Olman, Dong Xu, "Minimum Spanning Trees for Gene Expression Data Clustering", Genome Informatics, 12, 2001, pp. 24-33.

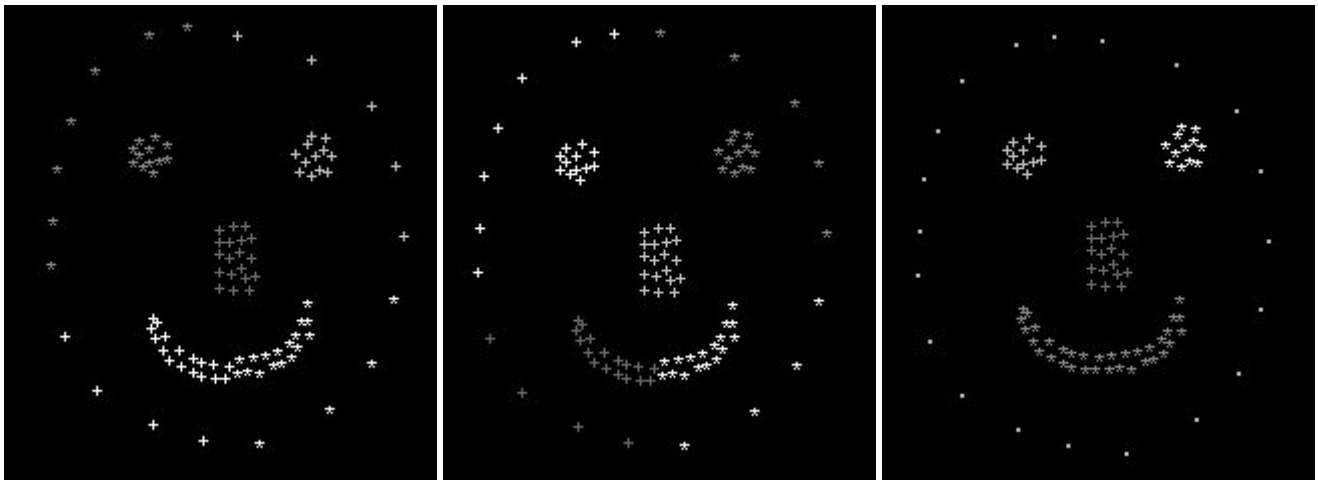
[8] C. Eldershaw, M. Hegland, "Cluster Analysis using Triangulation", Computational Techniques and Applications CTAC97, World Scientific, Singapore, 1997, pp. 201-208.

[9] Mir Abolfazl Mostafavi, Christopher Gold, Maciej Dakowicz, "Delete and insert operations in Voronoi/Delaunay methods and applications", Computers & Geosciences, 29(4), 2003, pp. 523-530.

[10] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, "Spatial Tessellations: Concepts and Applications of Voronoi Diagrams", John Wiley, 1992.

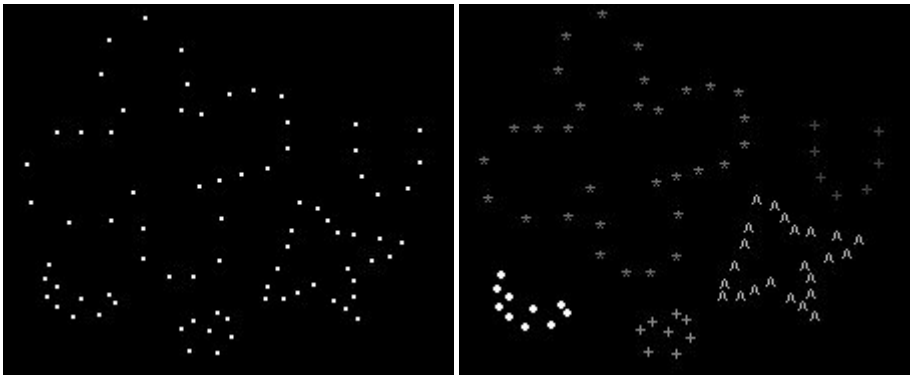


(a) Original data set (b) Standard K-Means (K = 5) (c) K-Means initialized with Kd-tree (K = 5)

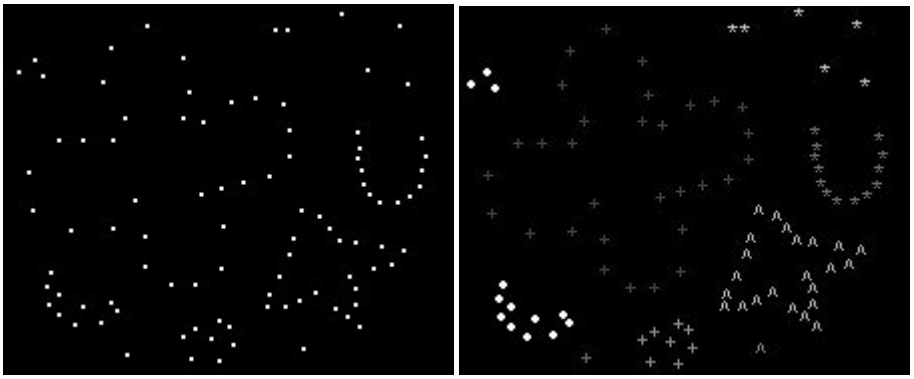


(d) Global K-Means (K = 5) (e) Greedy Elimination Method (K = 5) (f) CRYSTAL

Figure 1. Comparison with K-Means based approaches.

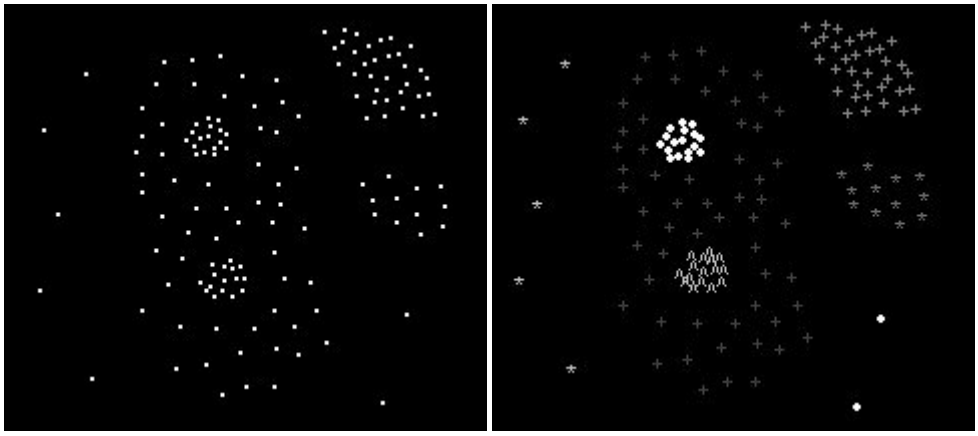


(a) Original data set (b) CRYSTAL



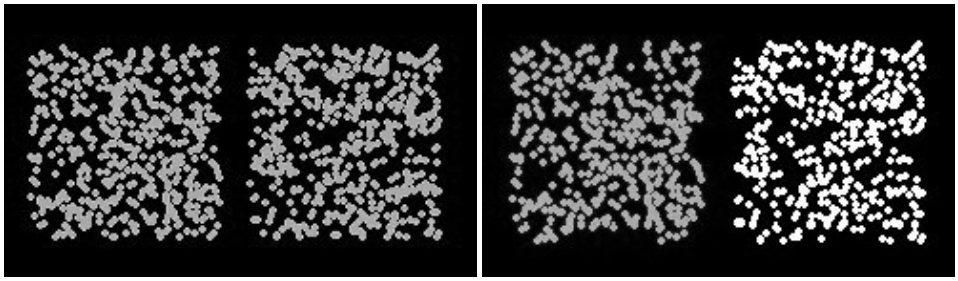
(c) Noise added (d) CRYSTAL

Figure 2. Clusters of different shapes.



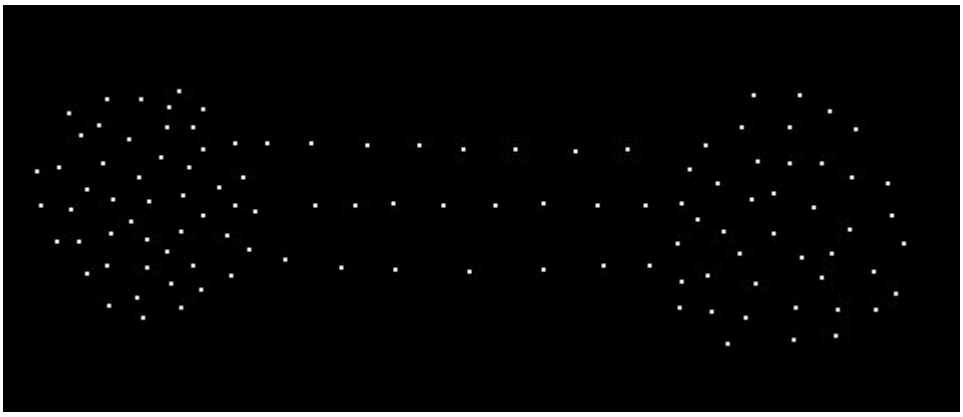
(a) Original data set (b) CRYSTAL

Figure 3. Sparse clusters in presence of dense ones.

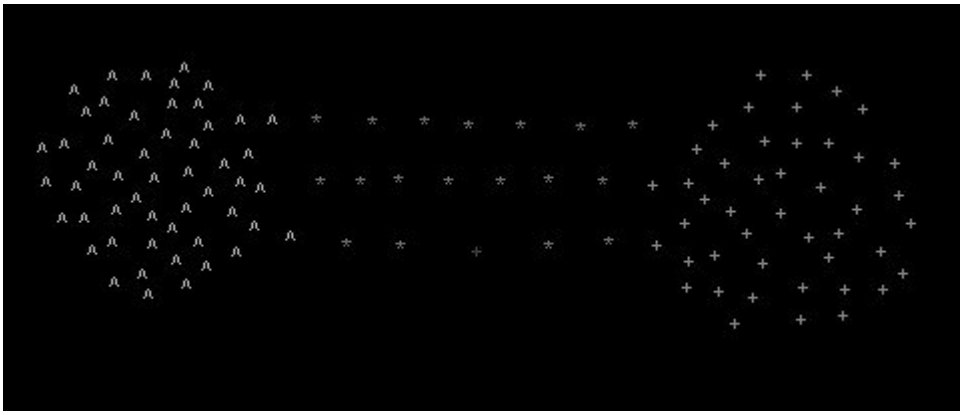


(a) Original data set (b) CRYSTAL

Figure 4. Dense clusters close to each other.

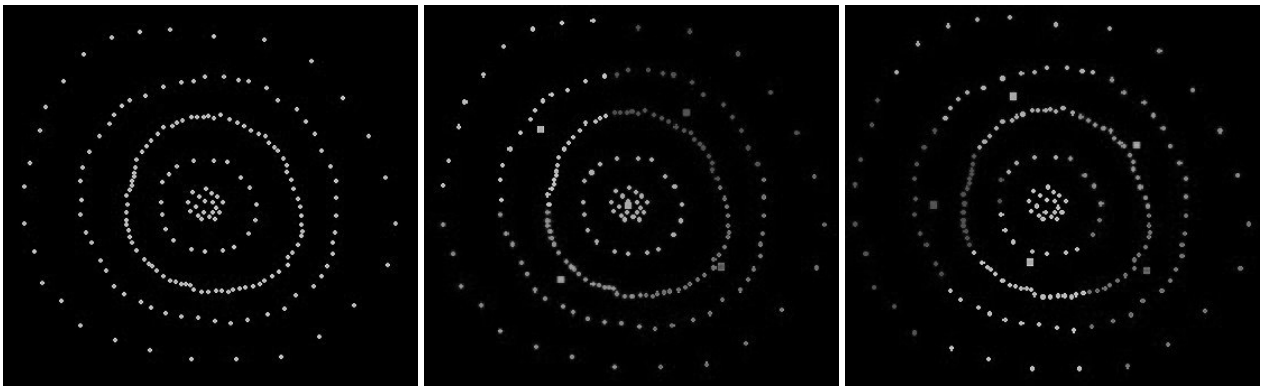


(a) Original data set

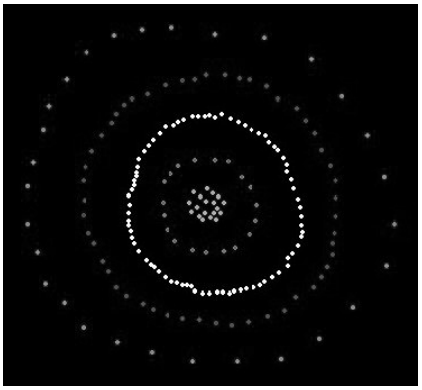


(b) CRYSTAL

Figure 5. Clusters connected by bridges.



(a) Original data set (b) K-Means (K = 5) (c) Greedy Elimination Method (K = 5)



(d) CRYSTAL

Figure 6. Co-centric clusters.

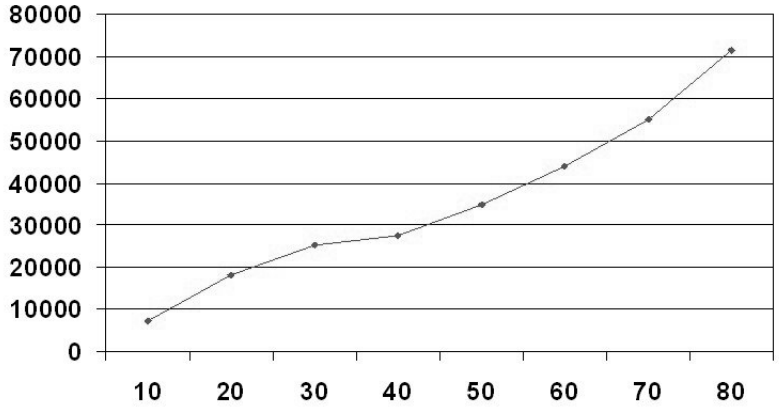
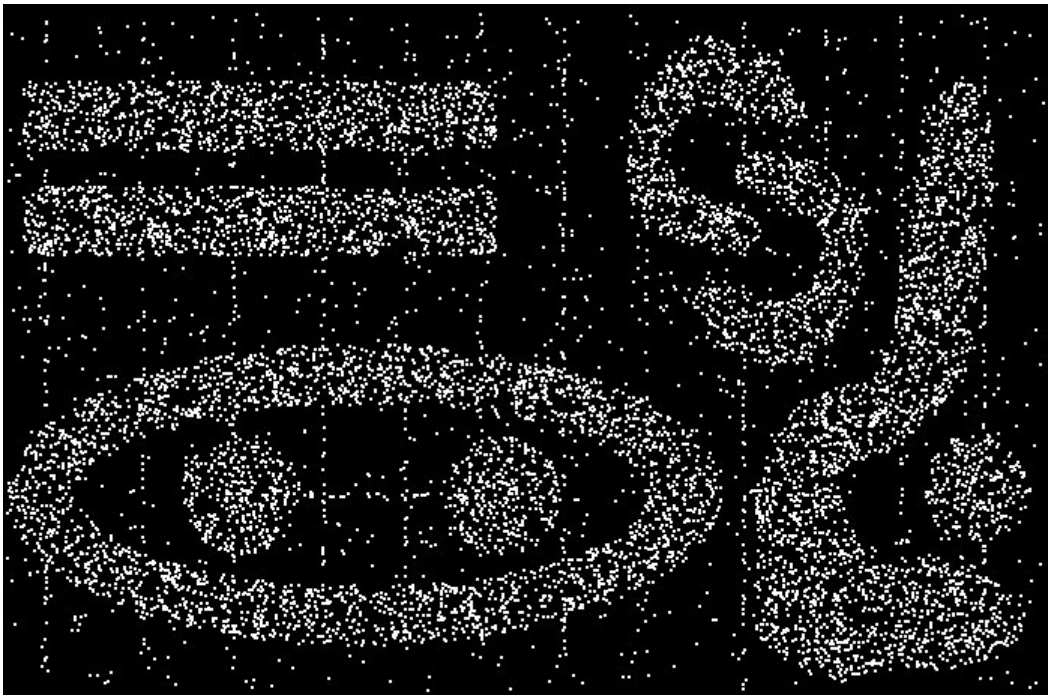
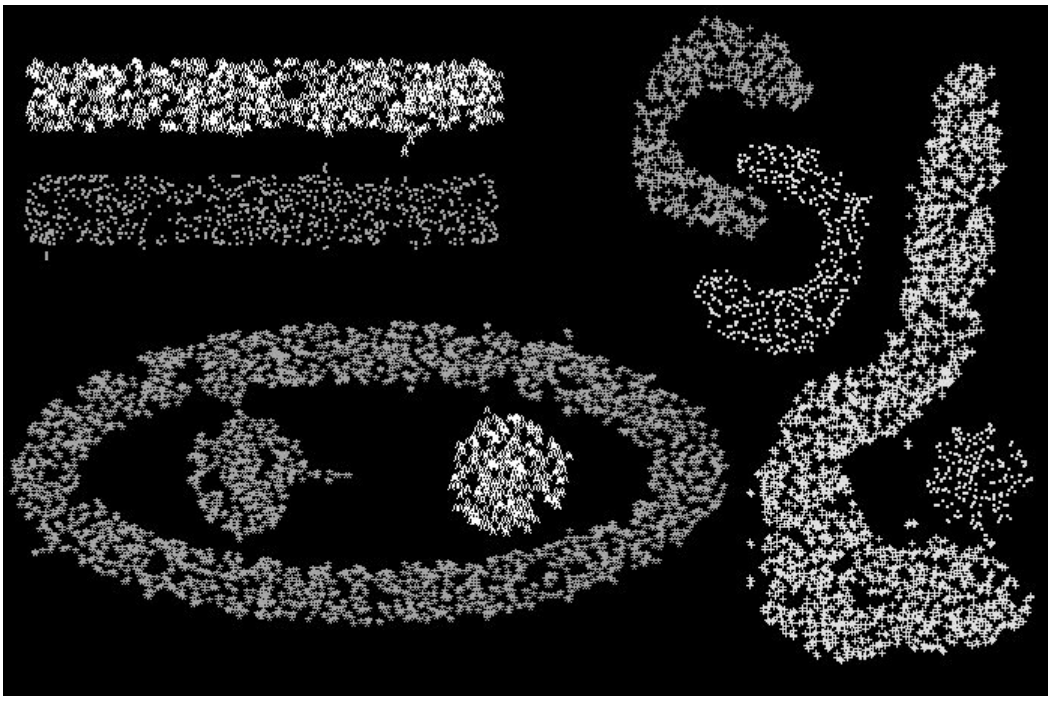


Figure 7. CRYSTAL: Cluster size (1000) Vs Time consumed (ms)



(a) t7.10k.dat data set



(b) CRYSTAL

Figure 8. Clustering result on t7.10k.dat.