ICSI 532 Project: Analyzing the Network of Email Correspondences

R Padmavathi Iyer
State University of New York
Albany, NY
riyer2@albany.edu

Younes Karimi
YK951683
State University of New York
Albany, NY
ykarimi@albany.edu

Emre Ugurlu
State University of New York
Albany, NY
eugurlu@albany.edu

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings.

KEYWORDS

Email Correspondence; Temporal Networks; Ego-Networks; Power Law Distribution; Change Point Analysis

ACM Reference Format:
R Padmavathi Iyer, Younes Karimi YK951683, and Emre Ugurlu. 2018. ICSI 532 Project: Analyzing the Network of Email Correspondences. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Emails have become an inherent form of communication in today's digital world. Millions of emails are sent to and received by people around the world each day since it is a reliable and efficient means of communicating information. Every email correspondence involves conveying some information from one party to another. An individual can send an email to another individual, or to a group (as in the case of mass communication).

Because emails are used abundantly these days, it is important to investigate the science of email correspondence for multiple reasons like personalizing customer experience or detecting some fraudulent or anti-social activities. Analyzing the patterns of communication between people can provide numerous types of information about the relationship between people. It can also provide some useful insights regarding strongly knit groups of people, or changes in the relationship between people over time.

In this project, we discuss various types of analysis on the network of email correspondences. Particularly, given a dataset of email correspondences, comprising of sender, receiver and the time of communication, we first create a graph, where nodes are the email addresses and edges are the number of emails sent from one email address to another, and then obtain different kinds of information from the graph pertaining to the pattern of email correspondences. At a high level, we consider the following analysis over the communication network:

- Basic network metrics like nodes, edges, diameter, degree distribution, and so on.
- Examining the network around each individual by creating an *Ego-Network* for each node (that is, email address).
- Reviewing the pattern of communications over time as a reflection of *Temporal Networks*.
- Identifying the timestamps corresponding to significant changes between temporal networks using a conventional algorithm called *Change Point Analysis*.

A network comprising of nodes and edges can have different types of structures like star, clique, etc., or it can also have a combination of different structures. A *star* graph structure is encountered when all edges are coming out or going into a single node. This usually happens when bulk email messages are sent out. In a *clique* structure, there is an edge between every pair of nodes. The density of cliques is, therefore, 1. The dataset of email communications considered in this project exhibits both star structures and near-clique structures (that persist with time indicating regular communication over the network).

Section 2 discusses the dataset of email correspondence in detail. Section 3 gives a detailed explanation of *Task 1* (which deals with the basic network matrices), followed by examination of *Task 2* (which deals with ego-networks) and *Task 3* (which deals with temporal networks) in Section 4 and Section 5 respectively. Section 6 reviews the *Extra Credit*, where we analyze the *Change Point Analysis* algorithm to identify the points in time where significant changes occurred between any two consequent temporal graphs. Finally, we conclude in Section 7, followed by a detailed description of work distribution among the team members in Section 8.

2 DATASET DESCRIPTION

In this project, we used a dataset comprising email correspondence in the following format: `<UnixEpoch sender receiver>` which exhibits both periodic structures (e.g., near-cliques that persevere with time) as a reflection of regular communication and one-shot structures (e.g., stars) due to bulk email messages being sent out. The dataset includes some self-loops, but we removed them in our dataset-reader program. The dataset totally has 84716 number of nodes which are unique senders or receivers, and 346572 edges which are the unique correspondences between those edges regardless of the number of times they have been repeated.

Conference'17, July 2017, Washington, DC, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

UnixEpoch – also known as POSIX time or UNIX Epoch time – is a system for describing a point in time, defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970,[2]. Every day is treated as if it contains exactly 86400 seconds[2].

3 TASK 1: BASIC METRICS

This section involves the basic metrics and measures of the underlying network.

3.1 Implementation

In what follows, we will elaborate on our approach for calculating each of the metrics for the network.

3.1.1 Read Dataset. For retrieving the dataset, first we read the dataset text file using the *pandas* library[6], then convert that data to a *numpy* array to be able to apply some analysis on the stored data. The Python function which reads the dataset is called *GenerateGraph.py* and has been attached to the supporting files of the project[1].

We converted the *UnixEpoch* times to date-time format using a library function from *datetime*[1] Python library, then we removed those correspondences occurred during weekends. Afterward, we created a directed graph with all the emails in each side of the correspondences as the nodes and each of the correspondences as an edge between the corresponding nodes. Based on the number of correspondences between each pair of nodes and in the same direction (e.g., the first node of pair as the sender and the second, as the receiver), we added weights to the graph edges.

3.1.2 Bidirectional Edges. First we find the number of edges like (u, v) in which the node u is also in the list of the neighbours of the node v, $G[v]$.

3.1.3 Diameter. For calculating the diameter of the graph, first we tried to obtain the set of all strongly connected components of the graph and find the maximum diameter of those sub-graphs. But, according to some errors which we got, we found out that the graph does not have any strongly connected component. Therefore, we converted our directed graph to an undirected graph and tried to retrieve the set of connected components of the undirected graph and find the maximum diameter of those graphs.

3.2 Summary Statistics

In this part, we calculated some measures of the created graph as follows:

Table 1: Graph info of the directed graph

Directed graph	
Number of nodes	84716
Number of edges	346572
Average degree	7.5076
Number of bidirectional edges	28566
Number of self-loops	0

Table 2: Summary statistics of the correspondences between the network nodes

		Min.	Avg.	Max.
Correspondences	Incoming	0	4.091	1332
	Outgoing	0	4.091	1543
	Total	0	8.182	1937

After converting the graph to an undirected graph for the purpose of finding the diameter of the network, the number of edges and average degree of the nodes will change, but obviously, the number of the nodes is the same. The reason is that when we convert that to an undirected graph, those edges which exist between the same nodes but in the different direction, will be merged together, thus decreasing the total number of edges and the average degree of the nodes in the network. So, the new values would be as follows:

Table 3: Graph info of the converted undirected graph

Undirected graph	
Number of nodes	84716
Number of edges	318006
Average degree	7.5076
Diameter of the graph	78524

As it can be seen in the Table 3, the new value for the number of edges is actually the differences between the previous number and the number of bidirectional edges in the Table 1.

3.3 Distribution of Degrees

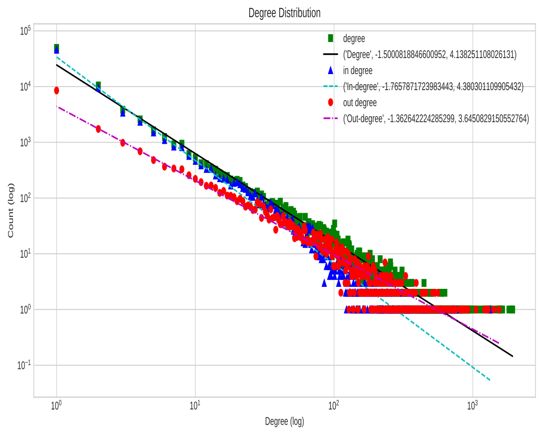
In the Figure 1a, we have plotted the distributions of *degree*, *in-degree*, and *out-degree* of nodes in the network on a log-log scale. We have also used the `polyfit` function from the `numpy` library in order to find the coefficients of the best corresponding fitting least square regression line for each of these distributions which are shown as the legends of the figure.

As you may see in the Table 2, the minimum numbers of degree, in-degree, and out-degree are equal to zero. So, for calculating the logarithm of the degrees, first we need to exclude the zero which is not in the range of logarithm function.

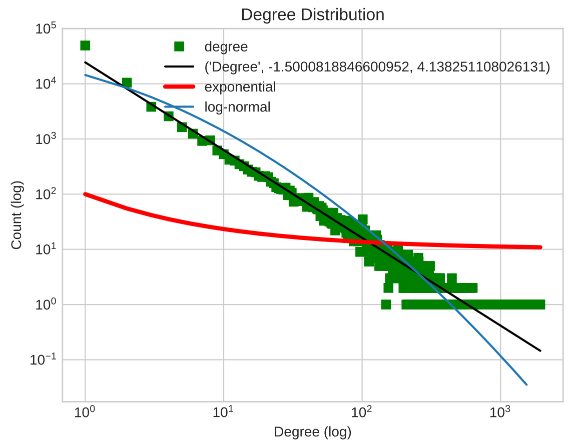
3.4 Comparing the Distributions

As it can be easily obtained from the Figure 1b which includes three different distributions of *Power-law*, *exponential*, and *log-normal*, and are plotted in colors of black, red, and blue respectively, the power-law distribution is the best possible fit and representation for the distribution of degrees of the nodes in the underlying network.

[1]All the project files, codes, and figures are accessible from GitHub: https://github.com/UKA0324/NS-Project.git

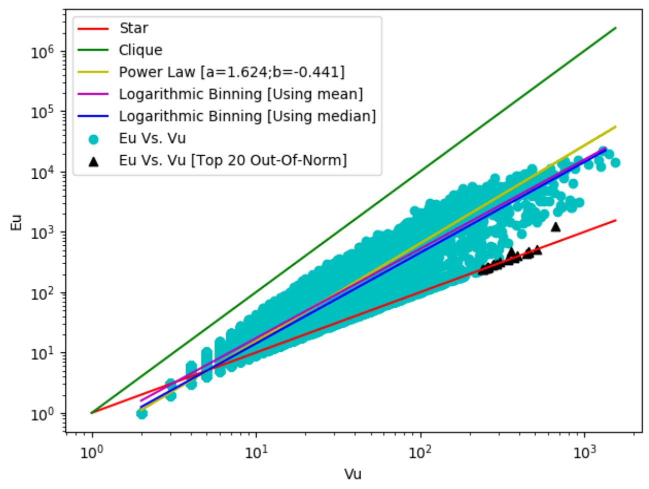


(a) Degree distributions and corresponding regression lines

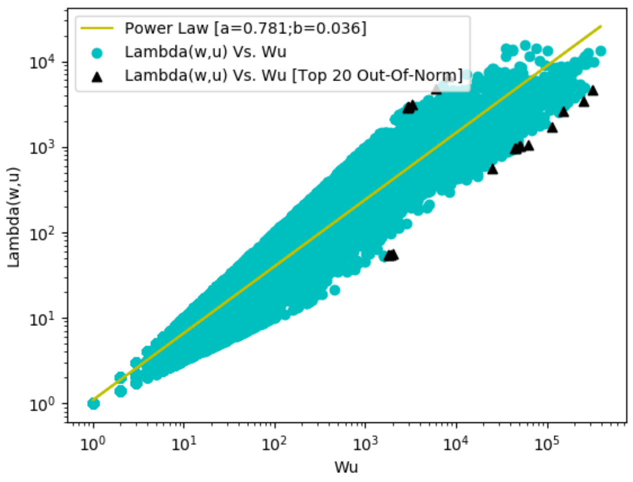


(b) Comparison of degree distributions

Figure 1: Degree distributions of the network



(a) Number of edges versus number of nodes



(b) Principal eigenvalue versus total weight

Figure 2: Analysis of ego-networks for each node and *out-of-the-norm* score of a node

4 TASK 2: EGO-NETWORKS

This section discusses about Task 2 in detail, which deals with creating ego-networks for each node and analyzing various parameters of the ego-network corresponding to each node. In the first subsection, we explain the concept of ego-networks and describe the objective of the task. Subsequently, in the next subsection, we enumerate the features of the data model, including any assumptions, to solve the task. The complete approach for answering this task, involving the programming perspectives, is elucidated in the following subsection. Finally, we note the observations and analyze the results to gain meaningful insights about the given email correspondence network.

4.1 Basic Concepts and Task Description

An *ego-network* consists of two components: *ego* and *alters*. The term ego is used to refer to a single node. Consequently, the neighbors of the node or the ego are referred to as its alters. Then 1-degree ego-network of a node is a subgraph of the original network centered at the ego and containing all its alters. Moreover, 1.5 egocentric network of a node is 1-degree ego-network of the node plus the connections between the node's neighbors.

The *out-of-the-norm* score of a node u, given the power law equation $y = Cx^{\alpha}$ for $f(x,y)$, is given as:

$$o(u) = \frac{max\{y_u, C{x_u}^{\alpha}\}}{min\{y_u, C{x_u}^{\alpha}\}} log(|y_u - C{x_u}^{\alpha}| + 1)$$

That is, $o(u) = 0$ when y_u is equal to the expected value $C{x_u}^{\alpha}$.

In Task 2, we need to extract the ego-network for every node in the email correspondence network, and compute the total number of nodes, total number of edges, total weight and the principal eigenvalue of the weighted adjacency matrix for every ego-network. Using these information, we are supposed to create two plots on a log-log scale, one for the number of edges versus number of nodes and the other for the principal eigenvalue versus total weight, and fit power laws corresponding to each of the plot. Additionally, we also need to depict top 20 out-of-the-norm nodes in each of the above two plots. Besides, in the number of edges versus number of nodes plot, we are also required to draw a fitting line through the median values for each bucket of points after applying logarithmic binning on the number of nodes axis, and two lines corresponding to stars and cliques with slopes 1 and 2 respectively.

4.2 Data Model

In this subsection, we describe the data model and assumptions, pertaining to the network of email correspondences, used to implement the task described above. We exclude the weekends and self-loops from the provided email correspondences network, as stated in previous sections. We consider 1.5-degree egocentric network for each node. Since the given email correspondences network is directed, we obtain the *out* neighborhood or successors for each ego. Moreover, the extracted ego-network for every node is undirected. In each of the ego-network, the weights associated with each edge will remain the same as in the original email correspondence network. Additionally, nodes having zero degree are removed from the network before extracting the ego-networks. This is because nodes with zero degree correspond to ego-networks with no edges or neighbors, so they can be ignored since they would not yield much information about the network under interest. Moreover, prior to performing a log-log plot, we need to remove all the zero entries from both the axes.

For both the log-log plots, we use base 10 on x-axis as well as y-axis. For the first plot, y-axis represents the number of edges and the x-axis represents number of nodes, for ego-network corresponding to every node. Similarly, for the second plot, y-axis represents the principal eigenvalue and x-axis represents the total weight. Also, in the calculation of principal eigenvalue of weighted adjacency matrix, only the real part is considered and the complex or imaginary part is ignored. In the logarithmic binning of the x-axis for the first plot, 20 buckets or 21 bins are used.

4.3 Implementation

In this task, we used Python 3.5 and employed the Python libraries: *networkx* [4], *numpy* [5] and *matplotlib* [3]. The networkx library was used to deal with graph related functions. The numpy library was mainly used for complex numerical computations, like calculating eigenvalues of matrices, and to also deal with large data in the form of vectors and arrays. Visualization of results through plots was implemented through the matplotlib library.

The code for this task is provided in *Task2.py* file. To extract the ego-networks for every node we utilized networkx's *ego_graph* function that returns induced subgraph of neighbors centered at a particular node, and then used networkx's *to_undirected* function to make all the ego-networks undirected. Python's in-built *map* function was used to get a list of ego-networks corresponding to each node in the email correspondence network, following which various parameters like number of nodes, number of edges, total weight and principal eigenvalue of weighted adjacency matrix was obtained for every ego-network. We used numpy's *linalg.eigvals* function to calculate the eigenvalues for a given matrix.

Numpy's *polyfit* function was used to compute the coefficients for the power law fits. Further, the *logspace* and *digitize* functions of numpy was employed to create logarithmic bins and divide the points among the various bins. Matplotlib's *scatter* and *plot* functions were used for drawing plots. Further, to plot on a log-log scale, we used the and *set_xscale set_yscale* functions provided by matplotlib.

4.4 Analysis of Results

Figure 2a depicts the observations from plotting total number of edges versus total number of nodes for ego-network corresponding to each node in the email correspondence network. As shown in the plot, the number of edges is increasing with the number of nodes, that is, if a node has more neighbors then there are be more edges in the ego-graph of that node. This makes sense because the given email correspondence network exhibits near-cliques that persevere with time, so more nodes means greater number of connections between nodes. Also, the spectrum is bounded by the two lines corresponding to star and clique, indicating periodic and one-shot structures of the provided network. Besides, the spectrum is slightly lower than the clique line because the given network exhibits near-clique structures and not actual clique structures, since none of the ego-networks has number of edges of the order of square of the number of nodes. Points on the lower side of the spectrum fall on the line corresponding to the star structure.

Figure 2a manifests the fitting power law to the edges versus nodes plot. The slope of the line, that is, the value of power law exponent, is 1.624, and the value of y-intercept of the fitting line is -0.441. This fitting power law shows that the number of edges versus number of nodes plot is a power law distribution. Additionally, the plot also demonstrates least square fit, after applying logarithmic binning on the x-axis, through the mean and median values for each bucket of points. There is a slight variation in the fitting line depending on whether we use the mean value or the median value for each bucket of points, especially when the number of nodes and number of edges is large. This is because mean can get distorted by very large or very small values, but the median is not affected by very large or very small numbers. Both the power law fitting line and the logarithmic binning fitting lines well represent the spectrum, however, the fitting lines from logarithm binning are better than the power law, since the power law fit is a little biased towards the near-clique ego-networks, particularly as the number of nodes increases.

Figure 2b shows the plot for the principal eigenvalue of weighted adjacency matrix versus the total weight, for ego-network corresponding to each node in the given email correspondence network. Similar to Figure 2a, this plot also has a positive slope which means that as the total weight of ego-network increases, the principal eigenvalue of weighted adjacency matrix also increases. Also, manifested in this plot is the fitting power law line. The slope of this line, that is the exponent of this power law, is 0.781, and the y-intercept of this fitting line is 0.036. This power law fit well represents the spectrum of principal eigenvalue versus total weight, and therefore, the plot corresponding to principal eigenvalue versus total weight for each ego-network is a power law distribution.

In addition, both the plots in Figure 2 demonstrate the top 20 out-of-the-norm nodes. In the first plot, most of these nodes lie on the line corresponding to the star structure. The top 20 out-of-the-norm nodes in both the plots represent those nodes whose ego-network parameters are most deviated from the fitting lines of the plots, compared to all other nodes in the network. In other words, the ego-network parameters of these nodes are most deviated from their expected values according to the power law fit and hence penalized the most with largest out-of-the-norm values. Points that are close to the out-of-the-norm points in these plots can also have large out-of-the-norm scores because of large deviation from their expected values, but the scores of these other points maybe less than the scores of top 20 nodes.

5 TASK 3: TEMPORAL NETWORKS

In this section we consider a temporal networks. Therefore, we create a sequence of graph snapshots, $G = G1, G2, ..., GT$, where $Vagr = \forall_\tau V_\tau$ and $Eagr = \forall_\tau E_\tau$, where $\tau = 1, ..., T$. Each snapshot represent a daily sample (excluding weekends). Considering our project dataset tuple format which are UnixEpoch time, Sender, and Receiver respectively. We have a tremendous data information to manipulate over time.

Before plotting the evolution of the size of the largest connected component(both in the weak and the strong sense), we need to make some changes in the dataset. First, we import the dataset in python3.5 by using pandas library. Then, we are able to convert the dataset as a numpy array. We can actually import the dataset by using other methods but as we know that using pandas read_csv method has a lot of advantages: faster, less CPU usage compare with others. In consequence of working on a large dataset, we must regard faster method in our program.

In the Figure 3 and Figure 4 show that the evolution of the largest strongly connected components and the evolution of the largest weakly connected components respectively. We consider our graph is a directed graph while working on the strongly connected components. At the same time, we have different story for weakly connected components because it is not possible to find weakly connected components in directed graph. we should convert the graph from directed to undirected. However, Networkx library can find out both strongly and weakly connected components for us. After excluding weekends, we have 936 days and we create a graph snapshots for each day $m(1 \leq m \leq 936)$ by using graphSnapShots method in our program.

In the Figure 5, we have plotted density of each G_τ over τ. In particular,for a simple graph, the graph density is defined as

$$D = \frac{2|E|}{|V|(|V| - 1)}$$

Where $|E|$ is the number of edges and $|V|$ is the number of nodes in the graph.

For clustering coefficient, we consider the number of edges among the neighbors of the node divided by the total number of possible edges among the neighbors of the node. Triadic closure increases the clustering coefficient that we have more edges or connection between common friends of the node.

5.1 Observation of Task-3

We have plotted the evolution of the strongly and weakly connected components in separate figure to observe the values clearly. Otherwise, the values of the largest strongly connected components cannot be observed. The reason is that when we look at the Figure 3, the higher size of the strongly connected component is around 300. At the same time in the Figure 4, the higher size of the weakly connected component is almost nearby 4000. This is actually what we want to observe in our result, because the weakly connected component graph is a undirected and the strongly connected component graph is a directed graph.

When we look at the Figure 5 which is density of the graph, it can be seen that during the initial days the density is more close 1. We can explain this situation with evolution of the largest weakly connected components graphs in figure 4. We can compare these two graph. The period, which is start almost from day 200 to day 800, the size of the nodes increase and we can observe that the density of graph decrease. Density is equal to total number of edges divided by total number of possible edges. Also, in figure 7 which can be shown that when the total number of nodes increase also the total number of edges increase and slope is equal to 1.109. However, if we look at the formula of density, the total number of possible edges increase with $n(n - 1)/2$. Therefore, it is normal to get this kind of graph based on our network dataset.

In the Figure 6 show that average clustering coefficient for each G_τ. We can easily say that first days average clustering coefficient is zero. its mean that on that days

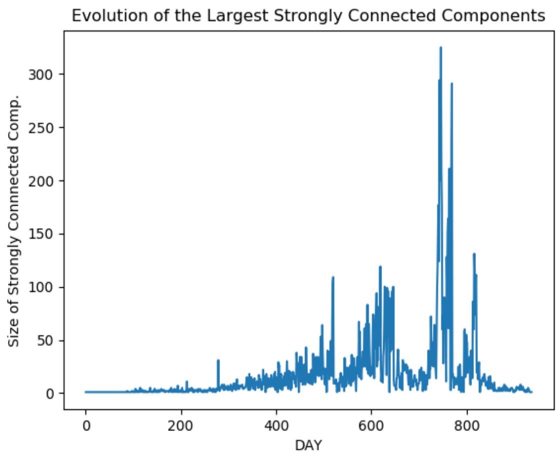


Figure 3: Evolution of the strong sense

6 EXTRA CREDIT: CHANGE-POINT ANALYSIS

In this section, we will discuss about the extra-credit task and its possible solution.

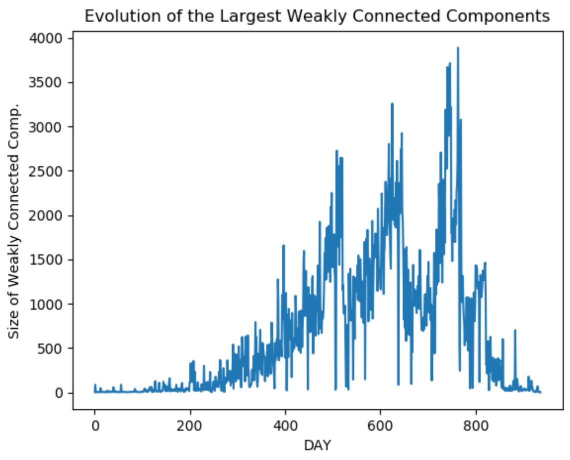


Figure 4: Evolution of the weak sense

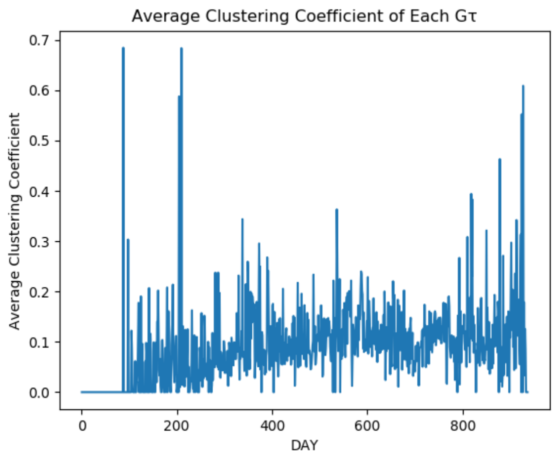


Figure 6: Average clustering coefficient

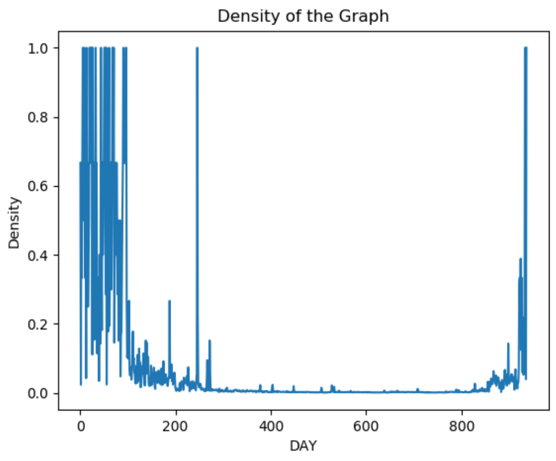


Figure 5: Density of the graph

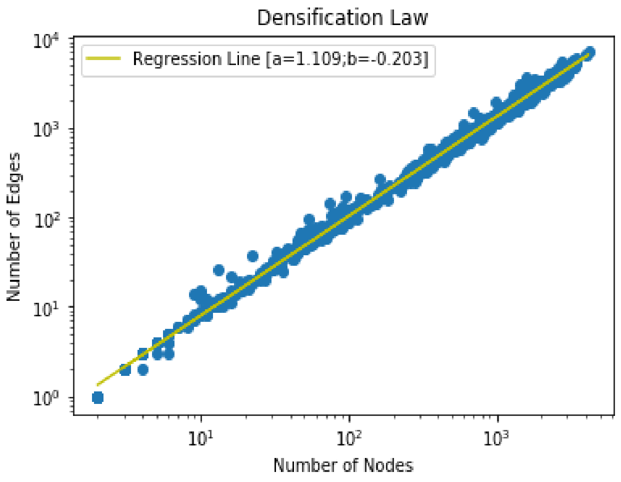


Figure 7: Densification of the graph

6.1 Problem Statement

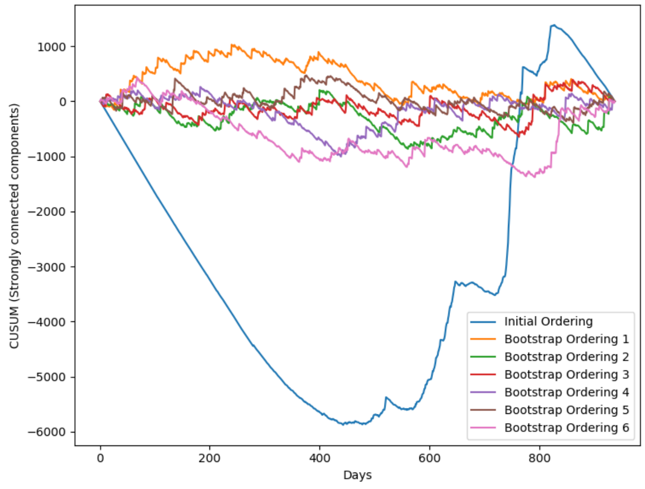
Given a series of network snapshots demonstrating the evolution of largest strongly connected components over time (that is, in days), as shown in Figure 3, the goal is to identify the time points at which the network snapshot changed significantly compared to its predecessor snapshot. That is, the presented approach should characterize the time stamps in decreasing order of significance with respect to the detected change between graph snapshots.

6.2 Solution

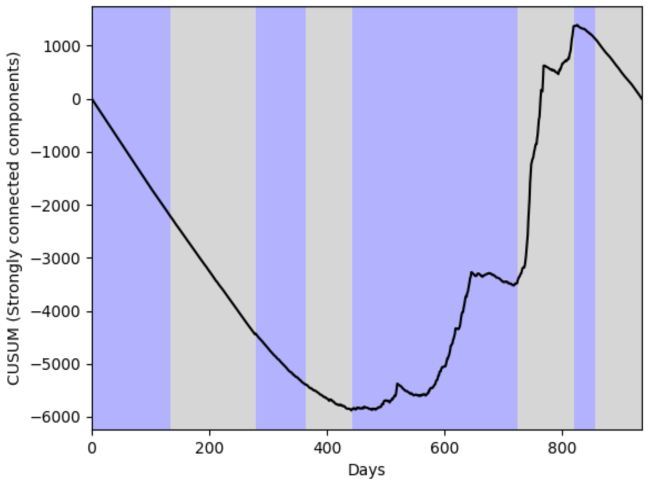
In this project, we use the flexible and powerful *change-point analysis* approach proposed by Taylor [10] for identifying whether a change has happened in a given historical data. Change-point analysis is capable of detecting multiple changes and suitable for large sets of data collected over time. Further, it is easier to interpret the detected changes, since this technique characterizes each detected change in terms of its timing, confidence and level. Confidence indicates the confidence of the analysis about the actual occurrence of a change, whereas level specifies the significance of a change.

There is a vast literature on change-point analysis [7–9]. Different approaches have been proposed for implementing this procedure. Compared to previous works, the approach proposed by Taylor, that is implemented in this project, combines two of the previously proposed approaches called *Cumulative Sum Charts (CUSUM) charts* and *Bootstrap analysis* in a novel fashion to achieve better results. Besides, in contrary to previous works, this approach is capable of detecting multiple changes.

Change-point analysis is based on the mean-shift model. That is, given a set of independent observations collected over time, the mean may shift at one or more time stamps. Formally, if X_1, X_2, ..., X_n represent historical data in time order, then the mean-shift



(a) Bootstrap samples versus original data



(b) CUSUM chart with significant changes in background

Figure 8: Manifestation of change-point analysis on the evolution of largest strongly connected components data

model can be mathematically given as:

$$X_i = \mu_i + \epsilon_i$$

Here, μ_i is the average at time i. The fundamental idea is that μ_i = μ_{i-1}, except for few time points which are referred to as the *change-points*. However, this approach makes an assumption that ϵ_i, which is the random error at time i, is independent and identically distributed with means of zero.

Table 4: Implementation Results

Day	Before Change	After Change	Level
443	6	38	1
279	3	31	2
821	111	21	2
134	1	2	3
364	5	17	3
725	23	72	3
857	8	2	3

6.3 Implementation

The first step involves creating a CUSUM chart, which is the cumulative sums of the differences between the data values and the average of all those values. This is calculated as follows:

$$S_i = S_{i-1} + (X_i - \bar{X})$$

where, X_i represent the data values, $\bar{X}$ is the average of all those values and $S_0 = 0$. The next step involves performing bootstrap analysis on the initial time-series data. Each bootstrap is a random re-ordering of the original data and then creating a CUSUM chart corresponding to that re-ordering. Each bootstrap specifies a CUSUM if no change had occurred, and hence will stay close to zero. Thus, the CUSUM generated from the original data can be compared with the CUSUM generated from bootstrap samples to determine how much the CUSUM of the original data varies compared to the case when no change occurred at all. Figure 8a shows that the initial ordering of the data varies largely compared to all five bootstrap samples, indicating that a change has definitely occurred in the original data.

To estimate the magnitude of the change, the range of CUSUM chart can be utilized which we will refer to as S_{diff}. Then, after performing a number of bootstrap samples, we can determine the number of samples where the initial ordering produced greater S_{diff} compared to the bootstrap. This will yield the confidence of the change. Usually, confidence of 0.9 is required before stating that an important change took place.

Once it is found that some important change has occurred, we need to calculate the timing of the change. For this, we calculate the point farthest from zero in the CUSUM chart of the original data. This point indicates the value before the change. The next point specifies the value after the change. Once a change has been detected, the original data can be split into two parts at the point of the change. Then, this procedure can be iteratively applied on each of the two segments of data to discover multiple changes.

Table 4 shows the results of performing the change-point analysis on the evolution of largest strongly connected components data. We created 1000 bootstrap samples and set the confidence threshold to be 1.0. The *Day* column indicates the day immediately after the change. *Before Change* and *After Change* are the sizes of largest strongly connected components before and after the change respectively. *Level* indicates the level or significance of the change. Figure 8b shows the significant changes by varying the background colors, which also correspond to changes in the slope of CUSUM chart of original data. The code for implementation is in *extra_credit.py* and *drawGraph.py* files.

6.4 Advantages

Change-point analysis is a flexible and powerful procedure for identifying subtle sustained changes in data that is collected over time. Apart from individual values, this approach can be applied to any characteristic of a time-series data such as range, standard deviations, averages, etc. Moreover, results of change-point analysis are easier to interpret, particularly when multiple changes have taken place and the data sets are large.

Change-point analysis is capable of detecting multiple changes and, even though some of them may be subtle, it categorizes each of the detected changes based on multiple parameters like confidence, level and the time of the change. Moreover, this procedure control the change-wise error rate instead of point-wise error rate. Consequently, there is high confidence that a detected change is true, thus reducing the number of false positives.

6.5 Limitations

Although change-point analysis is a powerful technique for detecting changes, even the subtle ones, along with its characteristics like the confidence and the level of the change, it suffers from some drawbacks. Autoregressive time series data such as stock prices are not suitable for a change-point analysis. This is because in autoregressive processes, the future values are affected by past values, but the random errors associated with each of the time values are assumed to be independent and identically distributed with means of zero under change-point analysis technique. Furthermore, although the CUSUM charts are optimal in identifying shifts of the mean, they do not discover isolated abnormal points. Besides, the bootstrap analysis may yield different results each time because of random reordering of the initial time-series data.

6.6 Applications

In this project, we demonstrated the application of change-point analysis on the strongly connected components of email correspondence network collected over time. This procedure of discovering multiple and subtle sustained changes can also be employed to ill-behaved data that has many outliers and does not follow any of the conventional data distributions. Some examples of scenarios where a change-point analysis can be utilized for identifying significant changes in large sets of historical data are as follows:

- **Investigating fluctuation of complaints:** Lets suppose a company collects complaint data from its customers. The complaint data was zero initially, but suddenly the complaint rate magnified over a course of time, and then started to decrease gradually. In this case, a change-point analysis could help in reviewing the cause for the sudden increase in complaint rate by providing a better interpretation of the number and timing of changes in the complaint rate.
- **Examining the quality of products:** Suppose that we have a set of part-strength data, that indicates the quality of some product by accessing the strength of its parts. If there is a sudden decrease in the part-strength data, then a change-point analysis could be employed to investigate the cause for the declining quality of product.
- **Studying the pattern of trade deficit:** Another major application of change-point analysis is in studying the cause for significant changes in the trade deficit data, by analyzing the exact timing for each of the numerous changes.

7 CONCLUSION

8 WORK DISTRIBUTION

- Padmavathi Iyer: Solving the second task and extra credit, and writing the sections 1, 4, and 6 of the report.
- Younes Karimi: Solving the first task and writing the sections 2, 3, and 8.
- Emre Ugurlu: Solving the third task and writing the abstract and the sections 5 and 7.

REFERENCES

[1] [n. d.]. datetime. https://docs.python.org/2/library/datetime.html
[2] 1972. UNIX Epoch Time. https://en.wikipedia.org/wiki/Unix_time
[3] 2003. Matplotlib. https://matplotlib.org/
[4] 2005. NetworkX. https://networkx.github.io/documentation/stable/
[5] 2006. NumPy. https://docs.scipy.org/doc/numpy/
[6] 2017. pandas. http://pandas.pydata.org/
[7] David Hinkley and Edna Schechtman. 1987. Conditional bootstrap methods in the mean-shift model. *Biometrika* 74, 1 (1987), 85–93.
[8] David V Hinkley. 1971. Inference about the change-point from cumulative sum tests. *Biometrika* 58, 3 (1971), 509–523.
[9] AN Pettitt. 1980. A simple cumulative sum type statistic for the change-point problem with zero-one observations. *Biometrika* 67, 1 (1980), 79–84.
[10] Wayne A. Taylor. 2000. Change-Point Analysis: A Powerful New Tool For Detecting Changes. http://www.variation.com/cpa/tech/changepoint.html