DBSCAN

~ Implement the DBSCAN algorithm for clustering the data ~

Professor: Sang-Wook Kim

Department of Computer Science

2012003716 Kwangil Cho

2017. 05. 23

0. Introduction

Density-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise (**DBSCAN**) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996. **It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors)**, marking as outlier points that lie alone in low-density regions (whose nearest neighbors are too far away).

1. Objectives and Environment

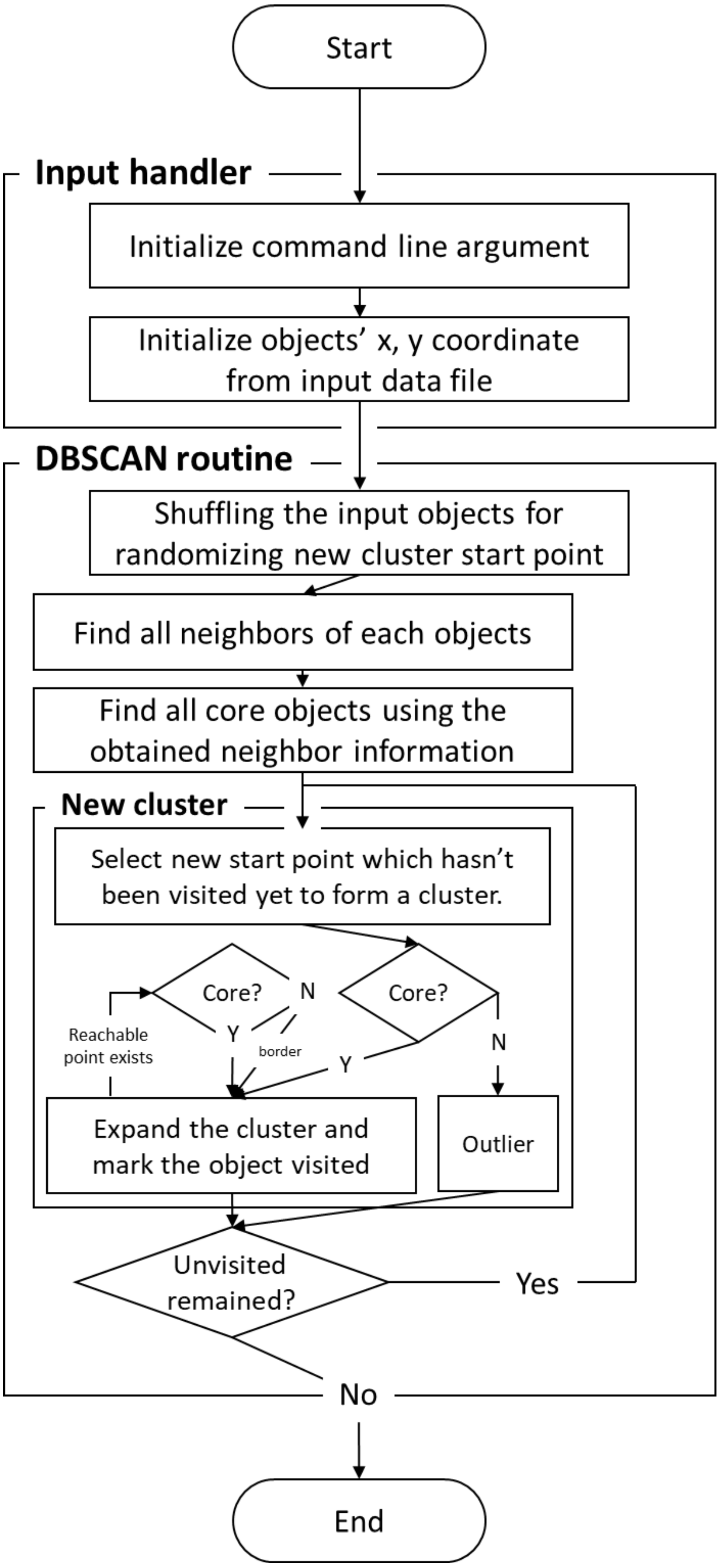
The goal of this project is to cluster the data by DBSCAN algorithm which performs clustering with respect to density of data. After clustering, I plot the clustered data and measure the accuracy of the algorithm.

I have been working on the following environment:

- Operating System: Ubuntu 14.05 LTS 64-bit
- Compilation: g++ compiler
- Language: C/C++ with C++ standard 11
- Source code editor: Vim
- Source code version control: Git
- Plot application: Gnuplot
- Followed HYU coding convention faithfully

2. Program Structures

The flowchart of the program is as follows:



Start
Input handler
Initialize command line argument
Initialize objects' x, y coordinate from input data file
DBSCAN routine
Shuffling the input objects for randomizing new cluster start point
Find all neighbors of each objects
Find all core objects using the obtained neighbor information
New cluster
Select new start point which hasn't been visited yet to form a cluster.
Core?
N
Y
border
Reachable point exists
Core?
Y
N
Expand the cluster and mark the object visited
Outlier
Unvisited remained?
Yes
No
End

(0) The program expects 5 arguments (executable file, input data file, number of clusters, epsilon which is the maximum radius of the neighborhood, Minpts which is the minimum number of points in an Eps-neighborhood of a given point) for executing DBSCAN algorithm. When the arguments are successfully ready, input handling routine works. First, input handler sets configuration of cluster. Number of clusters, epsilon value, and Minpts are set here. Second, handler opens input file for getting input data and save the input file name for future output data.

(1) When the environment settings are done, handler reads all of object information line by line from the input data file. At this time, object data which has x and y coordinate is made and entered into a global list by its object id. The input handling procedure is done.

(2) Before executing DBSCAN algorithm routine, there are some pre-processing steps. First, at each iteration, an object is randomly picked to create a new cluster. Therefore, the unvisited object vector which contains all object ids should be shuffled. Second, find all neighborhoods which is reachable points from each point and save the neighbors' id. Third, find all core points with respect to the number of neighbors of each point.

(3) After pre-processing steps, DBSCAN routine runs repeatedly until there is no unvisited object remained. First, pick the new clustering start object from unvisited object vector. If it is a core point, the new cluster is starting to form and expand the cluster to its neighborhoods continuously. If it's not a core, it means that the number of neighbors in Eps radius is less than MinPts. Then it is classified as an outlier.

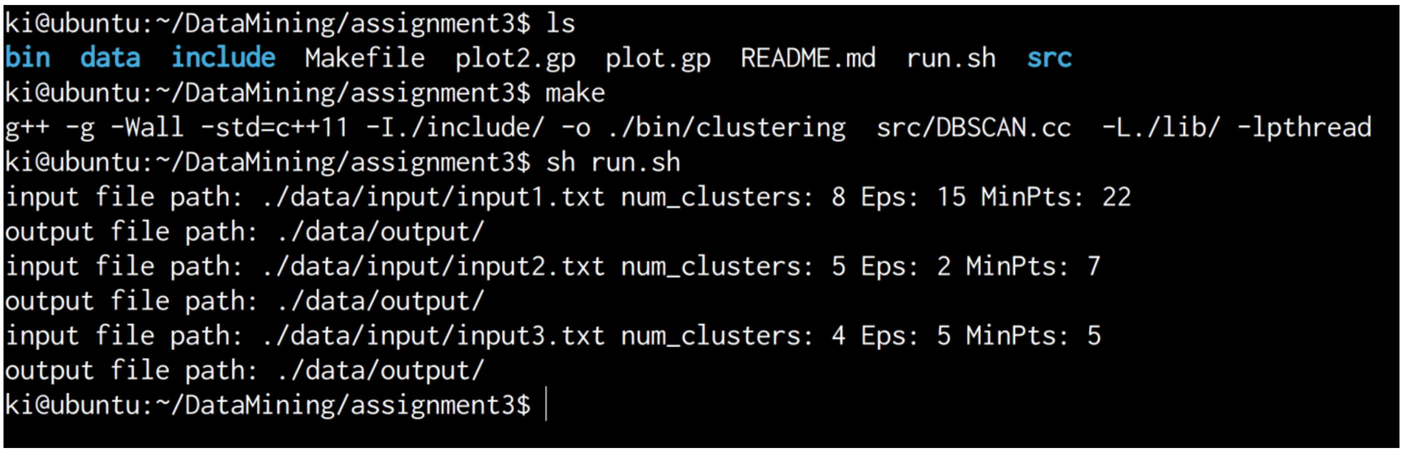
(4) There are two approaches to expand the cluster. Depth First Search and Breadth First Search. DFS approach searches the core point in recursive way. BFS approach finds core points in spreading way. While expanding the cluster, if reached point is not a core point, in other words, it is not defined OR already marked as outlier, it becomes border point.

(5) After DBSCAN, the algorithm might found more clusters than expected. Then, for accuracy optimization, cluster reorganization is needed. If the expected number of cluster is smaller than DBSCAN found, do the reorganize routine. First, sort the clusters in descending order. Second, calculate how many cluster should merge. Third, from the smallest cluster, merge the cluster into the nearest cluster. Here 'nearest cluster' means the closest distance between centroids. After merging, change the cluster from old one to new one.

(6) If post-process done, print the cluster data to output path. There are 4 types of output files. The original coordinate data, core point coordinate data, coordinate data for each cluster and object ids for each cluster.

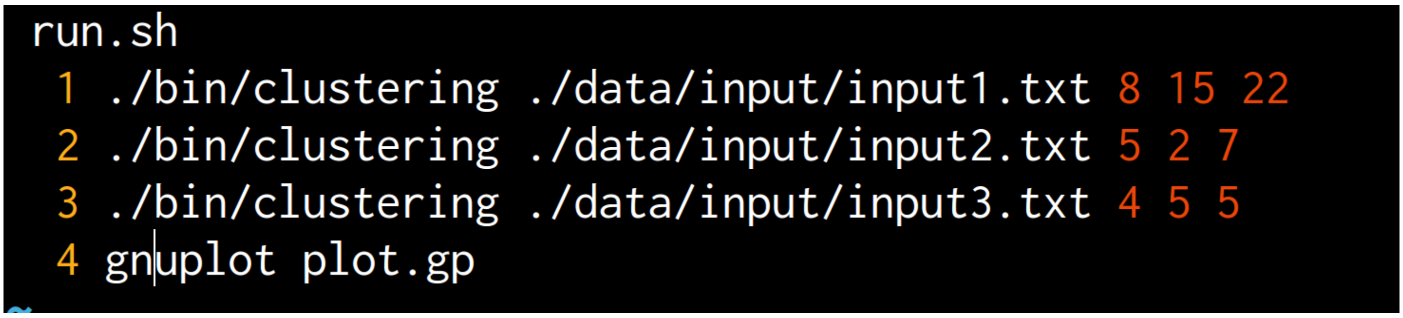
(7) When a prepared script for automatic test runs, 3 test cases will be run and plot the result data.

3. Execution



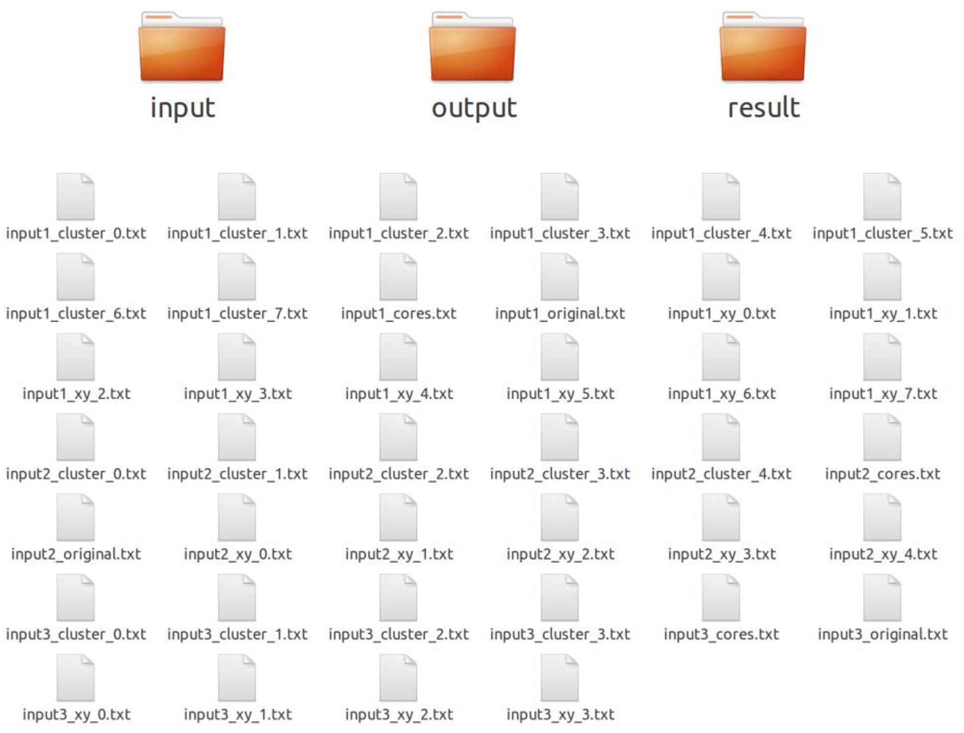
```
ki@ubuntu:~/DataMining/assignment3$ ls
bin  data  include  Makefile  plot2.gp  plot.gp  README.md  run.sh  src
ki@ubuntu:~/DataMining/assignment3$ make
g++ -g -Wall -std=c++11 -I./include/ -o ./bin/clustering  src/DBSCAN.cc  -L./lib/ -lpthread
ki@ubuntu:~/DataMining/assignment3$ sh run.sh
input file path: ./data/input/input1.txt num_clusters: 8 Eps: 15 MinPts: 22
output file path: ./data/output/
input file path: ./data/input/input2.txt num_clusters: 5 Eps: 2 MinPts: 7
output file path: ./data/output/
input file path: ./data/input/input3.txt num_clusters: 4 Eps: 5 MinPts: 5
output file path: ./data/output/
ki@ubuntu:~/DataMining/assignment3$
```

(1) Build and execute the program with prepared test script.



```
run.sh
./bin/clustering ./data/input/input1.txt 8 15 22
./bin/clustering ./data/input/input2.txt 5 2 7
./bin/clustering ./data/input/input3.txt 4 5 5
gnuplot plot.gp
```

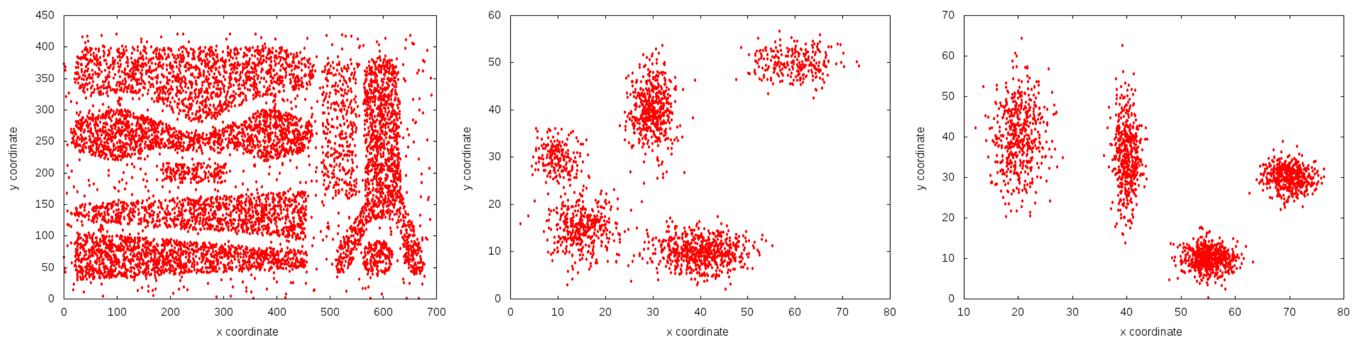
(2) Test script runs 3 tests iteratively with given appropriate arguments and plot the results all.



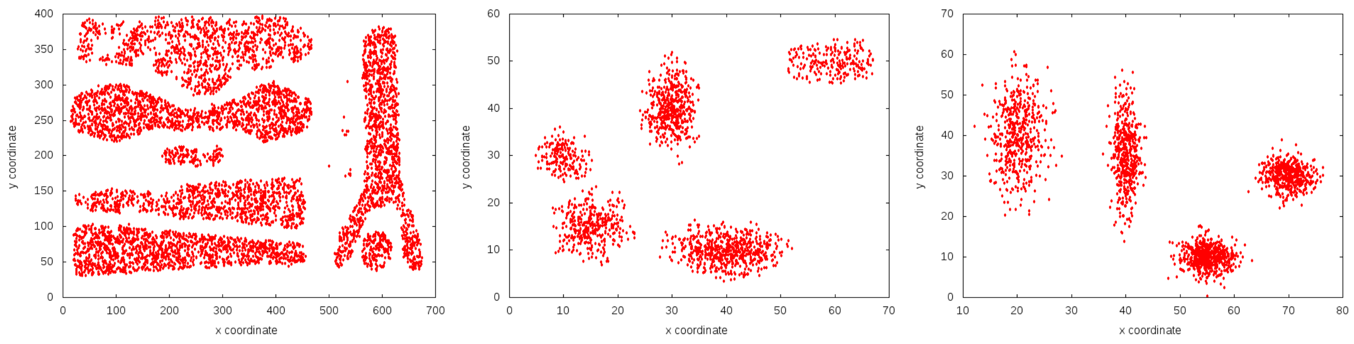
```
input    output    result

input1_cluster_0.txt  input1_cluster_1.txt  input1_cluster_2.txt  input1_cluster_3.txt  input1_cluster_4.txt  input1_cluster_5.txt
input1_cluster_6.txt  input1_cluster_7.txt  input1_cores.txt  input1_original.txt  input1_xy_0.txt  input1_xy_1.txt
input1_xy_2.txt  input1_xy_3.txt  input1_xy_4.txt  input1_xy_5.txt  input1_xy_6.txt  input1_xy_7.txt
input2_cluster_0.txt  input2_cluster_1.txt  input2_cluster_2.txt  input2_cluster_3.txt  input2_cluster_4.txt  input2_cores.txt
input2_original.txt  input2_xy_0.txt  input2_xy_1.txt  input2_xy_2.txt  input2_xy_3.txt  input2_xy_4.txt
input3_cluster_0.txt  input3_cluster_1.txt  input3_cluster_2.txt  input3_cluster_3.txt  input3_cores.txt  input3_original.txt
input3_xy_0.txt  input3_xy_1.txt  input3_xy_2.txt  input3_xy_3.txt
```

(3) There are 3 data directories. Input directory contains the original data before clustering. Output directory contains all kind of processed data after clustering. There are 4 types of output data. Every data files in output directory are distinguished by prefix that indicates the original input data file name. There are original input coordinates file of which postfix is '_original.txt', core points file with postfix '_cores.txt', clustered x, y coordinates file with postfix '_xy_#.txt', and clustered object ids with

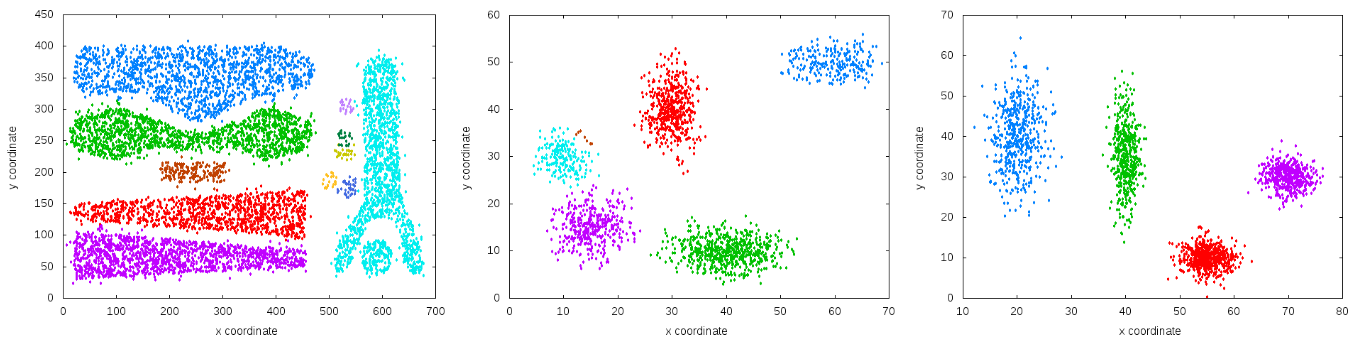
postfix '_cluster_#.txt'. Result directory contains result figures which is generated from plotting the output data.



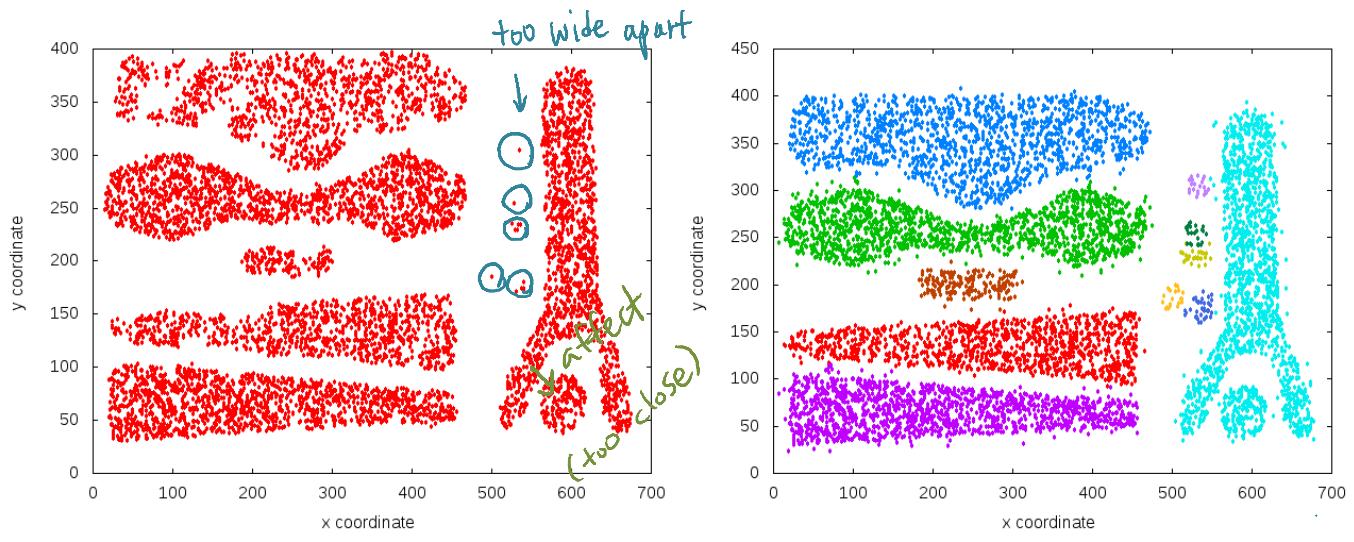
(4) By using the original coordinate files, plot the original objects with gnuplot.



(5) By using the core points files, plot the core objects of each data with gnuplot.

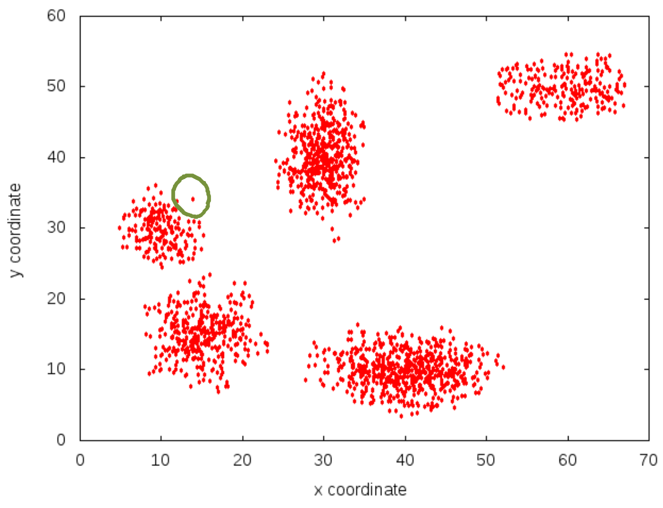


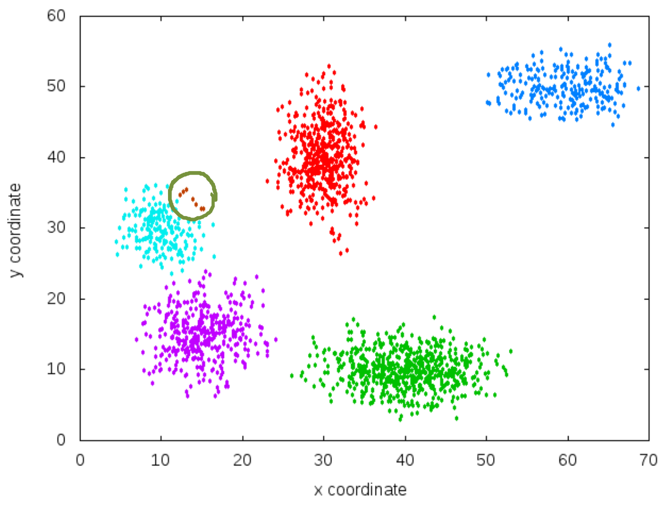
(6) By using coordinates of each cluster files, plot the clustered objects with gnuplot.



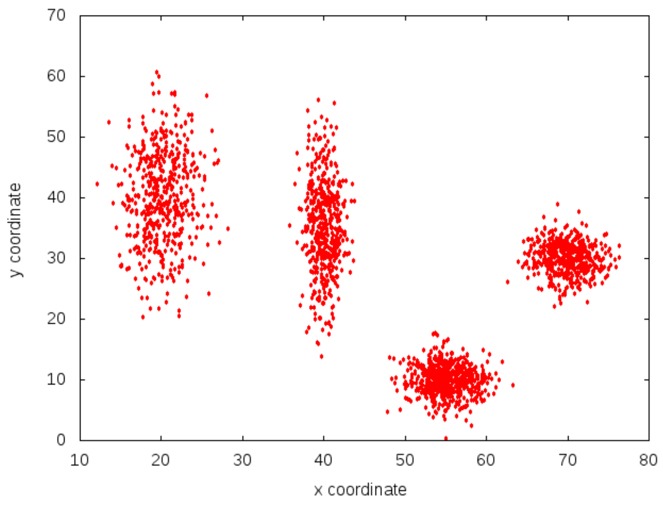
(7) One thing to note from this result is that there are more clusters than expected in test #1, and clusters

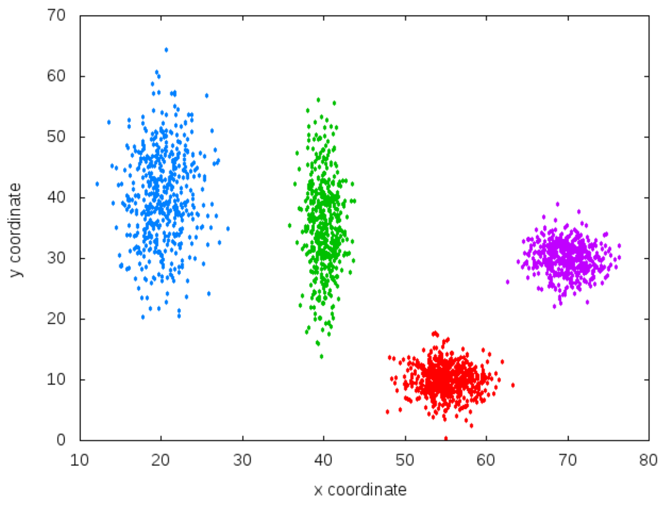
that should not be clustered together. The reason is that the core points shown are too wide apart or too close.





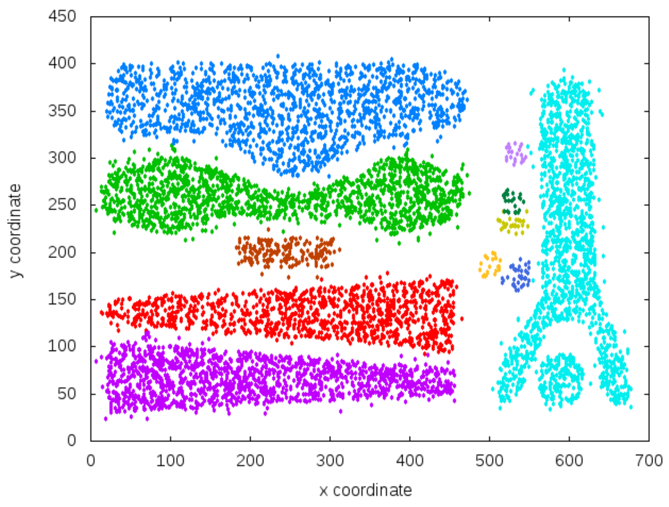
(8) Overall, the clustering was successful in test #2, but the slightly spaced core point formed a new unnecessary cluster.

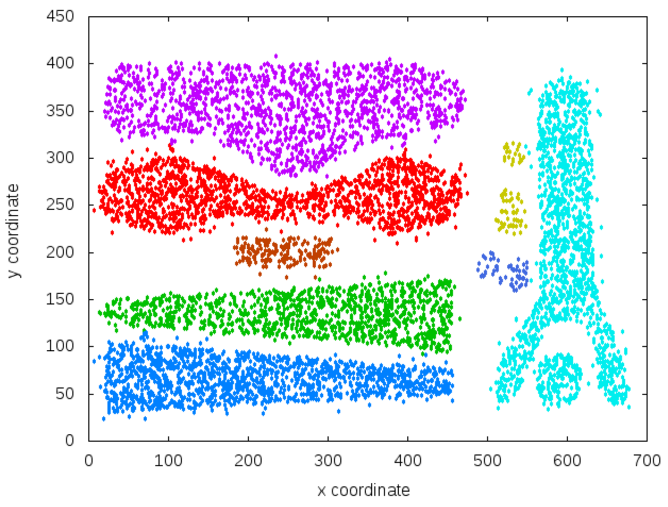




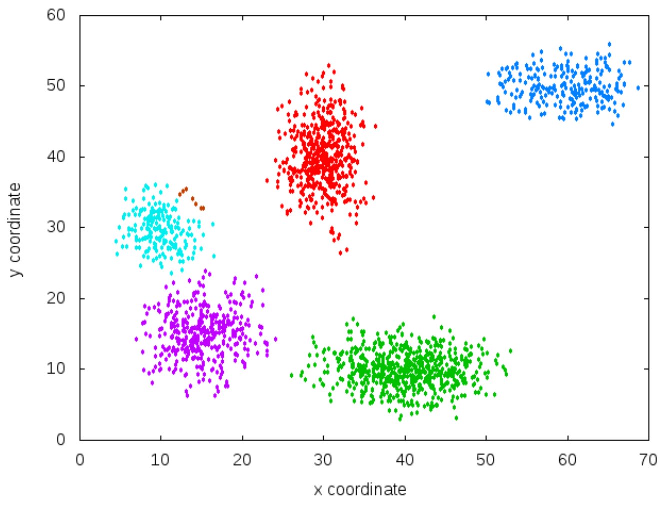
(9) The clustering has been almost successfully in test #3. Because there was no ambiguous core point.

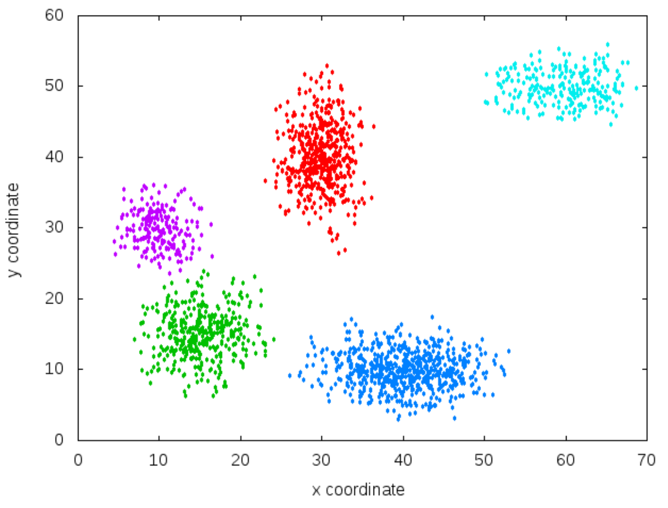
(10) Until above, there are the results of pure DBSCAN algorithm which just uses 2 arguments (epsilon and Minpts for forming cluster). However, in this project, I have another extra argument that specifies the expected number of clusters. I could try to optimize (merge or split the clusters) the clusters by using this argument.



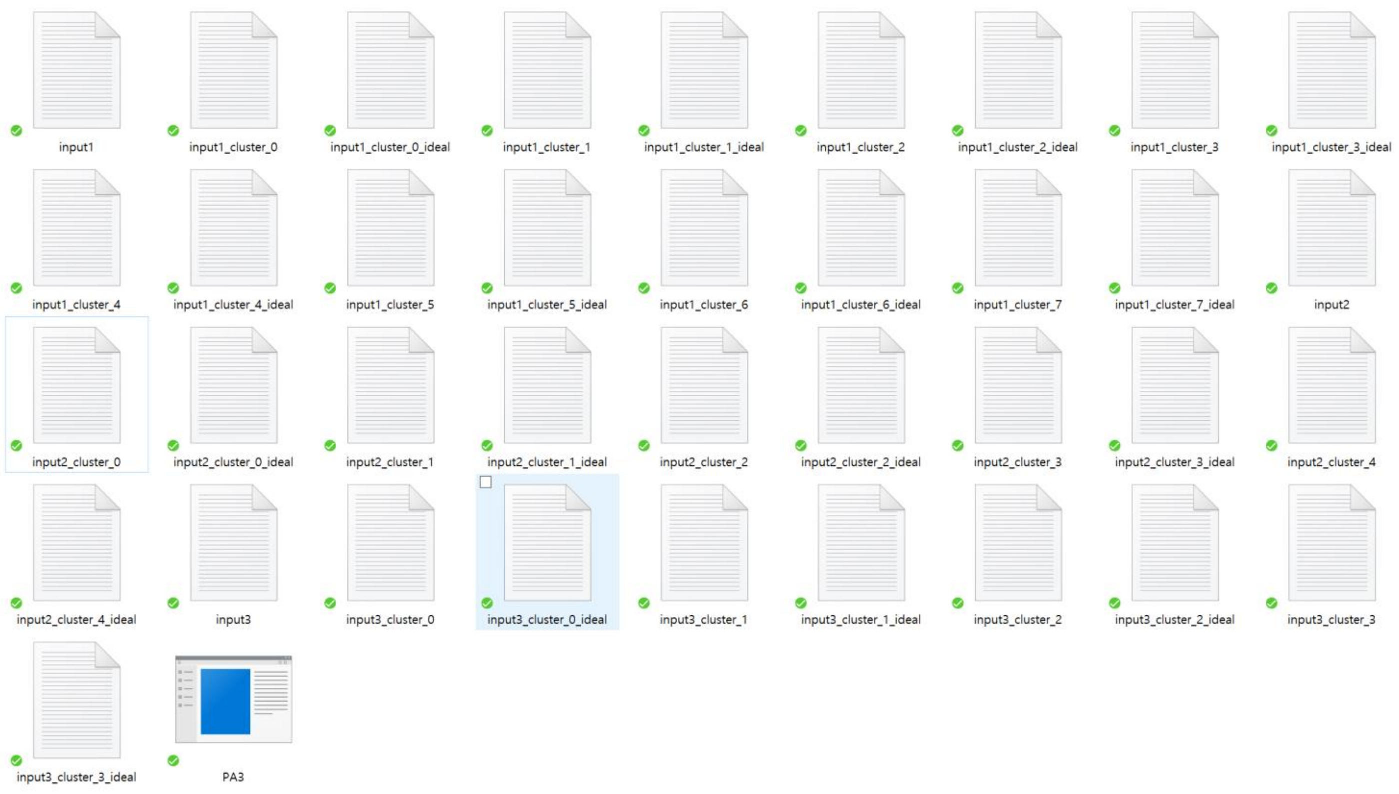


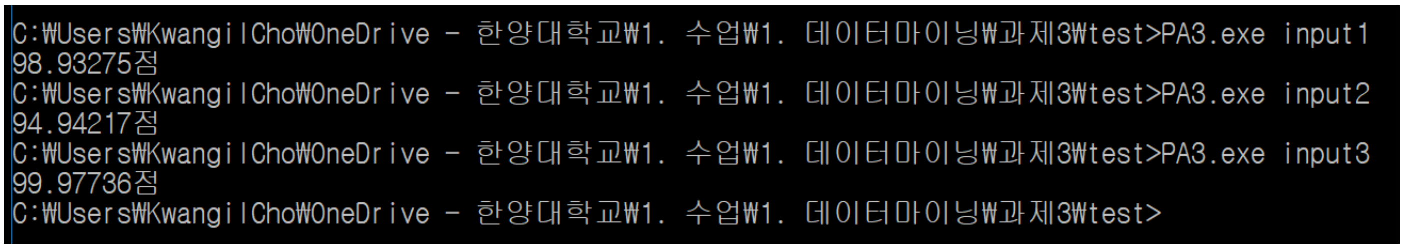
(11) In test #1, there were clusters that had been discovered more than I had wanted. Therefore, I reorganize the clusters with distance-base criteria. First, sort the clusters in descending order by its number of objects. Then, calculate the centroid of each cluster and merge the smallest one into the nearest cluster until the total number of cluster becomes what I expected. The result is still not perfect, this reorganization can fix the error and improve the accuracy of clustering.





(12) When I used the proposed cluster reorganization algorithm above for test #2, It merged a miss placed cluster successfully.




```
C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제3₩test>PA3.exe input1
98.93275점
C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제3₩test>PA3.exe input2
94.94217점
C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제3₩test>PA3.exe input3
99.97736점
C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제3₩test>
```

(13) Test with TA's testing program with 3 test clustered files. The clustering successfully passes the test with the expected accuracy.

4. Epilogue

Through this project, I was able to learn how to cluster the numerical data by using density based clustering method, DBSCAN. It was very interesting job to design and implement DBSCAN, and I was trying to make it better. Although the assignment was just implementing DBSCAN, I applied optimization algorithm additionally. It was also very fascinating to visualize the result data because I really wondered how clusters are formed. Thanks to professor and assistant who prepared for class and practices.

5. Reference

- Wikipedia: DBSCAN (https://en.wikipedia.org/wiki/DBSCAN)