

**Large-Scale Social and Complex Networks: Design and Algorithms**  
**ECE 232E Summer 2018**  
Prof Vwani Roychowdhury  
UCLA, Department of ECE

## **Project 4**

### **Graph Algorithms**

Due on Monday, August 20, 2018 by 11:59 pm

#### **Team Members**

Jennifer MacDonald, UID: 604501712  
Nguyen Nguyen, UID: 004870721  
Sam Yang, UID: 604034791

#### **Introduction**

In this project, we explored and studied different graph theories and algorithms by applying them in real data such as the stock market and Uber Movements data. In the first part of the project, we looked at a graph that models correlations between stock price time series. In the second part of the project, we analyzed data from Uber that represents traffic in San Francisco.

#### **1. Stock Market**

In this part of the project, we retrieved the raw stock dataset from the provided link: [https://www.dropbox.com/s/83160htndqpn3fv/finance\\_data.zip?%20dl=0](https://www.dropbox.com/s/83160htndqpn3fv/finance_data.zip?%20dl=0). The goal of this part of the project is to study the correlation structures among fluctuation patterns to mimic how investors study the market before investing in certain stocks. As an example, the stocks corresponding to the transportation sector may have different absolute prices but if the fuel prices changed a lot, then the investors would buy or sell a lot of stocks in that area. So, we created different graphs based on similarities among the time series and returns on stocks at different times. The data we used were from Yahoo Finance going back for 3 years, and is organized in a number of tables that contain the following fields: Date, Open, High, Low, Close, Volume, and Adj Close price.

To preprocess the data, we downloaded it and put the finance\_data folder in our current directory. We then went through all the company csv files in the directory, and calculate  $q$  and  $r$

for each ticket. We then took the names of the companies and their correlation score and compiled it into a text file that we called correlation.txt. To create a dataset that contained only data from Mondays, we went through all the csv files again and removed the rows that had non-Monday data and repeated the process, saving the file as monday\_correlation.txt.

## 1. Return correlation

For the first part of the project, we had to calculate the correlation between log-normalized stock-return time series data using the following variables:

- $p_i(t)$  is the closing price of stock  $i$  at the  $t$ th day
- $q_i(t)$  is the return of stock  $i$  over a period of  $[t - 1, t]$

$$q_i(t) = (p_i(t) - p_i(t - 1)) / (p_i(t - 1))$$

- $r_i(t)$  is the log-normalized return stock  $i$  over a period of  $[t - 1, t]$

$$r_i(t) = \log(1 + q_i(t))$$

We can use this to generate a formula for the correlation between the log-normalized stock-return time series data of stocks  $i$  and  $j$  as seen below:

$$\rho_{ij} = \frac{\langle r_i(t)r_j(t) \rangle - \langle r_i(t) \rangle \langle r_j(t) \rangle}{\sqrt{(\langle r_i(t)^2 \rangle - \langle r_i(t) \rangle^2)(\langle r_j(t)^2 \rangle - \langle r_j(t) \rangle^2)}}$$

Where  $\langle \cdot \rangle$  represents a temporal averages for the time given.

*QUESTION 1: What are upper and lower bounds on  $\rho_{ij}$ ? Provide a justification for using log-normalized return ( $r_i(t)$ ) instead of regular return ( $q_i(t)$ ).*

The upper bound and lower bound on  $\rho_{ij}$  is +1 and -1 respectively. 0 indicates no relationship at all, which +1 indicates a perfect relationship, and -1 indicates a perfect negative relationship. For example, when  $r_i(t)$  and  $r_j(t)$  are independent, then the numerator is 0 such that  $\langle r_i(t)r_j(t) \rangle = \langle r_i(t) \rangle \langle r_j(t) \rangle$ . However, when  $r_i(t)$  and  $r_j(t)$  are correlated, we can assume that  $r_i(t)$  is some  $\zeta$  coefficient away from  $r_j(t)$  such that  $r_i(t) = \zeta r_j(t)$ . Thus,  $\langle r_i(t)r_j(t) \rangle = \zeta \langle r_j(t)^2 \rangle$  and  $\langle r_i(t) \rangle \langle r_j(t) \rangle = \zeta \langle r_j(t) \rangle^2$ . After simplifying,  $\rho_{ij} = \zeta / \zeta^{1/2}$  such that  $\zeta$  is equal to 1 when positive, and equal to -1 when negative.

Lognormal distribution differs from the normal distribution such that a normal distribution is symmetrical while a lognormal one is not. Because the values in a lognormal distribution are positive, they can be skewed to the right. The skewness is important in determining which distribution is appropriate to use in investment decision-making. For example, stocks grow at compounded rate and typically an investor looks at the growth factor by taking the current stock price and multiply it by various rates of return. When the investor looks at continuously compounds the returns, it is essentially a lognormal distribution. It does not make sense for the stock prices to fall below \$0 since the future stock price will always be positive.

## 2. Constructing correlation graphs

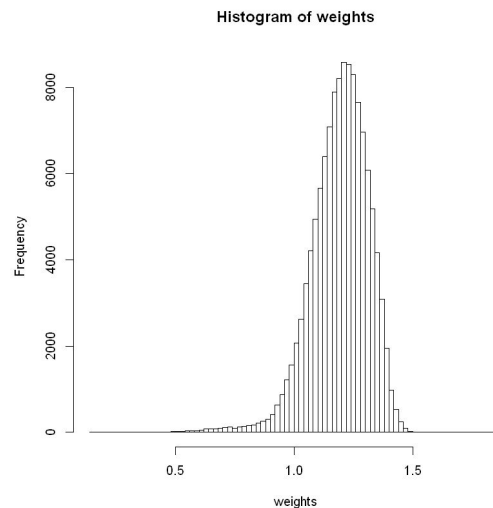
In this part we used the correlation coefficient we just computed to create a correlation graph that has the nodes and edge weights, with the edge weight being calculated using the following expression:

$$w_{ij} = \sqrt{2(1 - |\rho_{ij}|)}$$

*QUESTION 2: Plot a histogram showing the un-normalized distribution of edge weights.*

We read the correlation data and captured the values of the weights from the equation above, which we then plotted as as a histogram, seen below in Figure 1.

Figure 1: Histogram of weights



As we can see above, the weight values create a unimodal distribution between 0.5 and 1.5, with the frequency reaching over 8000 at its highest point. There doesn't appear to be any outliers in the distribution, and the values skew right towards the higher weights.

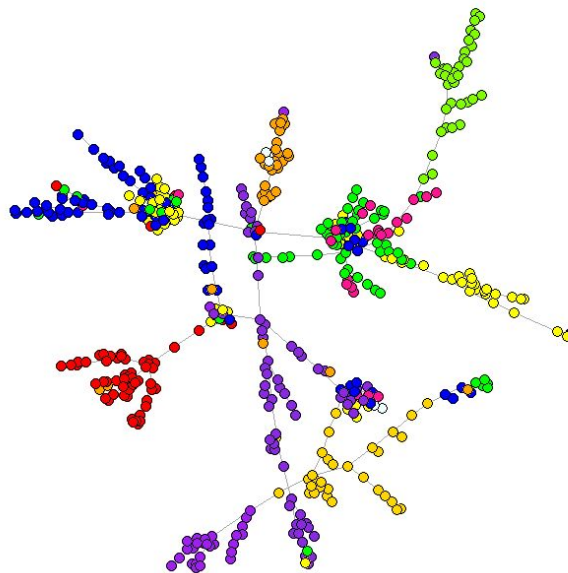
### 3. Minimum spanning tree (MST)

We also extracted the minimum spanning tree of the correlation graph.

*QUESTION 3: Extract the MST of the correlation graph. Each stock can be categorized into a sector, which can be found in Name sector.csv file. Plot the MST and color-code the nodes based on sectors. Do you see any pattern in the MST? The structures that you find in MST are called Vine clusters. Provide a detailed explanation about the pattern you observe.*

We split data into its individual sectors and assigned a color to each one and then created a minimum spanning tree of the graph with the different sectors color-coded. This graph can be seen below in Figure 2. The pattern we observed, as expected, is the Vine cluster. As we can see, the sectors tend to be grouped together on the MST, although this is not always the case.

Figure 2: Minimum Spanning Tree of the Correlation Graph for Daily Data



The minimum spanning tree (MST) is a subset of the edges of a connected edge-weighted undirected graph that connects all the vertices together without any cycles and with the minimum possible total edge weight. Therefore in Figure 2 above, the nodes which represent each individual stock tend to cluster lowest weights together which create this Vine cluster visualization.

#### 4. Sector clustering in MST's

In this portion, we turned our attention to predicting the market sector of an unknown stock. We evaluate the performance of our calculations by using the metric

$$\alpha = \frac{1}{|V|} \sum_{v_i \in V} P(v_i \in S_i)$$

Where  $S_i$  is the sector of node  $i$ . We also defined

$$P(v_i \in S_i) = \frac{|Q_i|}{|N_i|}$$

Where  $Q_i$  is the set of neighbors of node  $i$  that belong to the same sector as node  $i$  and  $N_i$  is the set of neighbors of node  $i$ . Compare  $\alpha$  with the case were

$$P(v_i \in S_i) = \frac{|S_i|}{|V|}$$

*QUESTION 4: Report the value of  $\alpha$  for the above two cases and provide an interpretation for the difference.*

To calculate the alpha value for the two cases, we counted the number of jobs in each sector and then for each node ID we looked at each of the neighbors. If the set of neighbors of that node belonged to the same sector of the original, then we incremented the  $Q_i$  value and added the value over the total to the alpha value for the first case. For the second, we calculated the sum of

the number of nodes in each sector over the total number of nodes. The alpha values for the two cases can be seen in Table 1 below.

Table 1: Alpha values for two different values of  $P(v_i \in S_i)$

$ Q_i / N_i $	$ S_i / V $
0.112	0.114

As you can see, there is a slight difference between the two cases. This is because the first calculation only looks at the neighbors of each node, whereas the second looks at more nodes because it looks at the probability of different nodes being in the same sector. Since it looks at nodes other than the neighbors, it has a slightly higher alpha value.

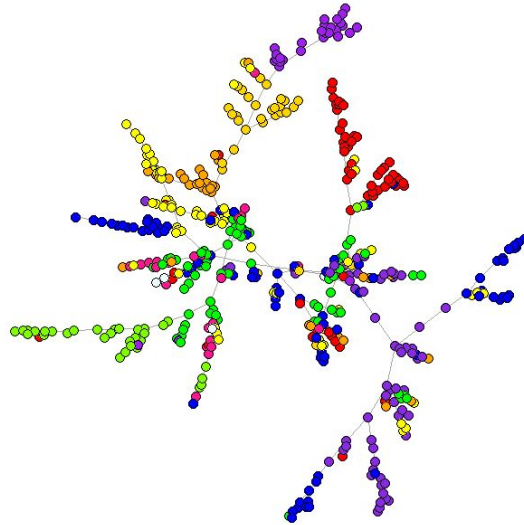
## 5. Correlation graphs for weekly data

Then we created a different correlation graph, but this one based on the weekly data. To create it, we isolated the data weekly on Mondays and then calculated  $\rho_{ij}$ , ignoring the weeks where a holiday fell on a Monday.

*QUESTION 5: Extract the MST from the correlation graph based on weekly data. Compare the pattern of this MST with the pattern of the MST found in Question 3.*

In this analysis, we're looking at the weekly data which means that we are filtering only Monday stock values. Again, we extracted the MST from correlation graph similar to our daily analysis above. We observed that the graph still exhibits the Vine clusters. Though, the clusters are less profound such that the nodes in weekly data are more separated than the ones in daily data.

Figure 3: Minimum Spanning Tree of the Correlation Graph for Weekly Data



## 2. Let's Help Santa!

In order to “help” Santa with his gift deliveries on Christmas Eve, we were to use a transportation dataset from Uber that contains information about traffic in San Francisco.

### 1. Download the Data

To obtain the dataset, we went to the “Uber Movement” website at <https://movement.uber.com/?lang=en-US>, clicked on the “Explore City Data” button, and selected San Francisco. We then clicked on the downloading data symbol, and clicked on “All Data” in the header. We then selected the quarter “2017 Quarter 4” from the dropdown and clicked on the the file “Monthly Aggregate (all days). The downloaded file was named `san_francisco-censustracts-2017-4-All-MonthlyAggregate.csv`. This dataset contains the time statistics between points in San Francisco, which are represented by unique IDs. To get the actual areas corresponding to the IDs, we also downloaded the Geo Boundaries data by clicking on “Geo Boundaries” in the header, agreeing to the licensing terms and downloading the file

SAN\_FRANCISCO\_CENSUSTRACTS.JSON. This file has the latitudes and longitudes of the corners of each area under DISPLAY\_NAME.

## 2. Build Your Graph

We used the dataset downloaded and took the data corresponding to December only, and removed the duplicates and keep the mean of duplicate travel times. The nodes represented the locations in the dataset and the undirected edges represented the mean traveling times. To each node, we added:

1. Display name: the street address
2. Location: mean of the coordinates of the polygon's corners (a 2-D vector)

Since the graph will contain isolated nodes from the locations in the Geo Boundaries data that does not exist in the December data, we isolated just the giant connected component and merged the duplicate edges by averaging the weights. We also went through each entry in the Geo Boundaries and retrieved the coordinate list and computed the mean coordinates. We named this resulting graph G.

*QUESTION 6: Report the number of nodes and edges in G.*

We counted the number of nodes and edges in G, as seen below in Table 2.

Table 2: Number of nodes and edges in G

Number of nodes	Number of edges
1898	321703

## 3. Traveling Salesman Problem

*QUESTION 7: Build a minimum spanning tree (MST) of graph G. Report the street addresses of the two endpoints of a few edges. Are the results intuitive?*

We built the minimum spanning tree of the new graph G we built, as seen below in Figure 4.

Figure 4: Minimum spanning tree of the graph G corresponding to December only data

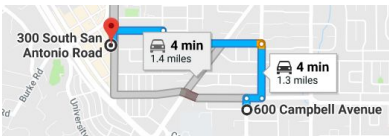
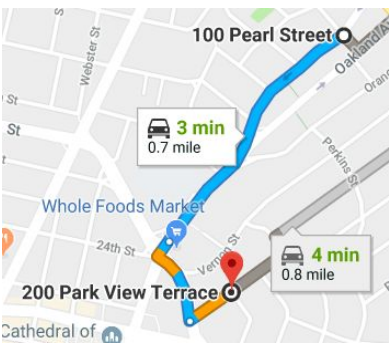




As a sanity check, when we randomly selected a few of the endpoint pairs to test, they all were in the same city, which is a good sign that the data makes sense. To further check that the data we produced is correct, we took the street addresses of the two endpoints of a few edges and used Google Maps to check whether the endpoints are intuitive, as seen below in Table 3.

Table 3: Checking intuition of endpoint street addresses

Starting endpoint address	Ending endpoint address	Google Maps
1000 Hayes Street, Fairfield	200 Beck Avenue, Fairfield	

600 Campbell Avenue, Los Altos	300 South San Antonio Road, North Los Altos, Los Altos	
100 Pearl Street, Oakland Ave - Harrison St, Oakland	200 Park View Terrace, Adams Point, Oakland	

Since the endpoints are all very close to each other, we concluded that yes, the results are intuitive.

*QUESTION 8: Determine what percentage of triangles in the graph (sets of 3 points on the map) satisfy the triangle inequality. You do not need to inspect all triangles, you can just estimate by random sampling of 1000 triangles.*

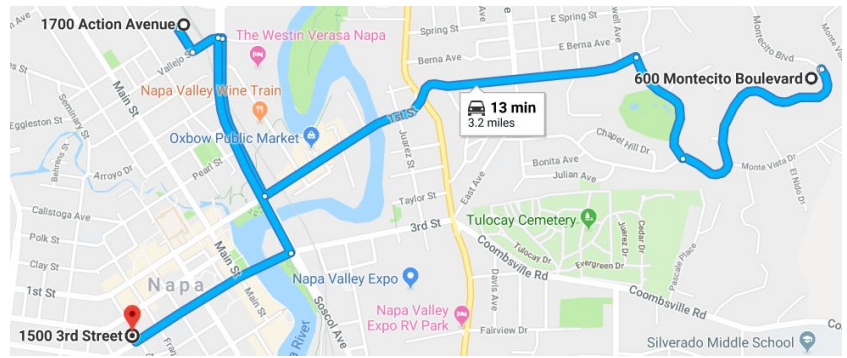
We randomly picked 1000 sets of 3 points on the map and checked to see if they fulfilled the triangle inequality, in which the sum of the lengths of any two sides of a triangle is greater than the length of the third side. We did this by getting the weights of how long the commute would take from one point to another on the triangle and performing the triangle inequality test. We found that about 95.3% of the triangles in the graph passed this test.

Now we can find a solution for the traveling salesman problem on  $G$  by applying the 1-approximate algorithm. We looked to see if these results were intuitive. To do this, we randomly took three separate areas and got three sequential locations to see if they were close to each other. The results are seen below in Table 4.

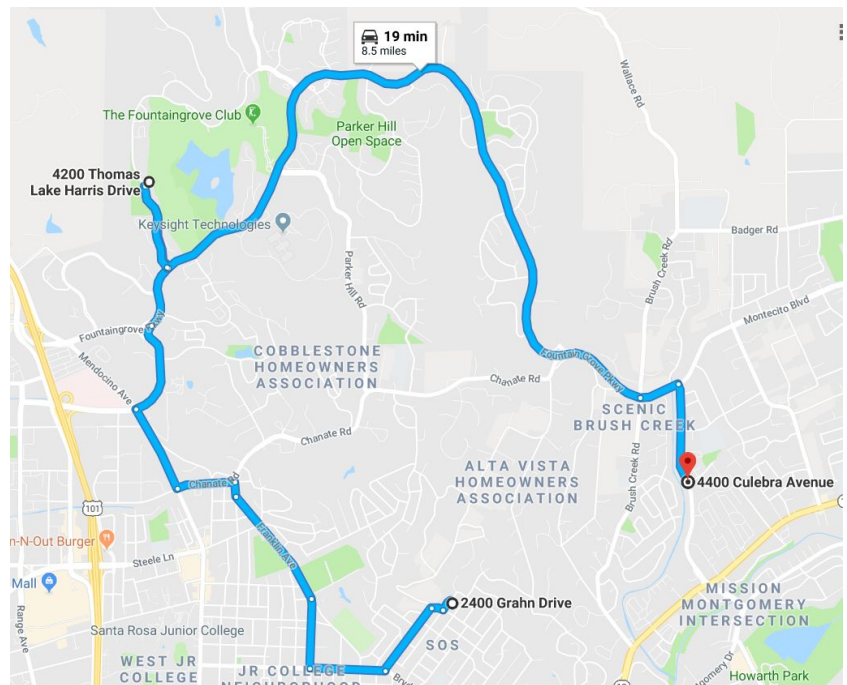
Table 4: Checking intuition of result of 1-approximate algorithm

Sequential addresses	Google Maps
----------------------	-------------

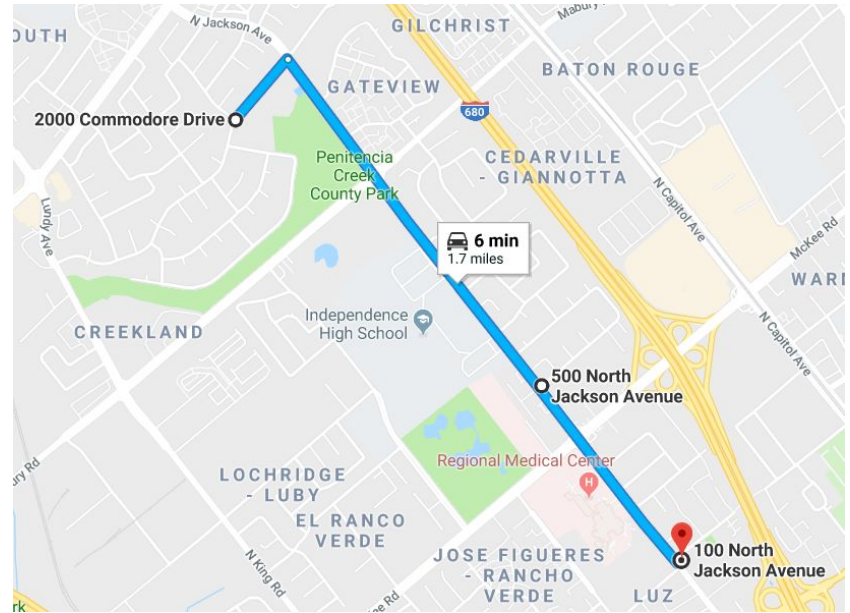
['600 Montecito Boulevard, Napa', '1700 Action Avenue, Napa', '1500 3rd Street, Napa']



['2400 Grahn Drive, SOS, Santa Rosa', '4200 Thomas Lake Harris Drive, Santa Rosa', '4400 Culebra Avenue, Santa Rosa']



['2000 Commodore Drive, East San Jose, San Jose', '500 North Jackson Avenue, East San Jose, San Jose', '100 North Jackson Avenue, East San Jose, San Jose']



We found our sequentially visited locations to be very close together, so we concluded that our results were very intuitive.

*QUESTION 9: Find an upper bound on the empirical performance of the approximate algorithm:*

$$\rho = \frac{\text{Approximate TSP Cost}}{\text{Optimal TSP Cost}}$$

To find the upper bound on the empirical performance of the approximate algorithm, we relied on finding the Eulerian spanning tree and one of the corresponding embedded tours. The steps are followed:

1. Find the minimum spanning tree  $T$  under  $[d_{ij}]$ .
2. Create a multigraph  $G$  by using *two copies* of each edge of  $T$ .
3. Find an Eulerian walk of  $G$  and an embedded tour.

The first step is simple in which we created the minimum spanning tree from the original Uber dataset. Then in step two, we created a graph that has two copies of each edge. For example, if there is a edge from A-B, we just simply added B-A. In the undirected graph, both edges are the same. In the third step, we used our Eulerian walk implementation.

The Eulerian walk is an algorithm such that each node appears at least once and each edge appears exactly one. This is a classical Traveller Salesman Problem (TSP) which is a NP-hard problem. Typically in a connected graph, it performs a DFS traversal (depth first search) to visit

all the nodes, which is essentially visiting every node twice. However, TSP's restriction such that it only allows visiting every node once which requires some deletion of repeated visits. The algorithm then recursively go through the graph until no more node is visited more than once. The following is a pseudocode of our implementation for the euler walk.

```

procedure Euler( $v_1$ )
  (comment: it returns an Eulerian walk of the connected
    component of  $G$  containing  $v_1$ )
  begin
    if  $v_1$  has no edges then return [ $v_1$ ] (comment: the empty walk)
    else
      begin
        starting from  $v_1$  create a walk of  $G$ , never visiting the
        same edge twice, until  $v_1$  is reached again;
        let [ $v_1, v_2, \dots, v_n, v_1$ ] be this walk;
        delete [ $v_1, v_2$ ],  $\dots$ , [ $v_n, v_1$ ] from  $G$ ;
        return [Euler( $v_1$ ), Euler( $v_2$ ),  $\dots$ , Euler( $v_n$ ),  $v_1$ ]†
      end
    end
  end

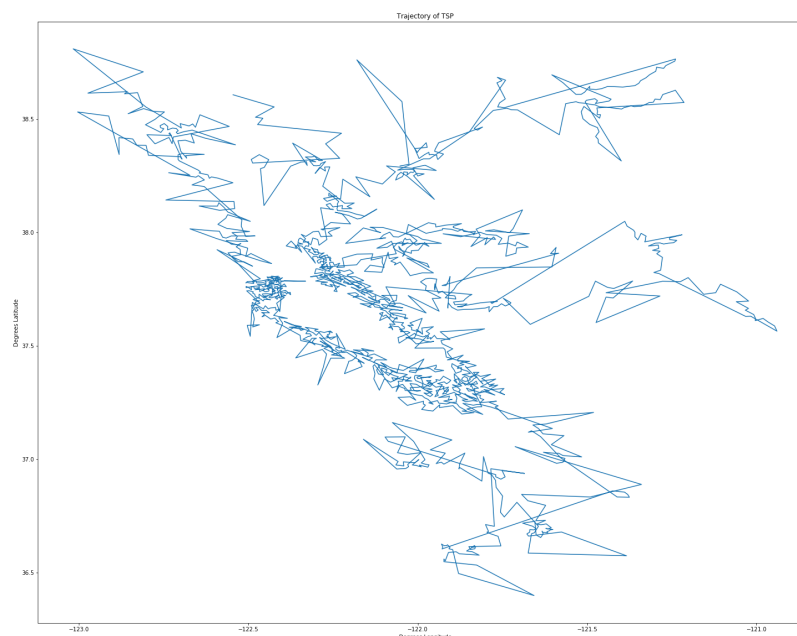
```

We observed that the minimum total weight is 479841.66. Likewise, the upper bound is determined by the lower bound denominator. Therefore, we divided our minimum total weight from our Eulerian graph by our sum of the weight our minimum spanning tree. We found that  $\rho = 1.658$ .

*QUESTION 10: Plot the trajectory that Santa has to travel!*

We took the data we had created in the previous problem and plotted the trajectory that Santa has to travel, as seen below in Figure 5.

Figure 5: Trajectory that Santa has to travel through the San Francisco Bay Area



#### 4. Analysing the Traffic Flow

The project then asked us to analyze a scenario in which we had to find the maximum traffic between Stanford to the University of California, Santa Cruz (UCSC) for a sporting event held at UCSC next December. Stanford fans are driving from their campus in Stanford to UCSC's campus in Santa Cruz.

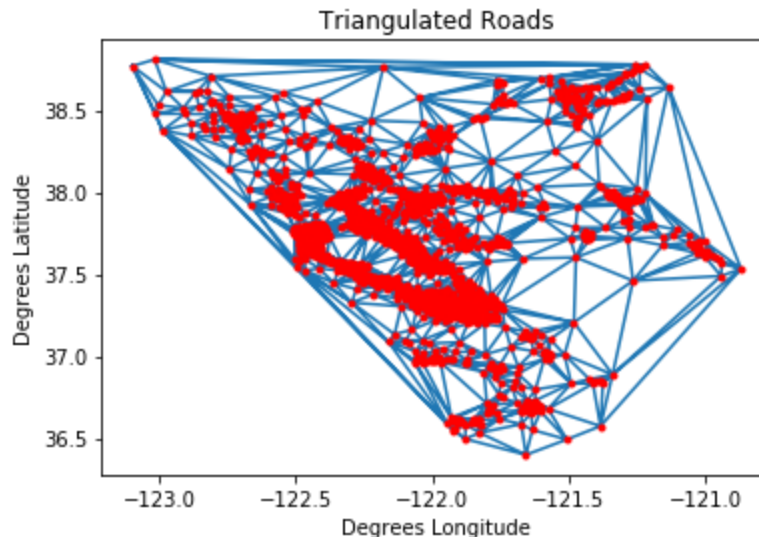
#### 5. Estimate the Roads

First, we were asked to estimate the roads by using the Delaunay triangulation algorithm and apply it to the nodes rather than using the data from the datasets we downloaded.

*QUESTION 11: Plot the road mesh that you obtain and explain the result. Create a graph  $G_{\Delta}$  whose nodes are different locations and its edges are produced by triangulation.*

Delaunay triangulation is an algorithm that takes a set of points and discovers connecting triangles with the condition that no point lies in the circumcircle of any derived triangle. We used scipy's implementation of Delaunay triangulation on the vertices in  $G$  (the cleaned graph), and plotted the vertices and edges on a flattened latitude/longitude plane:

Figure 6: Results of Delaunay Triangulation on Locations In Cleaned Graph ( $G$ )



#### 6. Calculate Road Traffic Flows



*QUESTION 12: Using simple math, calculate the traffic flow for each road in terms of cars/hour. Report your derivation.*

**Hint:** Consider the following assumptions:

- Each degree of latitude and longitude  $\approx 69$  miles
- Car length  $\approx 5 \text{ m} = 0.003 \text{ mile}$
- Cars maintain a safety distance of 2 seconds to the next car
- Each road has 2 lanes in each direction

*Assuming no traffic jam, consider the calculated traffic flow as the max capacity of each road.*

Using the given values of 69 miles per degree latitude and longitude, 0.003 mile car length, 2 second safety distance, and two lanes in each direction, we can derive the traffic flow for each road in terms of cars per hour.

This measurement can be derived by first calculating the average speed  $v$  of each car on the road. This speed  $v$  can be calculated by taking the length  $L$  of the road, and dividing by the time  $t$  that is provided in the dataset.

With these parameters, we can then figure out the amount of time that elapses between each car passing some point on the road. We first consider how much distance there is between each car (let's say front bumper of the first car to front bumper of the second car):

$$\Delta d = 0.003 + v\left(\frac{2}{3600}\right) \text{ miles, where } v \text{ is in miles/hour}$$

If we have a fixed “checkpoint” on the road, we can then calculate the number of time that elapses between each car passing that checkpoint. This can be calculated by:

$$\Delta t = \frac{\Delta d}{v} = \frac{0.0003 + v\left(\frac{2}{3600}\right)}{v} \text{ hours/car}$$

Finally, to get cars/hour, and add the fact that there are two lanes per road, we invert the above equation and multiply by 2:

$$\text{maxflow} = \frac{2v}{0.0003 + \frac{v}{1800}} \text{ cars/hour}$$

## 7. Calculate the Max Flow

We also looked at two addresses:

- Source address: 100 Campus Drive, Stanford
- Destination address: 700 Meder Street, Santa Cruz

*QUESTION 13: Calculate the maximum number of cars that can commute per hour from Stanford to UCSC. Also calculate the number of edge-disjoint paths between the two spots. Does the number of edge-disjoint paths match what you see on your road map?*

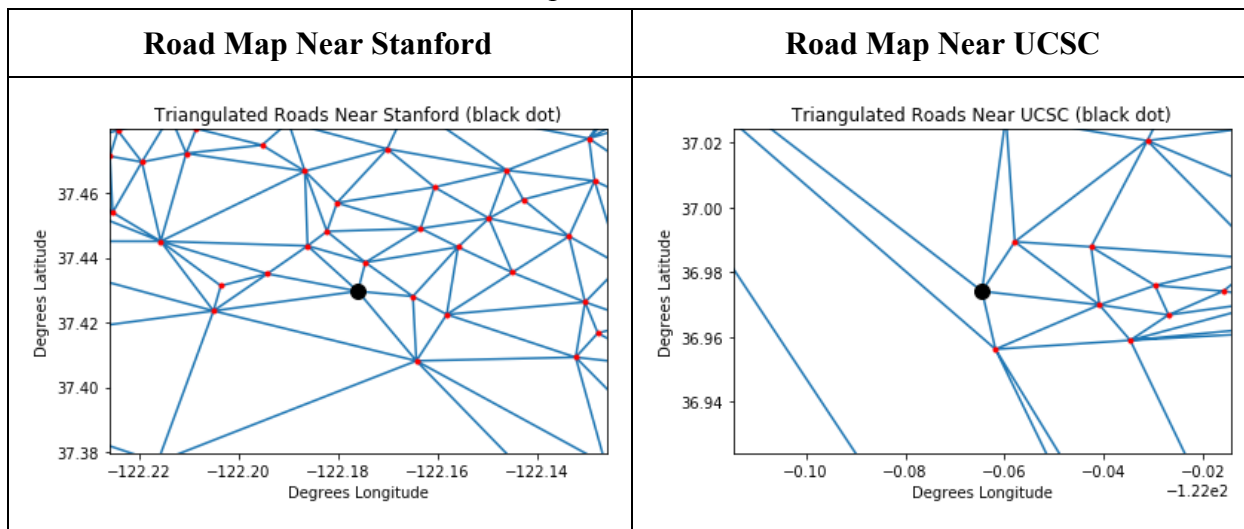
Using the Delaunay triangulated graphs and the location-to-location commute times provided by Uber, we compute the maximum flow between edges in the triangulated graph  $G_{\Delta}$ . We then use igraph's "max\_flow" and "edge\_disjoint\_paths" functions to compute the maximum flow and the number of edge-disjoint paths between Stanford and UC Santa Cruz.

Table 5: Max cars per hour from Stanford to UCSC and edge-disjoint paths

Maximum Flow from Stanford → UCSC	Number of Edge Disjoint Paths
14866.44 cars/hour	5

To verify whether the number of edge-disjoint paths matches what we see on our road map, we look at the zoomed in road map at both Stanford and UCSC. The black dot indicates the location of interest.

Table 6: Road maps around Stanford and UCSC



An edge-disjoint path is a path between the source in the sink that do not share any edges. These figures show that there are 6 roads that lead out of Stanford, and 5 roads that lead in. No matter



how many combinations of roads there may be between Stanford and UCSC, there can be a maximum of 5 edge-disjoint paths since there are only 5 ways to get into UCSC. The resulting calculation of edge-disjoint paths in igrph thus matches what we see on our roadmap.

## 8. Prune Your Graph

We then took graph  $G$  and removed the “unreal roads” from the graph. The unreal roads were considered to be roads like fake bridges, so to remove them we had to add a limit on the time travel of the roads in  $G_\Delta$  and remove everything else. We called this new graph  $\sim G_\Delta$ .

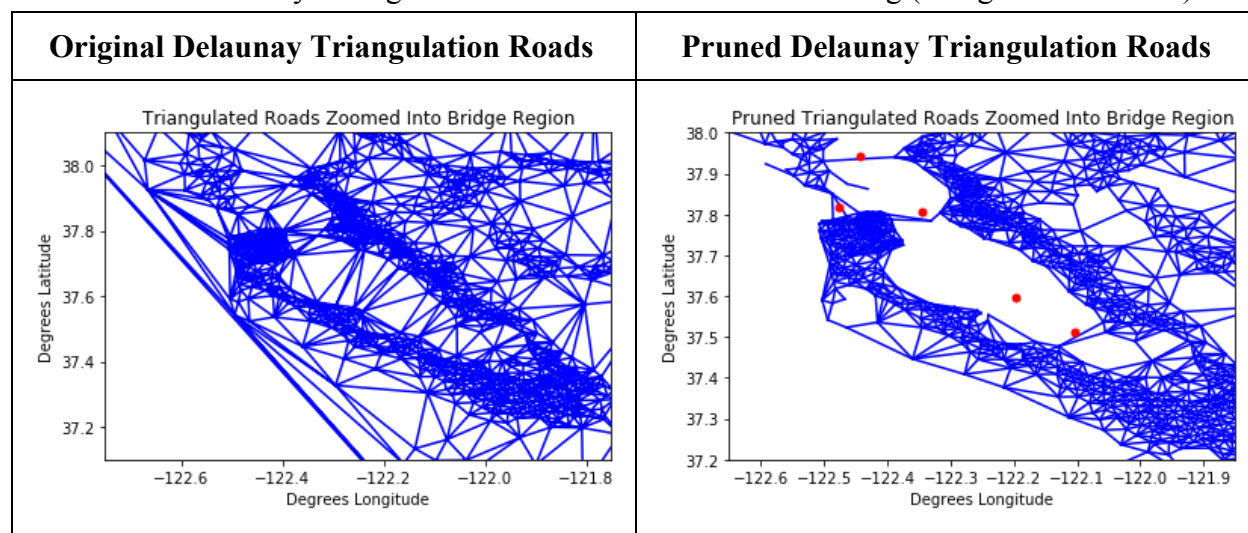
*QUESTION 14: Plot  $\sim G_\Delta$  on actual coordinates. Are real bridges preserved?*

*Hint: You can consider the following coordinates:*

- Golden Gate Bridge:  $[[ -122.475, 37.806], [ -122.479, 37.83]]$
- Richmond, San Rafael Bridge:  $[[ -122.501, 37.956], [ -122.387, 37.93]]$
- San Mateo Bridge:  $[[ -122.273, 37.563], [ -122.122, 37.627]]$
- Dambarton Bridge:  $[[ -122.142, 37.486], [ -122.067, 37.54]]$
- San Francisco - Oakland Bay Bridge:  $[[ -122.388, 37.788], [ -122.302, 37.825]]$

To remove most of the fake bridges while preserving most of the real ones, we pruned any roads from the Delaunay triangulation that took more than 12 minutes. This resulted in a removal of most fake bridges across the bay, at the cost of also removing the San Mateo bridge. We show a zoomed in plot of the bay before and after pruning:

Table 7: Delaunay Triangulated Roads Before and After Pruning (Bridges as Red Dots)



The reason we allowed for the pruning of the San Mateo Bridge is that when we set the threshold to be sufficiently high to allow the San Mateo Bridge to be put in, many other fake bridges appeared. A potential reason for why the San Mateo Bridge was pruned is that it appears to have the longest distance, thus leading to a longer commute time and potentially easier time being pruned.

*QUESTION 15: Now, repeat question 13 for  $\sim G_{\Delta}$  and report the results. Do you see any significant changes?*

After pruning most fake bridges (yet still preserving 4), we re-computed the maximum flow and the number of edge-disjoint paths between Stanford and UC Santa Cruz. We arrived at the same exact numbers as before:

Table 8: Max flow of cars/hour and edge disjoint paths before and after pruning

	Maximum Flow	# Edge Disjoint Paths
<b>Before Pruning</b>	14866.44 cars/hour	5
<b>After Pruning</b>	14866.44 cars/hour	5

To understand why this could be the case, we compare the unpruned and pruned graphs at the two locations.

Table 9: Delaunay Triangulated Roads Near Stanford Before and After Pruning

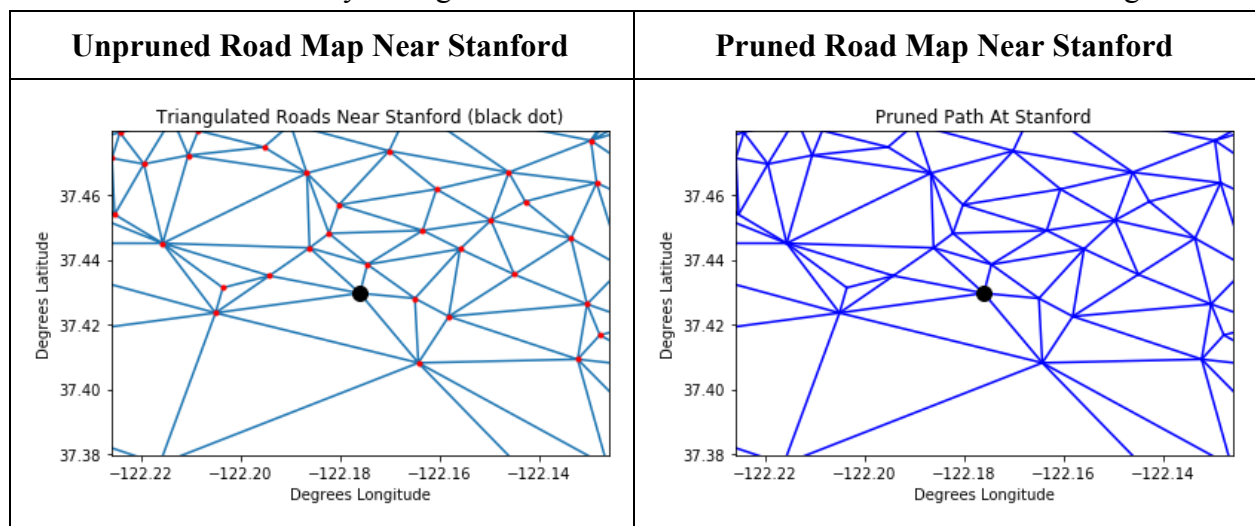
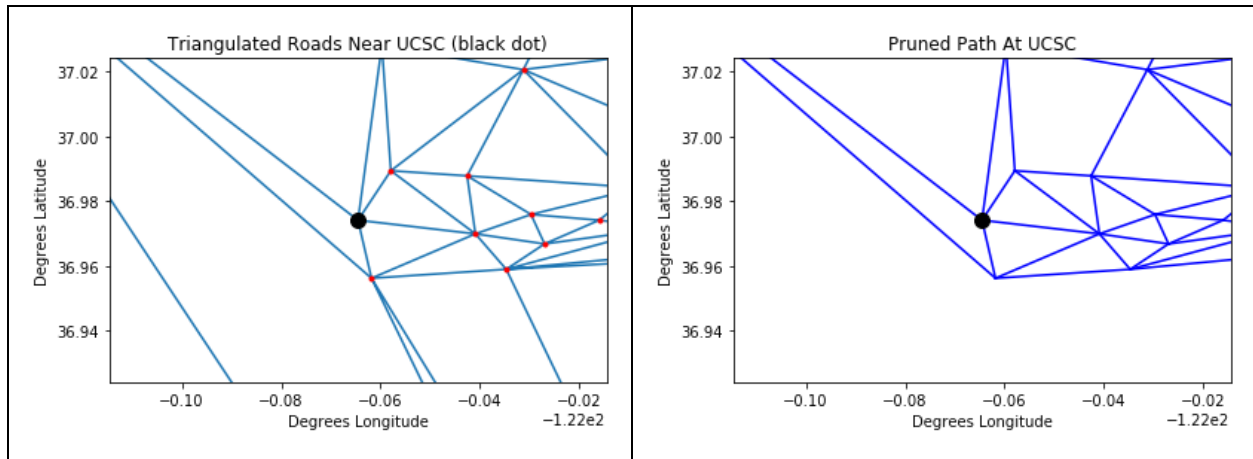


Table 10: Delaunay Triangulated Roads Near UCSC Before and After Pruning

Unpruned Road Map Near UCSC	Pruned Road Map Near UCSC
-----------------------------	---------------------------



The pruning had no effect on the roads near Stanford, as the points all look close together and the commute times between each point are either lower than the threshold of 12 minutes, or non-existent (perhaps people just don't Uber between locations that close).

As for UCSC, the pruning had some effect on the nearby roads, but not on any roads directly leading into UCSC. The same 5 roads that flowed into UCSC in the unpruned graph also exist in UCSC. This explains why even in the pruned graph, there are still 5 edge-disjoint paths. It is also reasonable to expect that the max flow may not change. If the bottle-necking roads between Stanford and UCSC still exist in the pruned path, then the removal of any other nodes will not have an effect on the maximum flow. Only the removal of bottlenecking roads will decrease the maximum flow between two destinations. Out of the 5,288 roads in the unpruned graph, 5074 remained in the pruned graph. The small percentage of roads that were pruned, along with the fact that the roads leading into UCSC are preserved, explain why we see identical values for maximum flow and number of edge-disjoint paths when compared to the unpruned graph.

As a sanity check that the maximum flow and the number of edge-disjoint paths can be affected by pruning at other locations, we ran the same analysis between two locations that should be significantly affected when a real bridge is removed. Since the San Mateo bridge was removed in our pruning, we can measure the maximum flow and the number of edge-disjoint paths between Foster City and Hayward to illustrate how removing roads should diminish flow.

Table 11: Max cars per hour from Foster City to Hayward and edge-disjoint paths before and after pruning

	Maximum Flow From Foster City to Hayward)	# Edge Disjoint Paths From Foster City to Hayward)
<b>Before Pruning</b>	9564.9 cars/hour	4
<b>After Pruning</b>	8232.75 cars/hour	3

Here we can clearly see the effects since we pruned the San Mateo Bridge (a bridge connecting Foster City and Hayward). The removal of the bridge diminished the maximum flow as well as the number of edge-disjoint paths between the two cities. We performed this analysis both to ensure that our result from #15 was not caused by a bug, and also to reinforce our understanding of how pruning roads should affect maximum flow.

## **Conclusion**

In this project, we learned how to apply graph theories on different real life applications. We studied how graphs can be applied to the stock market, and we studied how we can use an approximation algorithm on the Traveller Salesman Problem (TSP). Overall, this project is a good introduction to graph theories.