HACETTEPE UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

**BBM 425**

# INTERNSHIP REPORT

**Deniz Ece AKTAŞ**
**21626901**

**Performed at**

**Hacettepe University Computer Engineering Department**

**10/08/20 – 18/09/20**
**30 Work Days**

## TABLE OF CONTENTS

# 1 Introduction

I had to do my second internship with Dr. Engin DEMİR because of the COVID-19 pandemic so I am counting Hacettepe University as the company I did my internship in and I also take the department I did my internship in as the computer science department of Hacettepe University. Hacettepe University aims to be a university that leads to change and development.

Because of the pandemic, the companies I applied for internships either didn’t take any interns or decreased the number of interns so I couldn’t find an internship but the computer science department in Hacettepe University gave students an opportunity to be able to do their internship in collaboration with some of the faculty members. I was able to get one of the internships the department provided.

During my internship, I worked on two data clustering algorithms, the st-dbscan algorithm, and the CutESC algorithm, with the help of two academic articles Dr.Engin Demir provided I researched, implemented found suitable data, and tested the data. I had never worked with data before therefor this internship helped to improve my database management skills.

# 2 Company Information

## 2.1 About the company

In Hacettepe University, on July 8, 1967, educational activities began in the Faculties of Medicine, Health Sciences, Sciences, and Social Sciences. The School of Home Economics was established in 1968, and the Schools of Pharmacy and Dentistry were transformed into faculties in 1971. Hacettepe University, which was further expanded over time with the establishment of new departments and faculties, built its second campus on an area of 1.500 ha. in Beytepe District, 20 km. away from the Central Campus. [1]

Hacettepe University continues to be one of the leading universities in Turkey and continues the educational activities with many faculties and institutes. The vision of Hacettepe University is to become a leading university where the individuals are proud to be a member of both in the national and international arena, leading the change and development.[2] The keyword for Hacettepe University’s basic values are being farsighted, respect, aesthetics, and participation.

## 2.2 About your department

Because of the pandemic, I did my internship with Dr.Engin Demir so I will describe the computer science department of Hacettepe University as my department. The computer science department researches many different areas like software engineering, computer vision, database management systems, etc.  I worked on database management and did my internship as a researcher and programmer.

## 2.3 About the hardware and software systems

I was using my computer at the internship. As for the coding program, I used Pycharm with Anaconda and Jupyter Notebook and used python as the coding language.

## 2.4 About your supervisor

- **Name:** Dr. Engin DEMİR
- **Mail:** engindemir@cs.hacettepe.edu.tr
- **Tel:** +90 (312) 297 7500/ 128
- **Education:** Bilkent University –Computer Engineering and Information Science(1999)
- **Work Address:** Bilgisayar Mühendisliği Bölümü, Hacettepe Üniversitesi, Beytepe 06800 Çankaya/ ANKARA

# 3 Work Done

My internship's main goal was to implement two different data clustering and outlier detection algorithms from two academic articles given by Dr.Engin Demir, Spatio-temporal outlier detection algorithms based on computing behavioral outliers factor[3] academic article, CutESC: Cutting edge spatial clustering technique based on proximity graphs[4] academic article, and detecting the advantages and disadvantages of each algorithm. I was given "work packages" for each week which was basically the descriptions of the steps I would have to do during my internship. These "work packages" were divided by work by week for example first week was reading and research, the two weeks after that was coding, fourth week was data collecting and rearrangement, the week after was code verification and testing and the last week was analysis and improvements as shown on the Fig 1 below.

| | |
|---|---|
| Week 1 | Reading and research |
| Week 2 | Coding |
| Week 3 | Coding |
| Week 4 | Data collecting and rearrangement |
| Week 5 | Code verification and testing |
| Week 6 | Analysis and improvements |

Fig 1.Work Packages

### Reading and Research

During the first week of my internship, which was the first step of the "work package", I focused on going over the systems I would be working with, Python on PyCharm, Anaconda, and Jupyter Notebook. At that time other than the BBM371 Data Management class I didn't know anything about data management systems so especially the first couple of days I researched database management systems, some basic SQL.

After getting used to some of the basic database management knowledge I started to research the first algorithm that I would implement which is the st-dbscan algorithm. In my research about Spatio-temporal clustering algorithms I encountered other algorithms like the LDSCAN algorithm and dbscan algorithm but in the Spatio-temporal outlier detection algorithms based on computing behavioral outliers factor academic article it was written that st-dbscan detects the most accurate outlier numbers so I implemented the st-dbscan algorithm.

During the first week which was my research period I also researched, CutESC: Cutting edge spatial clustering technique based on proximity graphs academic article but there weren't many resources other than the academic article which was given to me by my supervisor so I tried the specific parts of the algorithm separately like Delaunay graph and Gabriel graph and so on. Also during the research week, I looked into and got used to scikit-learn, NumPy, and pandas packages which I had to use to implement both of the codes. Even though I was in reading and research part of the work package I tried to start implementing the st-dbscan algorithm but I did not get very far so I decided to try out the scikit-learn ready to use dbscan and try to find anomalies with a simple code so I could understand finding anomalies from clusters. This code generated random points and clustered them and showed the outliers with red dots and clusters with blue dots on a plot. The code fragment is shown on Fig 19 and the resulting plot is shown in Fig 20.

On the last day of my first week, I started to write the st-dbscan algorithm which is an algorithm that takes spatial and temporal thresholds and the minimum number of neighbors that should be in a cluster. The algorithm marks the outlier data as -1 class and clusters as positive integers. To do this st-dbscan algorithm goes over the data and if the point doesn’t belong to a cluster retrieves the neighboring points of the selected data and neighboring points are found by checking if the points are suitable for spatial and temporal thresholds if so the points are checked again if they are appropriate to be called a cluster by checking if they are bigger or smaller than the minimum number of neighbors threshold. If the points are less than the threshold they are marked as clusters if not the algorithm tries to make a new cluster by calling the neighboring points and checking again if they fit all of the thresholds. I wrote the parts for clustering and outlier marking parts of the code, fragment from code is shown on Fig 21, but couldn’t implement finding the neighbors section of the code in my first week’s last day.

## Coding

During the second week of my internship, which was the coding  step of the “work package”, I researched and wrote getting neighbors function and while doing this I also researched and wrote a function that finds the distance of two latitude and longitudes. The neighbors function determines the neighbors by checking if the location is smaller or equal to the spatial threshold and also by checking the time of the variable and if that time is inside the time threshold. Code fragment from neighbors function is shown on Fig 22 and fragment from disc_calc which finds the distance between two longitudes and latitudes is shown on Fig 23.

Afterward, I started to write code for the CutESC algorithm given in CutESC: Cutting edge spatial clustering technique based on proximity graphs academic article. CutESC algorithm works by, just like the st-dbscan algorithm, marking the outliers as -1 and other clusters as positive integers. The algorithm does this by taking data points and constructing a sparse graph using Delaunay triangulation then forming a Gabriel graph from the Delaunay triangulation then finding globally long edges, removing them and finding locally long edges and removing them and finding other locally long edges and getting rid of them and finally assigning cluster labels by looking into the strong and weak connections of points.

I read the CutESC academic article from scratch once again to prepare. I started with writing the Delaunay triangular part of the algorithm. To do this part Delaunay triangulation can be made with scipy.spatial.Delaunay function, the code fragment for Delaunay graph is shown on Fig 24. Then I focused on the Gabriel graph part of the code. Gabriel graph is a subgraph of Delaunay triangular. The connection is Gabriel graph edge if points can be connected by a circle and that circle doesn't have any other points. To do this center and midpoint of each edge is found and then search the nearest neighbors and if those points are in the circle's distance then those points are not a Gabriel edge. The code fragment for Gabriel graph is shown on Fig 25.

To find the globally long edges the mean length of incident edges of vertex Pi, the mean length of all edges in the Gabriel graph, and the standard derivation of the mean length of edges in the first order of neighbors. Mean(Pi) is found by the sum of the length of edges in the first order of the neighborhood divided by the number of edges that are directly connected to Pi. Mean(GG)is calculated by the sum of edges' length's sum in the edges in Gabriel graph divided by the number of edges in Gabriel graph. The standard derivation is found by the sum of mean(Pi) minus mean(GG) squared in the number of vertices in Gabriel graph.

Global cut can be found by the sum od mean(GG)and standard derivation multiplied by mean(GG)divided by mean(Pi).  In the end, if the length of an incident edge of Pi is longer than or equal to the calculated global cut value, it is removed. The code fragment for the function that does these is shown on the Fig 26. After finishing these steps the second week of my internship was over.

In my third week of internship, which was still the coding step of the “work package”, I started by looking into ıdentifying locally long edges part of the CutESC algorithm. In this part what is needed to find these edges are the mean length of edges in the second-order neighborhood of vertex Pi in a subgraph of the Gabriel graph; and the standard derivation of the length of all edges that are directly connected to vertex Pi which is a member of the sub-graph of Gabriel graph. Mean(Pi) is found by the sum of second-order neighborhoods' edges' Euclidean length and then that number is divided by the number of edges that are directly connected to Pi.

Mean(Gk) is found by the sum of the edges in the sub-graph's mean(pi). Standard derivation of the length of incident edges of vertex Pi in Gk is expressed by the sum of the square of second-order neighborhoods' edges' Euclidean length minus the mean(Gk) and then this number is divided by second-order neighborhoods' edges' number minus one and then that number’s square root is taken and this gives the standard derivation. The code fragment for the findLoc function that does these is shown on the Fig 27.

I continued from removing the locally long edges from the graph part. In the given formula for this section, the edges are detected if the length of the edge which is in the first order of the neighborhood's graph's edges, should be bigger or equal to the sum of the mean(Gk)and the multiplication of alpha, standard derivation of Pi and mean(Gk) divided by length of the edge. Alpha is a parameter that should be between 0 and 1 and it is set to 1 by default. Then I started to ıdentifying other locally long edges part of the algorithm and this part is similar to finding locally long edges section. What is needed is the mean length of edges in the second-order neighborhood of vertex Pi and the standard derivation of the length of all edges that are directly connected to vertex Pi. And the long edges are found if the length of the edge is greater or equal to the sum of the mean(Pi) and multiplication of beta, standard derivation of Pi and mean(Pi) divided by length of the edge. Beta like alpha is a control parameter that should be between 0 and 1 and is set to 1 by default. The code fragment for the findOtherLocal function that does these is shown on the Fig 28.

Afterward, I continued the CutESC algorithm from assigning data points to the clusters section which is shown on the Fig 29. In this part, the main goal is to assign the data points based on the remaining connections. After the edge detections and removals, there are still data points that don't belong to anywhere. In this section, these data points are separated by being a part of the clusters or being an outlier. To finding this out, the elbow method is used. The method is to form clusters from strongly connected components. Then turn these clusters into a histogram in descending order and find the elbow point that is on a convex curve in decreasing direction. Elbow point will be the threshold that is used to find outliers.In the last days of my third week, I rearranged the CutEsc algorithm's code. I divided the parts into functions so it is easier to understand and compute. I went over the st-dbscan algorithm's code again and checked for errors that I might have missed before.

## Data Collecting and Rearrangement

During the fourth week of my internship, which was the data collecting and rearrangement step of the “work package”, I collected and rearranged the data that was used in the academic articles and I would use it to test the codes I wrote. For the st-dbscan algorithm, Spatio-temporal outlier detection algorithms based on computing behavioral outliers factor academic article uses the data a buoy dataset that consists of 170,645 readings from the Gulf of Mexico for the year 2005. A buoy is a sensor that takes hourly readings of weather conditions and the data was available on the National Data Buoy Center’s website. To clean up this data I found a code from a GitHub [1] that I changed so I could take the wanted data in the desired format. Other than that I tried to find other datasets that I could use. From my research, the most important and needed components of the dataset that are Spatio-temporal are time, longitude, and latitude. That is why I started to search for datasets that have these three components and I found some examples from two datasets from Chicago's bike-sharing service[2] and used them to test my st-dbscan code.

The CutESC algorithm article and in the article they give the list of datasets that have been used and also provide a link for the GitHub that processed the data for the missing values as well. After I collected all the datasets used then I took some extra ones from the GitHub provided as some additional datasets. At first, I collected the wrong data for some of them so I had to download some of the datasets again but I was able to download the correct ones at the end. On the last day of my fourth week, I wrote the input and plotted the cluster parts of both the st-dbscan and CutESC algorithms.

## Code Verification and Testing

In my fifth week, which was the code verification and testing step of the “work package”, I noticed the errors I had made on both of the codes and to fix these errors took a couple of days but towards the end of the week, I could test all of the datasets I acquired. For some time, I couldn’t figure out how to test the buoy dataset but after a while, I was able to run it by dividing the data by days. I am adding most of the results I found to the results section down below (Fig 2-18).

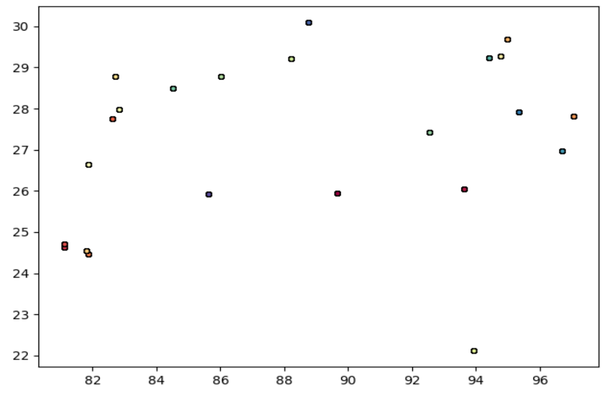
[1] https://github.com/modern-fortran/weather-buoys/blob/master/data/get_buoy.py
[2] https://www.divvybikes.com/system-data

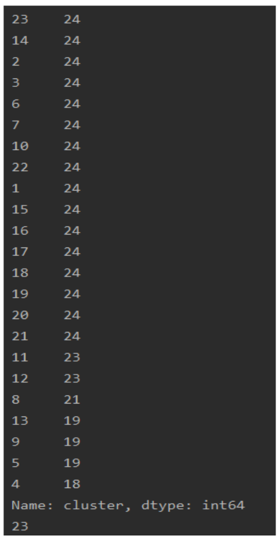
## Analysis and Improvements

In my final week, which was the analysis and improvements step of the "work package", I wrote the conclusions and future enhancements for both of the algorithms. The conclusion is that the St-dbscan algorithm's main goal is to find outliers in Spatio-temporal databases according to local Spatio-temporal proximity. It includes the steps; clustering, checking spatial neighbors, and finding spatial outliers, and then checking temporal neighbors and finding Spatio-temporal outliers. 25th - 29th of August 2005 Gulf of Mexico buoy dataset is used to test the algorithm. The algorithm takes location - longitude, latitude - and date-time attributes of the dataset to find outliers. In the future, st-dbscan can be tested with very large databases.

CutESC algorithm is another clustering algorithm but CutESC is based on proximity graphs while st-dbscan is based on local Spatio-temporal proximity. Also, another difference between st-dbscan and CutESC is that the CutESC algorithm doesn't need any prior information like thresholds. CutESC's steps are; turning the data points into a Delaunay graph and then turning the graph into a Gabriel graph which is a subgraph of a Delaunay graph. After these steps algorithm finds the long and irrelevant edges in the graph and gets rid of them. After removing globally long edges clusters are formed then long and irrelevant edges inside the clusters are identified locally and removed. Afterward, other locally long edges are found and removed. To test the CutESC algorithm clustering datasets given by the article and also other clustering datasets found on the internet have been used. CutESC's main advantage is that the algorithm can cluster the data without any prior knowledge also to be able to use the CutESC algorithm the data doesn't have to be Spatio-temporal. In the future, the algorithm could be tested on very large databases. The algorithm also could be used in different areas of computer science like computer vision, the ways to incorporate this algorithm can be discussed in the future as well.

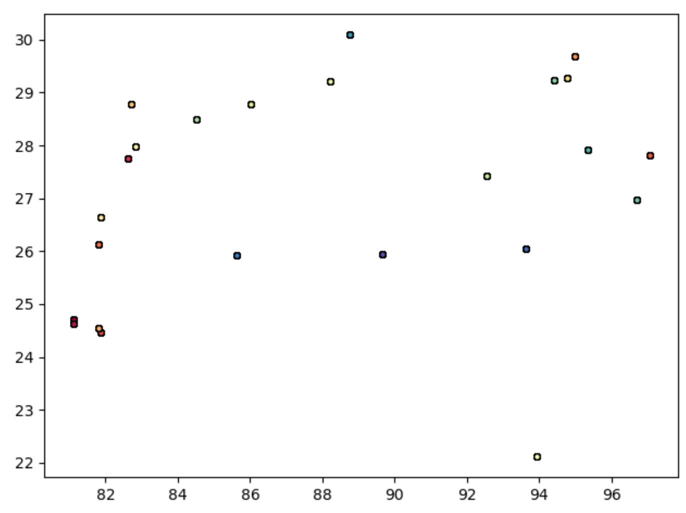
## Experimental Results and Discussion

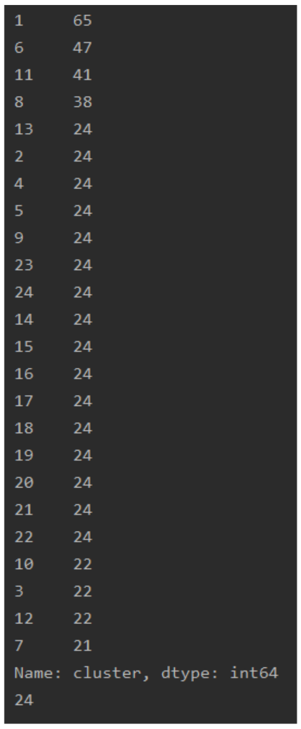




```
23    24
14    24
2     24
3     24
6     24
7     24
10    24
22    24
1     24
15    24
16    24
17    24
18    24
19    24
20    24
21    24
11    23
12    23
8     21
13    19
9     19
5     19
4     18
Name: cluster, dtype: int64
23
```

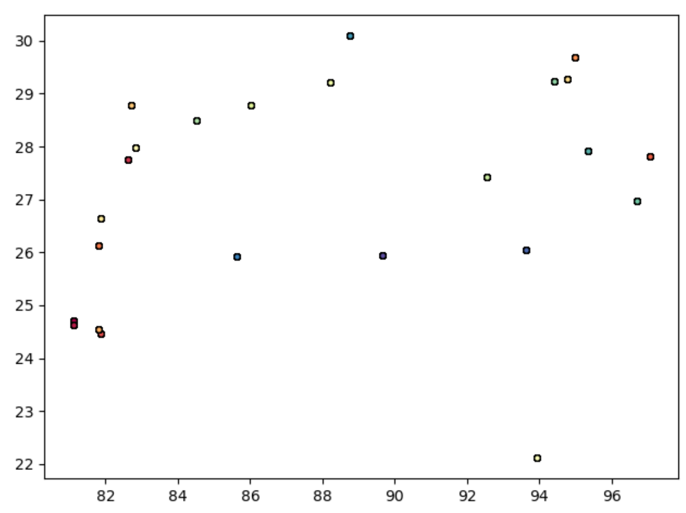
Fig 2. 25th of August 2005 Buoy Data st-dbscan results

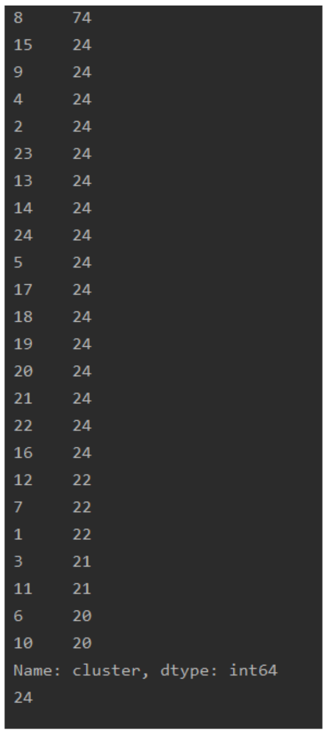




```
1     65
6     47
11    41
8     38
13    24
2     24
4     24
5     24
9     24
23    24
24    24
14    24
15    24
16    24
17    24
18    24
19    24
20    24
21    24
22    24
10    22
3     22
12    22
7     21
Name: cluster, dtype: int64
24
```

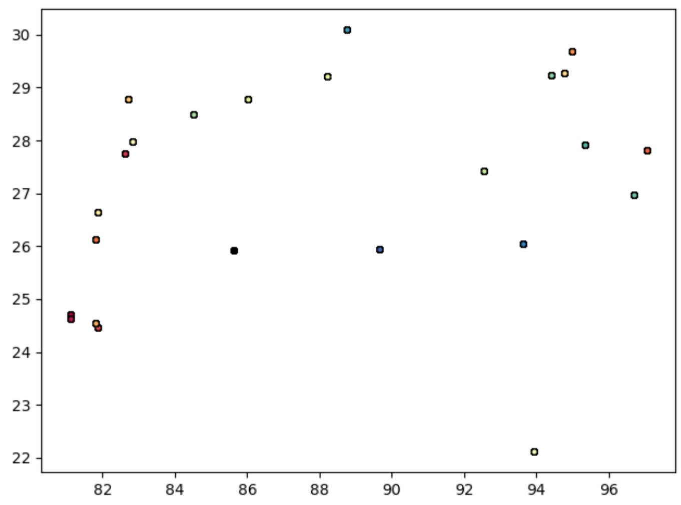
Fig 3. 26th of August 2005 Buoy Data st-dbscan results

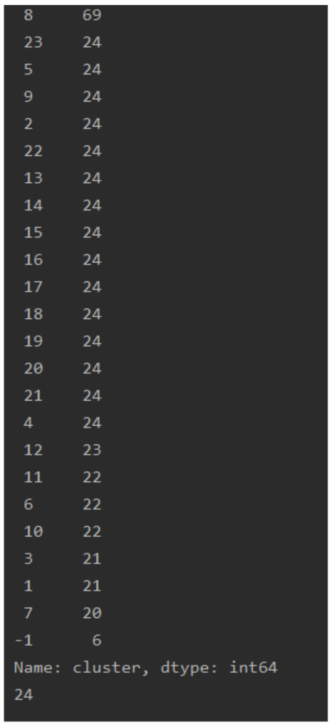




```
8     74
15    24
9     24
4     24
2     24
23    24
13    24
14    24
24    24
5     24
17    24
18    24
19    24
20    24
21    24
22    24
16    24
12    22
7     22
1     22
3     21
11    21
6     20
10    20
Name: cluster, dtype: int64
24
```

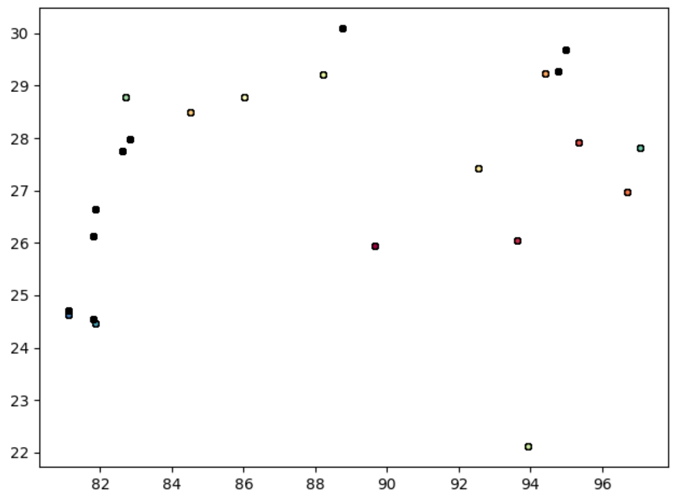
Fig 4. 27th of August 2005 Buoy Data st-dbscan results

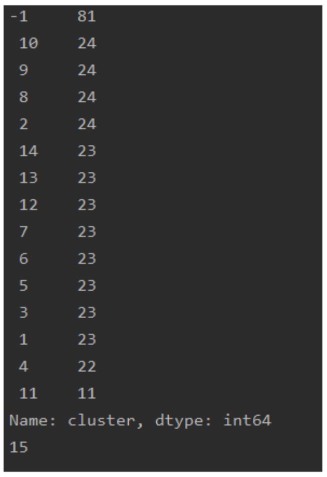




```
 8     69
 23    24
 5     24
 9     24
 2     24
 22    24
 13    24
 14    24
 15    24
 16    24
 17    24
 18    24
 19    24
 20    24
 21    24
 4     24
 12    23
 11    22
 6     22
 10    22
 3     21
 1     21
 7     20
-1      6
Name: cluster, dtype: int64
24
```

Fig 5. 28th of August 2005 Buoy Data st-dbscan results

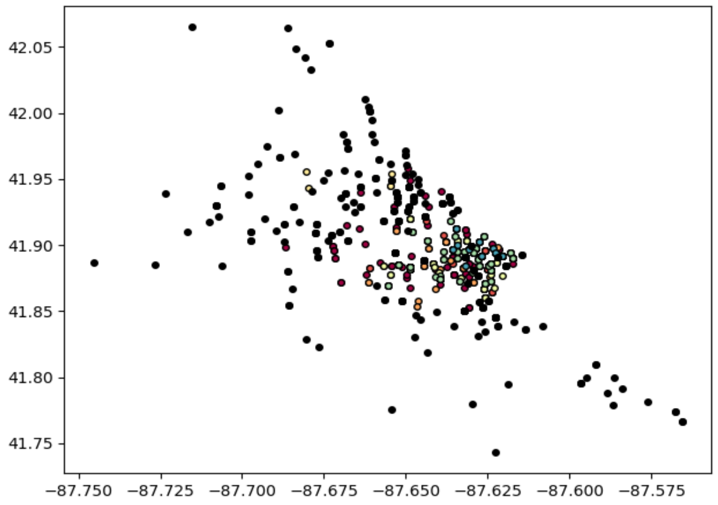


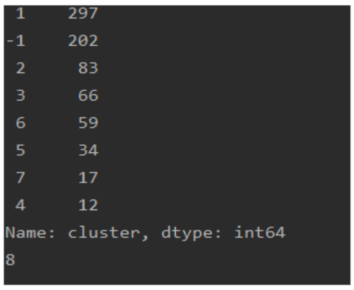


```
-1     81
 10    24
 9     24
 8     24
 2     24
 14    23
 13    23
 12    23
 7     23
 6     23
 5     23
 3     23
 1     23
 4     22
 11    11
Name: cluster, dtype: int64
15
```

Fig 6. 29th of August 2005 Buoy Data st-dbscan results

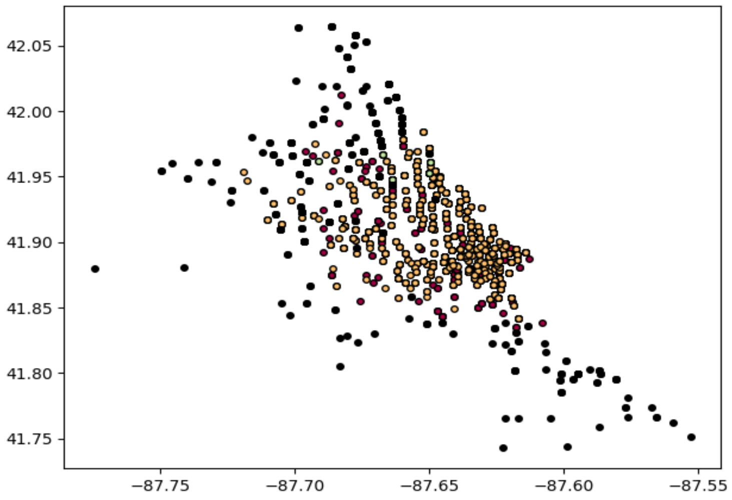
From the buoy, dataset results can be seen from Fig 2-6. In the first three days which are August 25th, August 26th, and August 27th there are no outliers detected and in the 28th and 29th there are outliers detected, they are shown with -1 number, this makes sense because Hurricane Katrina happened during this time and as the time passed more buoys started to malfunction so there are more errors that occurred. Also, there are the least amount of entries on the 29th so this day has the least number of clusters. It can be deduced that the more data there may be more clusters can be found.

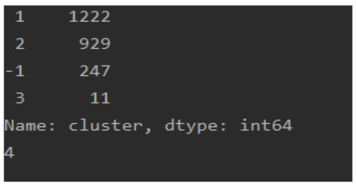




```
 1    297
-1    202
 2     83
 3     66
 6     59
 5     34
 7     17
 4     12
Name: cluster, dtype: int64
8
```

Fig 7. Part of Chicago's bike-sharing service data st-dbscan results





```
 1    1222
 2     929
-1     247
 3      11
Name: cluster, dtype: int64
4
```

Fig 8. Other example data st-dbscan result

Chicago's bike-sharing service data was divided into 771 entries and resulted in Fig 7 and also divided into 2410 entries which resulted in Fig 8. The second dataset has more data but doesn't have more clusters but the reason for this could be; the data selected on the second bike dataset is a lot closer to each other according to the timeframe so the temporal threshold makes the difference but it can be seen from the dataset getting farther from the cluster will result in outlier detection because of the spatial threshold. St-dbscan will show better results for larger datasets but this makes the program run slower.

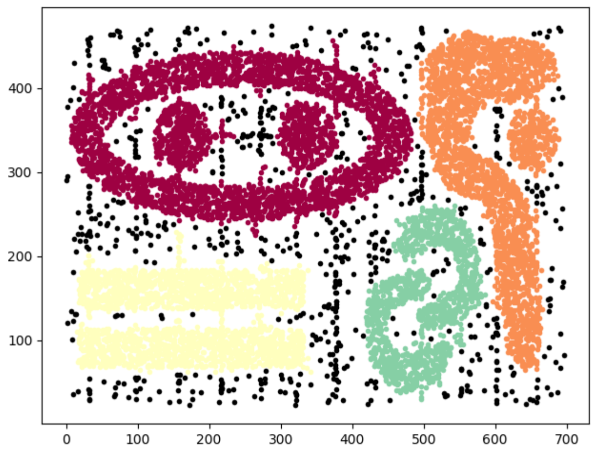


Fig 9. t7-10k clustering data CutESC result

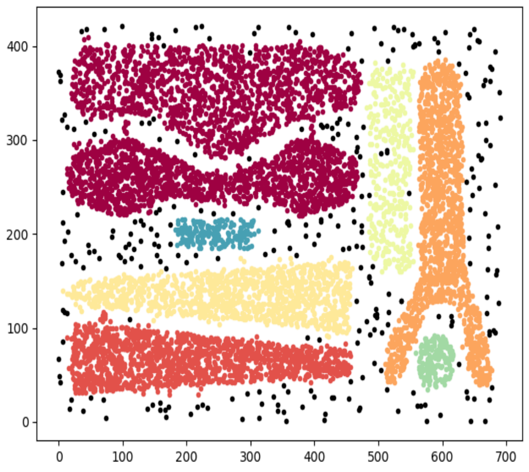


Fig 10. t8-8k clustering data CutESC result

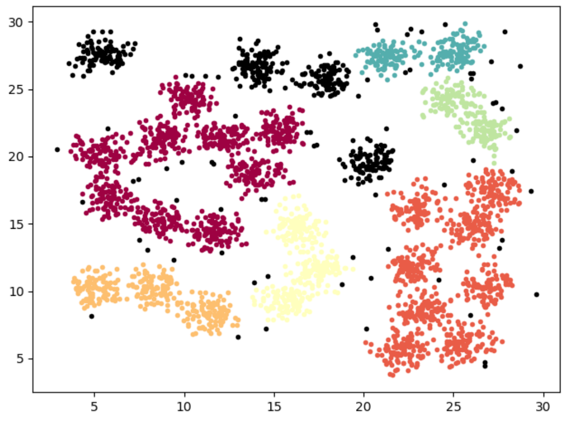


Fig 11. D31 clustering data CutESC result

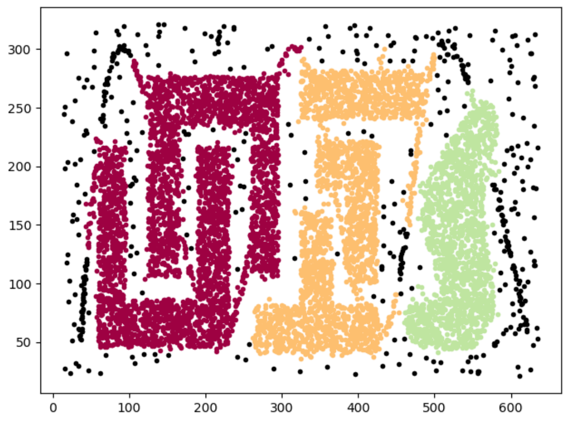


Fig 12. t4-8k clustering data CutESC result

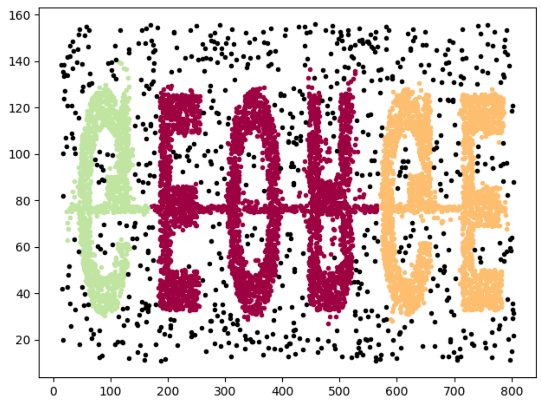


Fig 13. t5-8k clustering data CutESC result

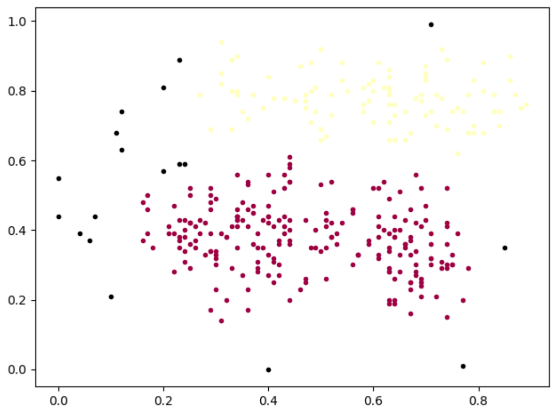


Fig 14. ecoli clustering data CutESC result

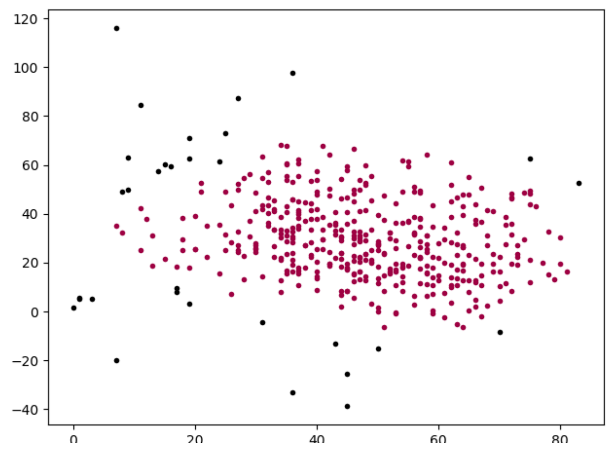


Fig 15. arrhythmia clustering data CutESC result

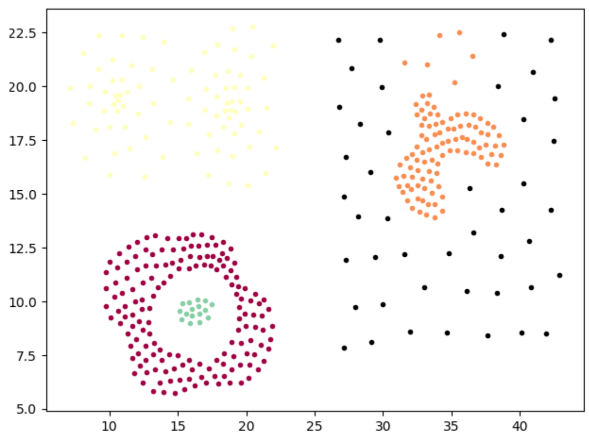


Fig 16. compound clustering data CutESC result

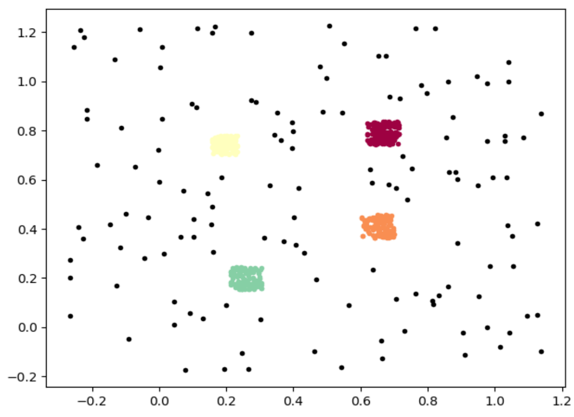


Fig 17. Zelnik-4 clustering data CutESC result

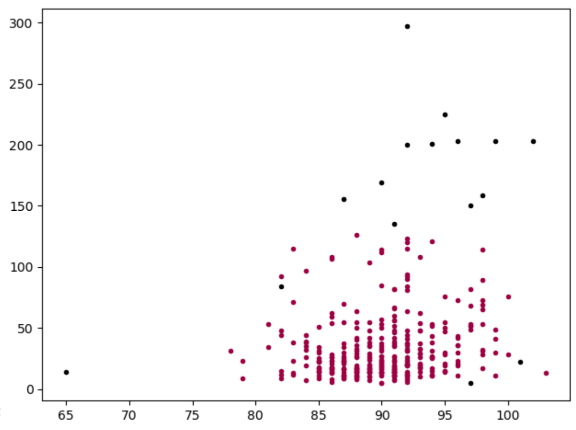


Fig 18. Bupa clustering data CutESC result

CutESC algorithm was tested on bupa, arrhythmia, compound, zelnik-4, D31, t4-8k, t5-8k, t7-10k, t8-8k datasets. It can be deduced that the more data there are, the better results the CutESC would show but again just like st-dbscan that would make the program run slower so we could say there is a tradeoff.  If there is a bigger number of classes of data there are, there will be more clusters detected and if the cluster points are close to each other, the clusters will look more precise.

## Other Clustering Algorithms and Comparison

While I was researching, I encountered and read about a couple of other clustering and outlier detection algorithms like DBSCAN (Density-based spatial clustering of applications with noise), LDBSCAN (Local density based on spatial clustering algorithm with noise), and the ones I implemented which are St-dbscan (Algorithm for clustering Spatio-temporal data) and CutESC (Cutting edge spatial clustering technique based on proximity graphs) algorithms. DBSCAN, LDBSCAN, and St-dbscan algorithms are density-based algorithms that are based on dividing the areas with high density which are clusters, and low density which are outliers, unlike the CutESC algorithm which uses proximity graphs to detect clusters. DBSCAN algorithm is the basic density-based algorithm, it takes two parameters which are the spatial threshold and minimum points threshold. LDBSCAN is improved upon DBSCAN and introduces local outlier factor (LOF) and local reachability density (LRD); LDBSCAN takes three parameters which are the minimum point threshold for LOF, minimum point threshold, and pct threshold. The st-dbscan algorithm is also improved upon the DBSCAN algorithm and can work on Spatio-temporal data and takes three parameters which are spatial threshold, temporal threshold, and minimum points threshold. Lastly, the CutESC algorithm unlike the density-based algorithms mentioned before doesn't take parameters so CutESC doesn't require any prior knowledge like thresholds.

# 4 Performance and Outcomes

## 4.1 Applying Knowledge and Skills Learned at Hacettepe

For writing the codes for clustering algorithms I used python programming language. In Hacettepe I learned Python, Java, and C because of this I easily remembered how to code in Python and wrote the codes very fast.

During understanding the algorithms process; algorithms (BBM202) and data management (BBM371) knowledge was very useful.

## 4.2 Solving Engineering Problems

I have worked on clustering algorithms when I need to look for something I searched online, read articles and documents about the problem. Also, I watched videos about the topic I was researching on Youtube. If I couldn't find anything useful for me on the documents or couldn't figure out what to do, I asked my supervisor.

## 4.3 Teamwork

Because of the COVID – 19 pandemic, I did my internship with Hacettepe University faculty member Dr.Engin Demir so I did not have a team nor any teammates during my internship. I worked on researching the internship topics, coding, finding suitable data, and testing the data by myself but my supervisor helped with some of the problems I encountered.

## 4.4 Multi-Disciplinary Work

Because of the COVID-19 pandemic, I did my internship with Dr. Engin Demir, so I didn't have any teammates but I did all the work for researching, coding, finding suitable data, and testing the data by myself and the algorithms that I implemented can be used on many different areas for example the dataset I used for st-dbscan is a buoy dataset and another one for CutESC is for health. So it could be said that although I worked without teammates the area I worked in is multidisiplinary.

## 4.5 Professional and Ethical Issues

During my internship, I did not observe professional issues or work-related ethical issues.

## 4.6 Impact of Engineering Solutions

While I was researching the data clustering and outlier detection algorithms, I encountered some examples for the areas clustering algorithms used in. Some of these areas are; document classification, delivery store optimization, insurance fraud detection, and automatic clustering of IT alert, and many more[5].

Economically, If a clustering algorithm is used on areas like online shopping optimization or fraud detection that means that there will be more customers and fewer problems which means a higher chance of sales but the situations like these require different algorithms this is why I learned to implement two different algorithms; St-dbscan and CutESC algorithms so this would help me choose which algorithm should be implemented.

For global impact, we want the data clustering algorithm to be suitable to many systems so it could be implemented on many systems all around the world. This is why in my internship I learned about two different algorithms so I could decide the work requires which algorithm.

## 4.7 Locating Sources and Self-Learning

During my internship, I learned how to work with two data clustering and outlier detection algorithms. I researched the algorithms with Google Academics and looked into Youtube tutorials, I mostly used Spatio-temporal outlier detection algorithms based on computing behavioral outliers factor academic article, CutESC: Cutting edge spatial clustering technique based on proximity graphs academic article and Spatio-temporal outlier detection in large databases[6] academic article. To implement the algorithms I had to use some packages like networkX, kneedle so I researched the documentation for these packages as well.

To collect the data that is needed for the codes; for the st-dbscan algorithm, I used the national data buoy center's website[3] and for CutESC algorithm, I used some GitHub that provide data for clustering algorithms[4].

## 4.8 Using New Tools and Technologies

Although I always wanted to learn more about database management during the school terms I couldn't find time to do so. Naturally, I was very happy to start learning. I learned how data clustering algorithms work and which areas they can be used.

I didn't know much about data management at the start of the internship so I had to learn them from the basics but with the help of this internship, now I can make understand database management a lot better, and also I learned how to do extensive research. And before this internship, I hadn't used Jupyter Notebook too.

[3] https://www.ndbc.noaa.gov

[4] https://github.com/milaan9/Clustering-Datasets

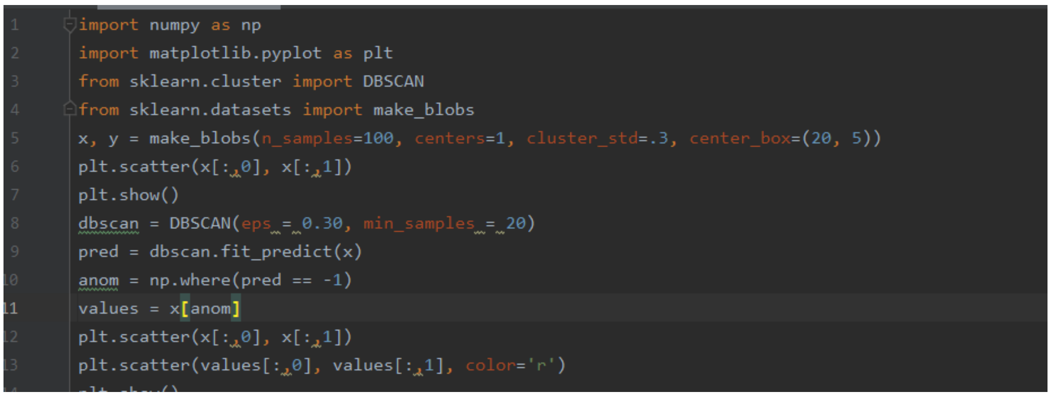
# 5 Conclusions

In conclusion, I worked on two data clustering and outlier detection algorithms and they are the st-dbscan algorithm and CutESC algorithm. I learned how to work on data management and use cases for clustering algorithms. I worked with python on PyCharm and used Jupyter Notebook. We learned python in the first term of our education so I knew the coding language. I used the things I learned in my BBM371 Data Management class. I worked solo on my internship and because in Hacettepe we work mostly alone so I was used to this kind of environment. I didn't have any teammates so I did all of the work alone, this was quite challenging but I think that helped me to be more responsible.

# References

[1] "History of Hacettepe University", URL:
https://www.hacettepe.edu.tr/english/hakkinda/tarihce [Accessed; September, 2020].
[2] "Vision of Hacettepe University", URL:
https://www.hacettepe.edu.tr/english/hakkinda/misyonvizyondegerler [Accessed; September, 2020].
[3] Duggimpudi, Maria Bala & Abbady, Shaaban & Chen, Jian & Raghavan, Vijay. (2019). Spatio-Temporal Outlier Detection Algorithms Based on Computing Behavioral Outlierness Factor. Data & Knowledge Engineering. 122. 1-24. 10.1016/j.datak.2017.12.001.
[4] Aksac, Alper & Özyer, Tansel. (2019). CutESC: Cutting Edge Spatial Clustering Technique based on Proximity Graphs. Pattern Recognition. 96. 10.1016/j.patcog.2019.06.014.
[5] "Use cases for clustering algorithms", URL: https://dzone.com/articles/10-interesting-use-cases-for-the-k-means-algorithm [Accessed; September, 2020].
[6] D. Birant, A.Kut, Spatio-temporal outlier detection in large databases, J. Comput. Inf. Technol. CIT 14 (4) (2006)

# Appendices



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
x, y = make_blobs(n_samples=100, centers=1, cluster_std=.3, center_box=(20, 5))
plt.scatter(x[:,0], x[:,1])
plt.show()
dbscan = DBSCAN(eps = 0.30, min_samples = 20)
pred = dbscan.fit_predict(x)
anom = np.where(pred == -1)
values = x[anom]
plt.scatter(x[:,0], x[:,1])
plt.scatter(values[:,0], values[:,1], color='r')
```

Fig 19.The code

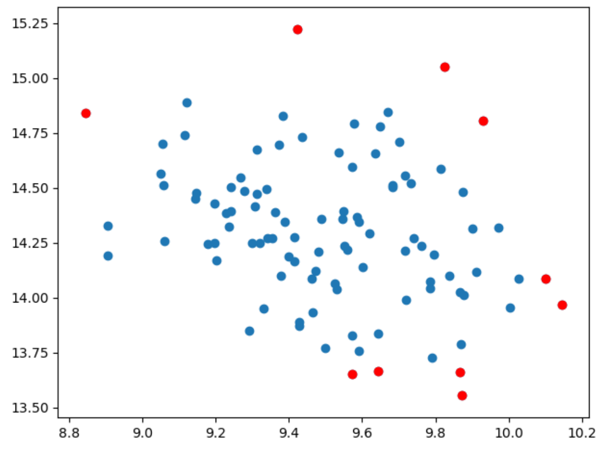
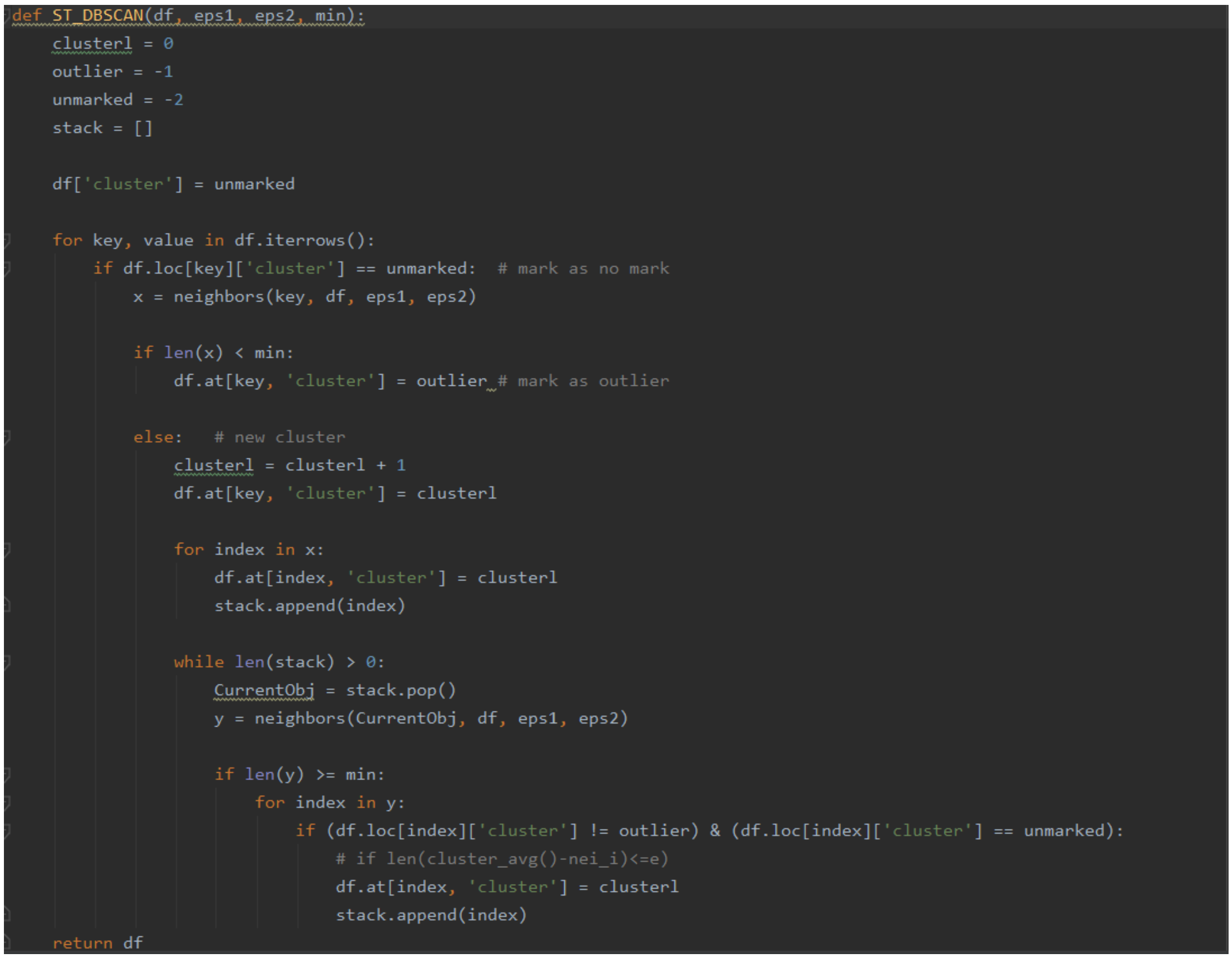
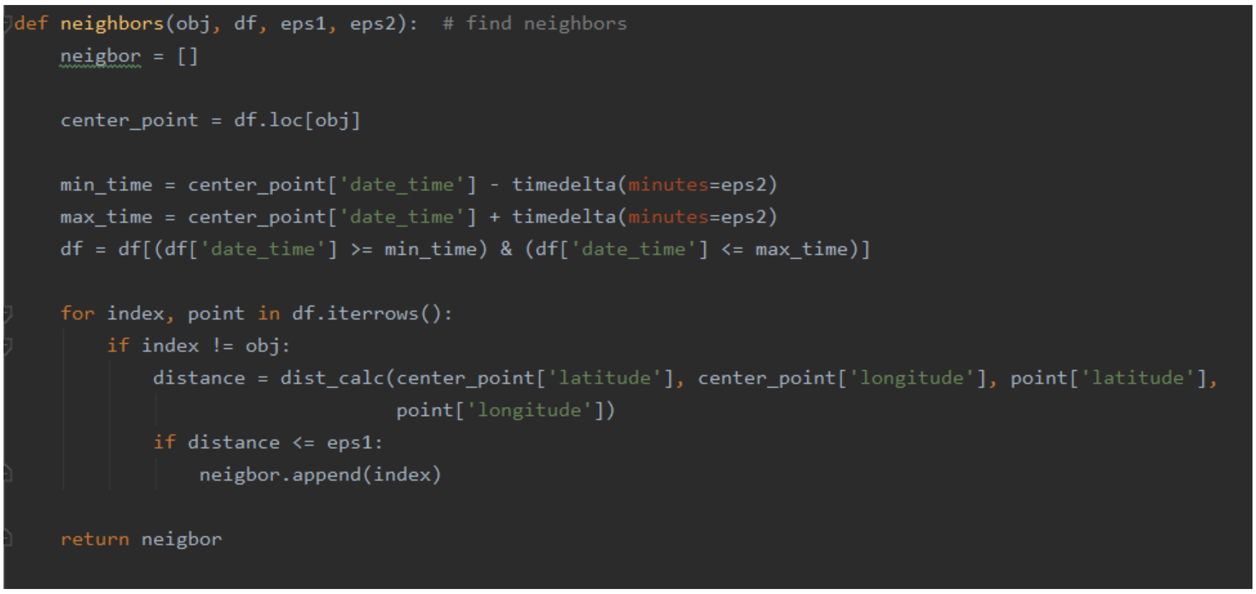


Fig 20. The result



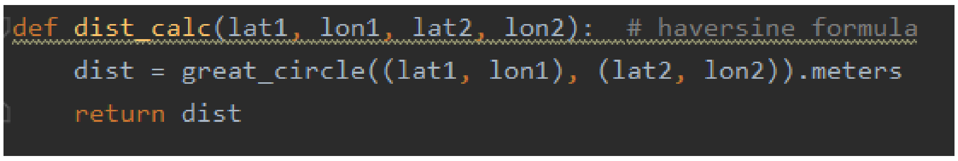
```
def ST_DBSCAN(df, eps1, eps2, min):
    clusterl = 0
    outlier = -1
    unmarked = -2
    stack = []

    df['cluster'] = unmarked

    for key, value in df.iterrows():
        if df.loc[key]['cluster'] == unmarked:  # mark as no mark
            x = neighbors(key, df, eps1, eps2)

            if len(x) < min:
                df.at[key, 'cluster'] = outlier # mark as outlier

            else:   # new cluster
                clusterl = clusterl + 1
                df.at[key, 'cluster'] = clusterl

                for index in x:
                    df.at[index, 'cluster'] = clusterl
                    stack.append(index)

                while len(stack) > 0:
                    CurrentObj = stack.pop()
                    y = neighbors(CurrentObj, df, eps1, eps2)

                    if len(y) >= min:
                        for index in y:
                            if (df.loc[index]['cluster'] != outlier) & (df.loc[index]['cluster'] == unmarked):
                                # if len(cluster_avg()-nei_i)<=e)
                                df.at[index, 'cluster'] = clusterl
                                stack.append(index)
    return df
```

Fig 21. The code fragment from St-dbscan function



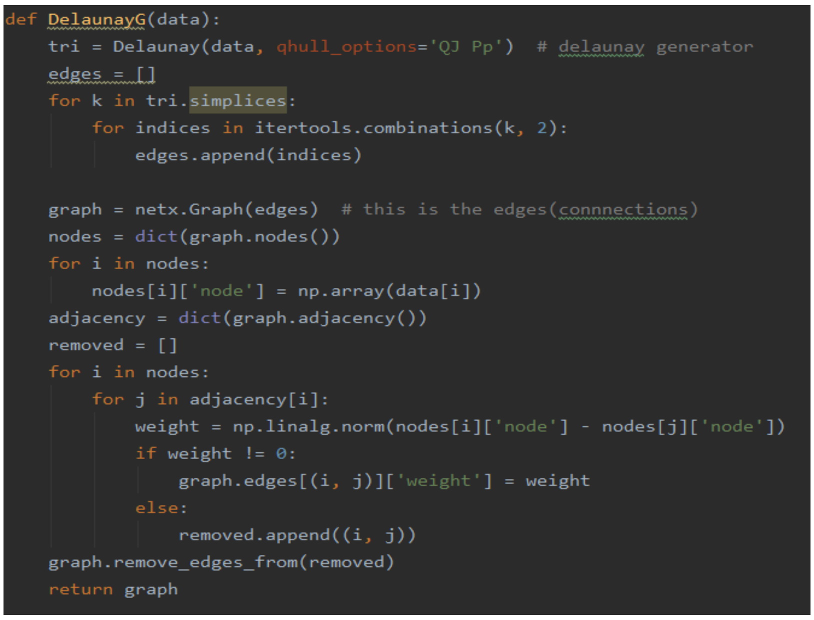
```python
def neighbors(obj, df, eps1, eps2):  # find neighbors
    neigbor = []

    center_point = df.loc[obj]

    min_time = center_point['date_time'] - timedelta(minutes=eps2)
    max_time = center_point['date_time'] + timedelta(minutes=eps2)
    df = df[(df['date_time'] >= min_time) & (df['date_time'] <= max_time)]

    for index, point in df.iterrows():
        if index != obj:
            distance = dist_calc(center_point['latitude'], center_point['longitude'], point['latitude'],
                                 point['longitude'])
            if distance <= eps1:
                neigbor.append(index)

    return neigbor
```

Fig 22. The code fragment from neighbors function



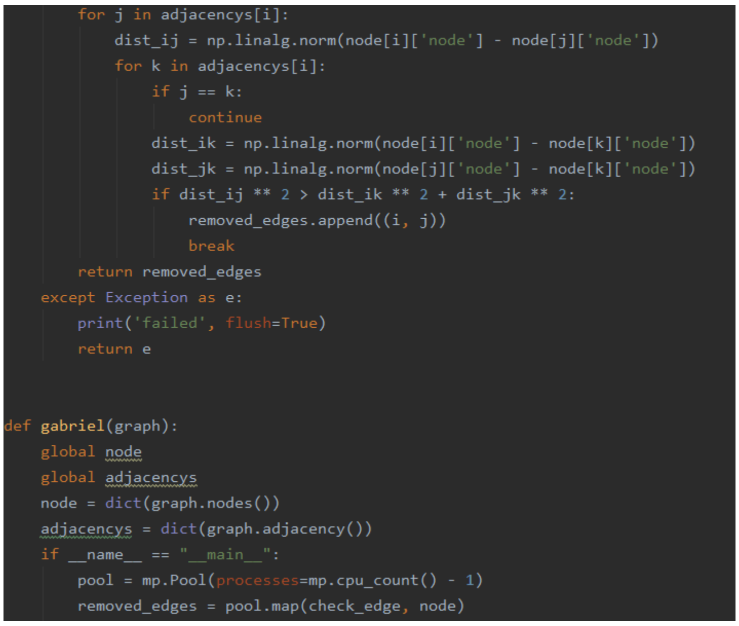
```python
def dist_calc(lat1, lon1, lat2, lon2):  # haversine formula
    dist = great_circle((lat1, lon1), (lat2, lon2)).meters
    return dist
```

Fig 23. The code fragment from disc_calc function



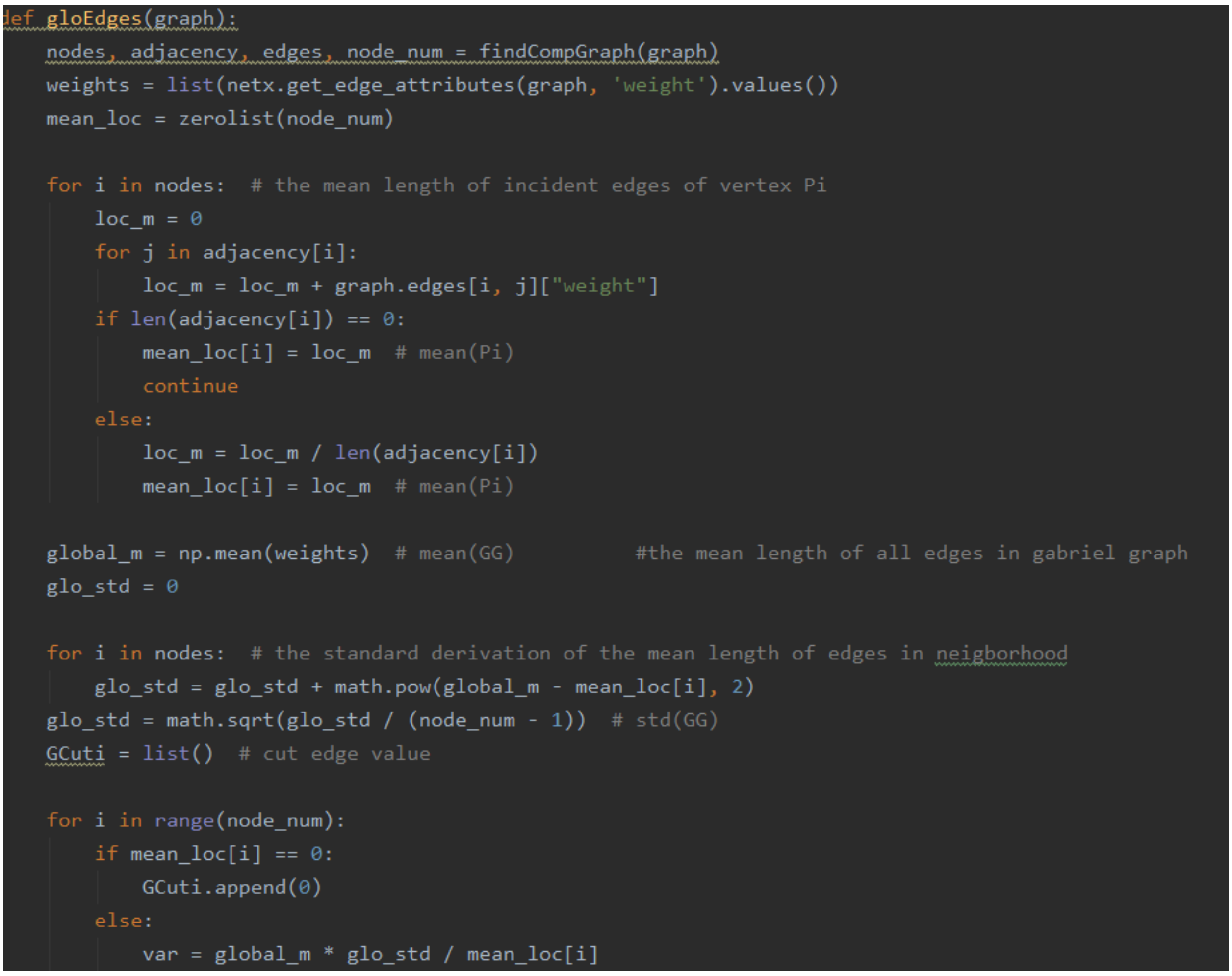
```python
def DelaunayG(data):
    tri = Delaunay(data, qhull_options='QJ Pp')  # delaunay generator
    edges = []
    for k in tri.simplices:
        for indices in itertools.combinations(k, 2):
            edges.append(indices)

    graph = netx.Graph(edges)  # this is the edges(connnections)
    nodes = dict(graph.nodes())
    for i in nodes:
        nodes[i]['node'] = np.array(data[i])
    adjacency = dict(graph.adjacency())
    removed = []
    for i in nodes:
        for j in adjacency[i]:
            weight = np.linalg.norm(nodes[i]['node'] - nodes[j]['node'])
            if weight != 0:
                graph.edges[(i, j)]['weight'] = weight
            else:
                removed.append((i, j))
    graph.remove_edges_from(removed)
    return graph
```

Fig 24. The code fragment from DelaunayG function



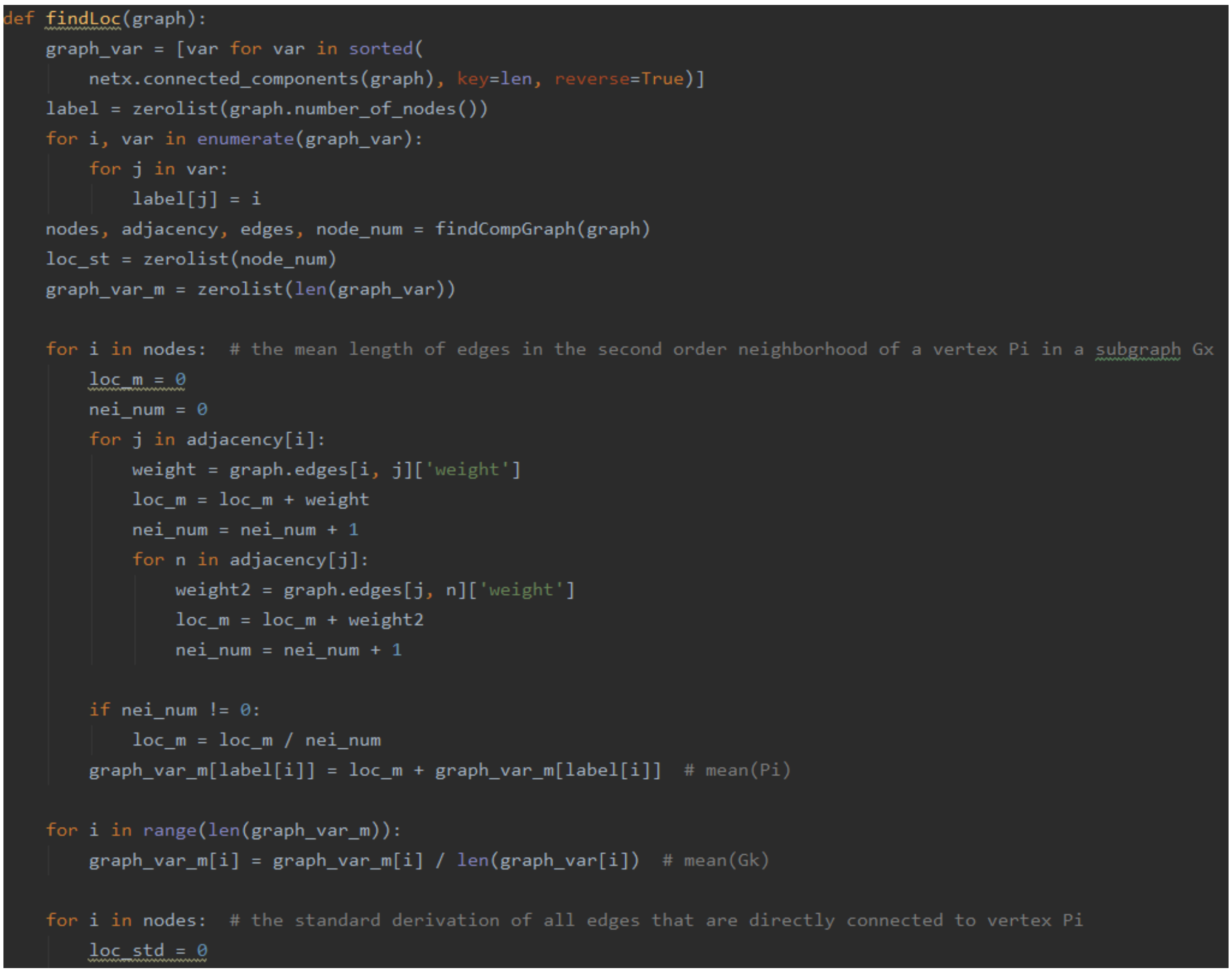
```
        for j in adjacencys[i]:
            dist_ij = np.linalg.norm(node[i]['node'] - node[j]['node'])
            for k in adjacencys[i]:
                if j == k:
                    continue
                dist_ik = np.linalg.norm(node[i]['node'] - node[k]['node'])
                dist_jk = np.linalg.norm(node[j]['node'] - node[k]['node'])
                if dist_ij ** 2 > dist_ik ** 2 + dist_jk ** 2:
                    removed_edges.append((i, j))
                    break
        return removed_edges
    except Exception as e:
        print('failed', flush=True)
        return e


def gabriel(graph):
    global node
    global adjacencys
    node = dict(graph.nodes())
    adjacencys = dict(graph.adjacency())
    if __name__ == "__main__":
        pool = mp.Pool(processes=mp.cpu_count() - 1)
        removed_edges = pool.map(check_edge, node)
```

Fig 25. The code fragment from gabriel function



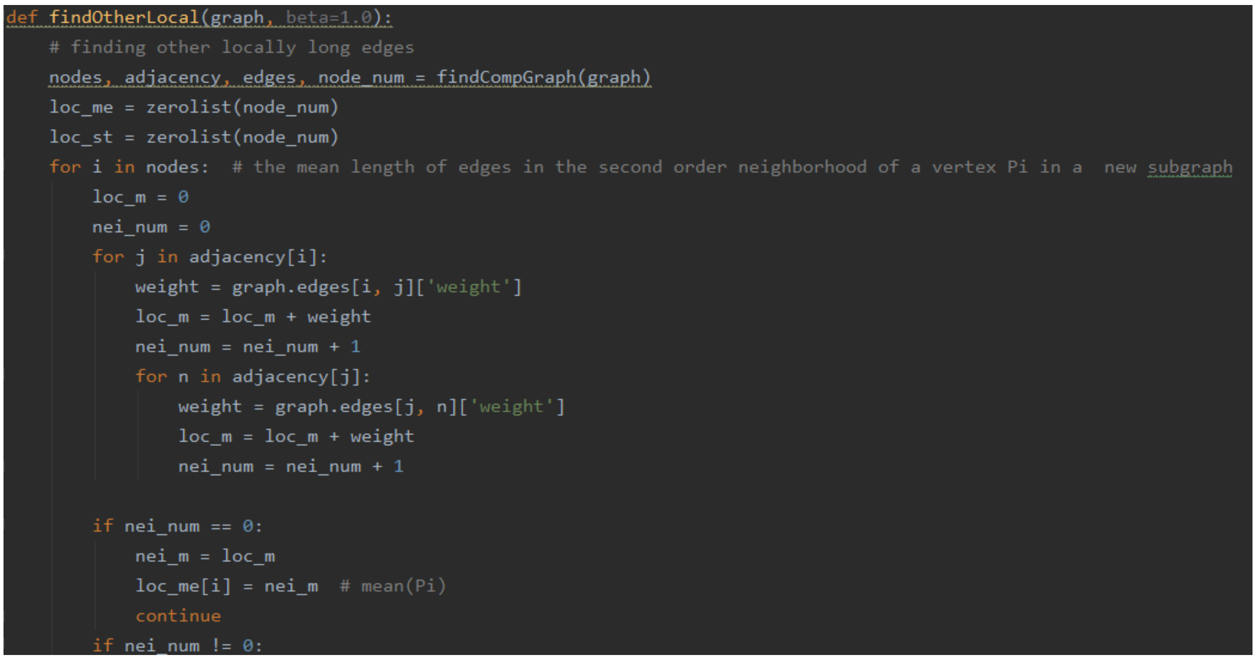
```
def gloEdges(graph):
    nodes, adjacency, edges, node_num = findCompGraph(graph)
    weights = list(netx.get_edge_attributes(graph, 'weight').values())
    mean_loc = zerolist(node_num)

    for i in nodes:  # the mean length of incident edges of vertex Pi
        loc_m = 0
        for j in adjacency[i]:
            loc_m = loc_m + graph.edges[i, j]["weight"]
        if len(adjacency[i]) == 0:
            mean_loc[i] = loc_m  # mean(Pi)
            continue
        else:
            loc_m = loc_m / len(adjacency[i])
            mean_loc[i] = loc_m  # mean(Pi)

    global_m = np.mean(weights)  # mean(GG)          #the mean length of all edges in gabriel graph
    glo_std = 0

    for i in nodes:  # the standard derivation of the mean length of edges in neigborhood
        glo_std = glo_std + math.pow(global_m - mean_loc[i], 2)
    glo_std = math.sqrt(glo_std / (node_num - 1))  # std(GG)
    GCuti = list()  # cut edge value

    for i in range(node_num):
        if mean_loc[i] == 0:
            GCuti.append(0)
        else:
            var = global_m * glo_std / mean_loc[i]
```

Fig 26. The code fragment from gloEdges function



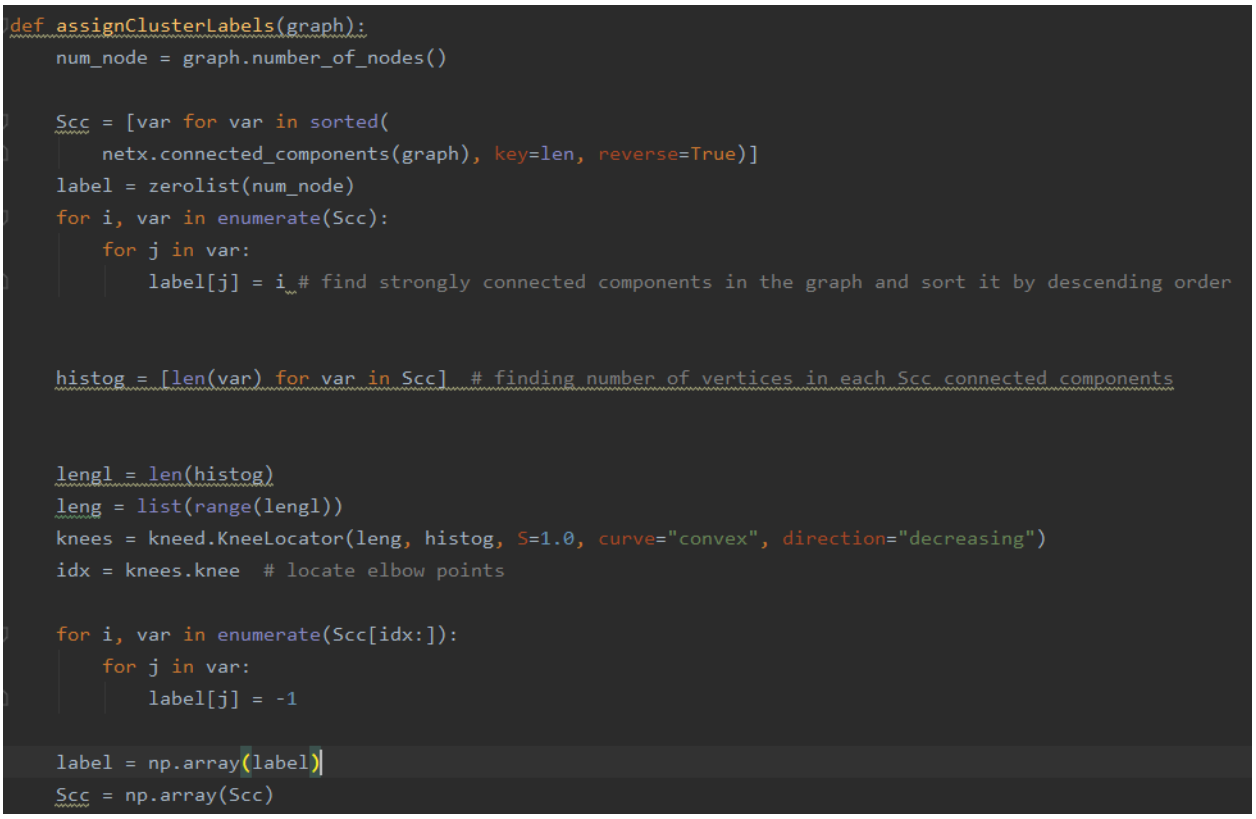
```python
def findLoc(graph):
    graph_var = [var for var in sorted(
        netx.connected_components(graph), key=len, reverse=True)]
    label = zerolist(graph.number_of_nodes())
    for i, var in enumerate(graph_var):
        for j in var:
            label[j] = i
    nodes, adjacency, edges, node_num = findCompGraph(graph)
    loc_st = zerolist(node_num)
    graph_var_m = zerolist(len(graph_var))

    for i in nodes:  # the mean length of edges in the second order neighborhood of a vertex Pi in a subgraph Gx
        loc_m = 0
        nei_num = 0
        for j in adjacency[i]:
            weight = graph.edges[i, j]['weight']
            loc_m = loc_m + weight
            nei_num = nei_num + 1
            for n in adjacency[j]:
                weight2 = graph.edges[j, n]['weight']
                loc_m = loc_m + weight2
                nei_num = nei_num + 1

        if nei_num != 0:
            loc_m = loc_m / nei_num
        graph_var_m[label[i]] = loc_m + graph_var_m[label[i]]  # mean(Pi)

    for i in range(len(graph_var_m)):
        graph_var_m[i] = graph_var_m[i] / len(graph_var[i])  # mean(Gk)

    for i in nodes:  # the standard derivation of all edges that are directly connected to vertex Pi
        loc_std = 0
```

Fig 27. The code fragment from findLoc function



```python
def findOtherLocal(graph, beta=1.0):
    # finding other locally long edges
    nodes, adjacency, edges, node_num = findCompGraph(graph)
    loc_me = zerolist(node_num)
    loc_st = zerolist(node_num)
    for i in nodes:  # the mean length of edges in the second order neighborhood of a vertex Pi in a  new subgraph
        loc_m = 0
        nei_num = 0
        for j in adjacency[i]:
            weight = graph.edges[i, j]['weight']
            loc_m = loc_m + weight
            nei_num = nei_num + 1
            for n in adjacency[j]:
                weight = graph.edges[j, n]['weight']
                loc_m = loc_m + weight
                nei_num = nei_num + 1

        if nei_num == 0:
            nei_m = loc_m
            loc_me[i] = nei_m  # mean(Pi)
            continue
        if nei_num != 0:
```

Fig 28. The code fragment from findOtherLocal function



```python
def assignClusterLabels(graph):
    num_node = graph.number_of_nodes()

    Scc = [var for var in sorted(
        netx.connected_components(graph), key=len, reverse=True)]
    label = zerolist(num_node)
    for i, var in enumerate(Scc):
        for j in var:
            label[j] = i # find strongly connected components in the graph and sort it by descending order


    histog = [len(var) for var in Scc]  # finding number of vertices in each Scc connected components


    lengl = len(histog)
    leng = list(range(lengl))
    knees = kneed.KneeLocator(leng, histog, S=1.0, curve="convex", direction="decreasing")
    idx = knees.knee  # locate elbow points

    for i, var in enumerate(Scc[idx:]):
        for j in var:
            label[j] = -1

    label = np.array(label)
    Scc = np.array(Scc)
```

Fig 29. The code fragment from assignClusterLabels function