# Project 4: Graph Algorithms

University of California, Los Angeles

ECE 232 Large Scale Social and Complex Networks

Wayne Zhang
UID: 905845716

**All the code and reports are finished only by my myself. Many thanks to TAs and one of my classmates for providing valuable hints!**

# Contents

# List of Figures

# Question 1

Pearson correlation coefficient is given by $\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$. An important property of the coefficient is that $\rho_{X,Y} \in [-1, 1]$. A value of exactly 1.0 means there is a perfect positive relationship between the two variables. For a positive increase in one variable, there is also a positive increase in the second variable. A value of -1.0 means there is a perfect negative relationship between the two variables. This shows that the variables move in opposite directions-for a positive increase in one variable, there is a decrease in the second variable. If the correlation between two variables is 0, there is no linear relationship between them.

The strength of the relationship varies in degree based on the value of the correlation coefficient. For example, a value of 0.2 shows there is a positive correlation between two variables, but it is weak and likely unimportant. Analysts in some fields of study do not consider correlations important until the value surpasses at least 0.8. However, a correlation coefficient with an absolute value of 0.9 or greater would represent a very strong relationship.

There are mainly three reasons why we use log-normalised return $r_i(t)$ instead of regular return $q_i(t)$ to calculate the correlation coefficient $\rho_{ij}$:

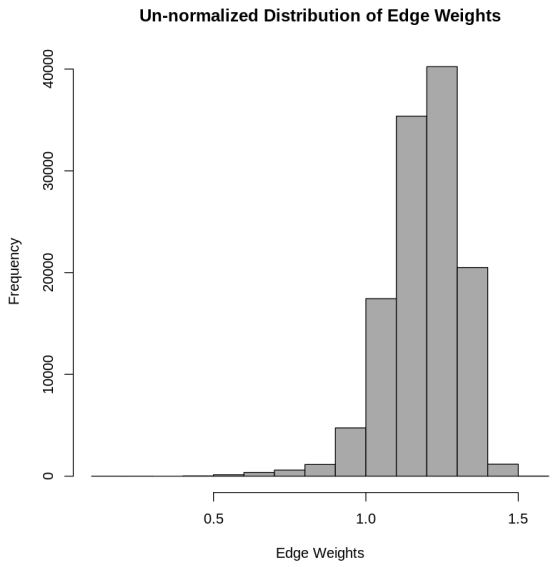
- When the stock price remains the same, $q_i(t) = r_i(t) = 0$. Since the closing price $p_i(t) \geqslant 0$, $q_i(t) \in [-1, +\infty]$. This means the positive and negative gain do not contribute evenly for the regular return. While $r_i(t) \in [-\infty, +\infty]$, showing a equal range of positive and negative gains.
- For a low fluctuation stock market, the daily regular return $q_i(t)$ can be a very small percentage around 0. By taking a log from the regular return, $r_i(t)$ better captures slight changes of the stock price.
- Log-normalization turns multiplication into addition, enabling a easier calculation of the closing price in a certain duration.

# Question 2

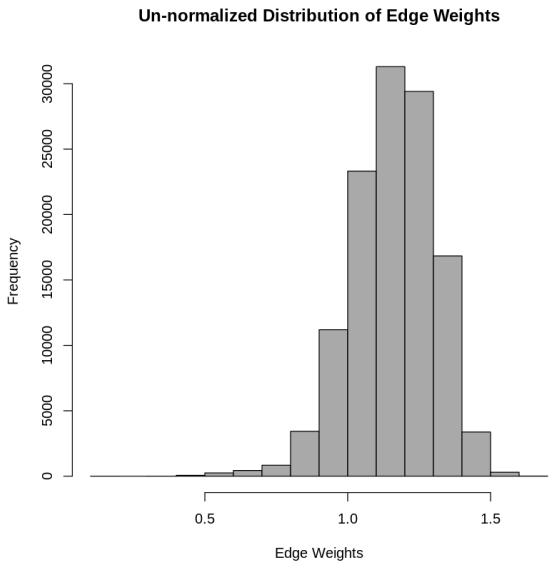
We compute the edge weights and plot its distribution as Figure 1(a). We know from the above discussion that:

- If the two stocks have a perfect positive linear relationship, $\rho_{ij} = 1$, then $w_{ij} = 0$
- If the two stocks have a perfect negative linear relationship, $\rho_{ij} = -1$, then $w_{ij} = 2$
- If the two stocks have no linear relationship, $\rho_{ij} = 0$, then $w_{ij} = \sqrt{2} \approx 1.41$

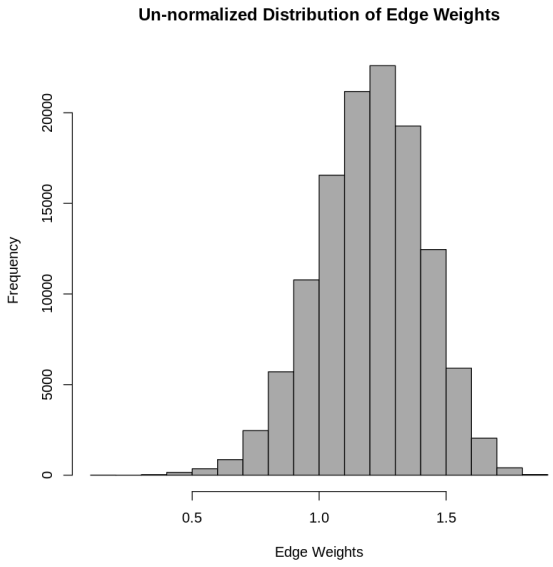
We can see from the edge distribution that most pairs of stocks have a weak positive relationship and no pair has a perfect linear relationship. This is a similar pattern to the real world where every stock is different but follows a largely similar trend, especially for stocks in the same sector.




(a) Daily




(b) Weweekly




(c) Monthly

Figure 1: Un-normalized Distribution of Edge Weights

## Question 3

We extract the minimum spanning tree (MST) of the graph and plot it in Figure 2(a), the color of the vertices indicate each stock's sector, which is the ground truth label for each vertex. The MST connects all the nodes of a connected and edge-weighted undirected graph without any cycles and with the lowest possible cumulative edge weights. We can see the vine cluster structure in the MST: most vertices on the same branch tend to have the same color. The reason for this structure is that, stocks from the same sector often have a similar fluctuation trend when facing external influences, so their prices are more positively related. From our discussion in Question 2 we know that this will result in a lower edge weight. Therefore, according to the Prim's algorithm that we use to construct the MST, they will be more likely to be clustered together and share the same branch.

## Question 4

We use the `cluster_walktrap` method provided by iGraph to find the community structure in MST. We plot the clusters in Figure 3(a) and calculate the homogeneity and completeness between the cluster we extracted and the ground truth label (sector of the company) in Table 1.

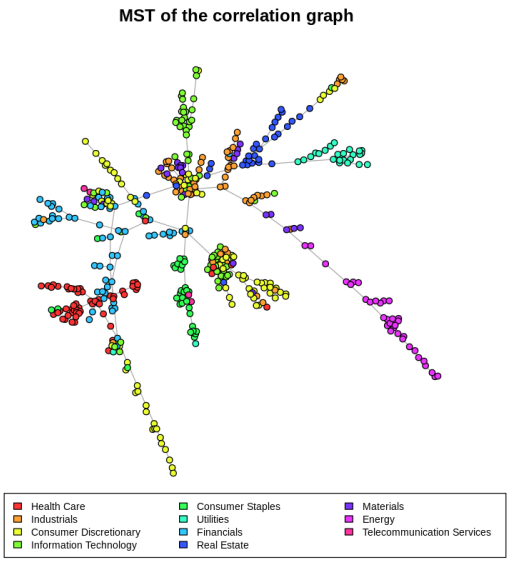
Table 1: Graph metrics for three time intervals

| Interval | Homogeneity | Completeness | $\alpha 1$ | $\alpha 2$ |
|---|---|---|---|---|
| Day | 0.68264 | 0.47928 | 0.82893 | 0.11419 |
| Week | 0.58112 | 0.39004 | 0.74395 | 0.11430 |
| Month | 0.47944 | 0.27755 | 0.48444 | 0.11430 |

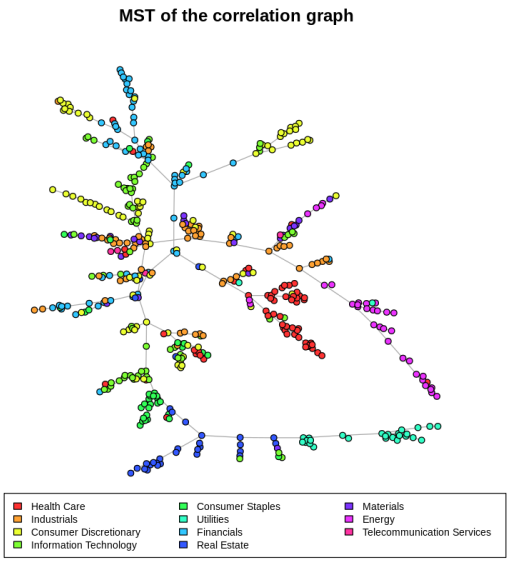
## Question 5

We calculate $\alpha$ by two definitions to evaluate the performance of sector clustering. The results are shown in Table 1. We provide an interpretation for each of the definitions:

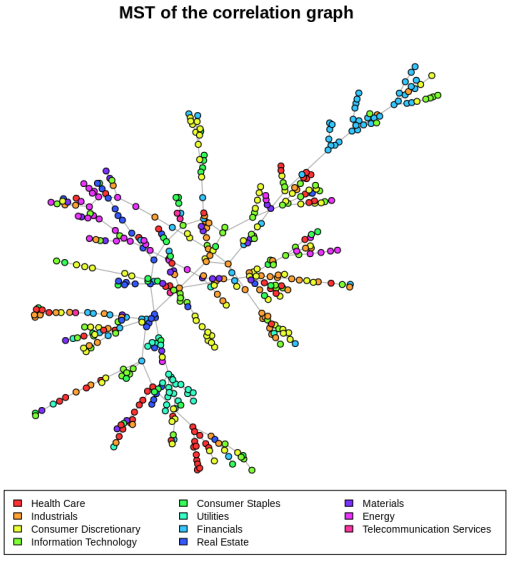
- For definition 1, we calculate the possibility that a certain vertex has the same sector as its neighbors. We can see that the $\alpha 1$ we get is close to 1 as expected. Because definition 1 exploits the MST vine cluster structures of the correlation graph, taking into account which nodes are highly correlated and flock together instead of all the nodes. In other words, technique 1 exploits local connectivity among neighboring nodes instead of the global correlation graph to make decisions.
- For definition 2, we calculate the possibility that any vertex in the graph might belongs to a certain sector. This definition considers all the vertices that belong to a sector and fails to extract local spatial connections or cluster formations. Definition 2 thus provides only a general probability estimation, its value $\alpha 2$ is also significantly lower.




(a) Daily

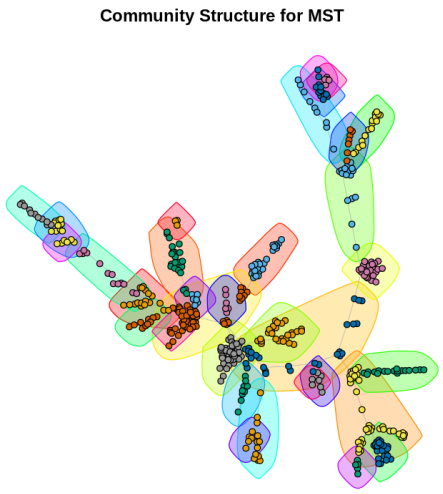



(b) Weekly

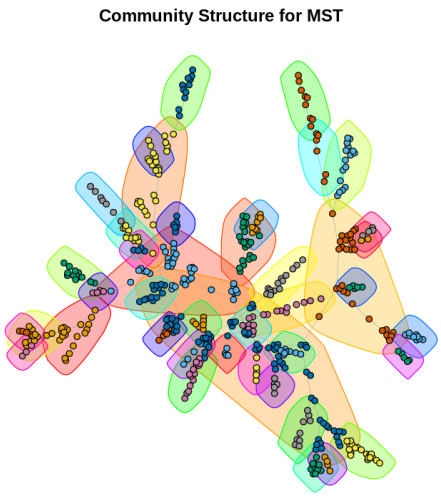



(c) Monthly

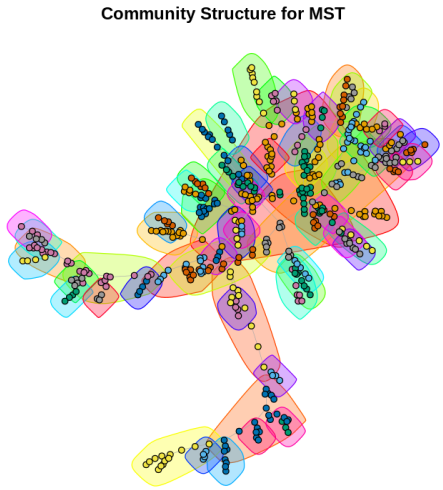
Figure 2: MST of the correlation graph




(a) Daily




(b) Weekly




(c) Monthly

Figure 3: Community Structure for MST

## Question 6

We only keep Mondays in the data set and create the graph `g_w`. We find that there are in total of 143 trading Mondays in 765 days. We ignore 13 companies with invalid data. The resulting plot and metrics are given above.

## Question 7

We only keep 15th day each month in the data set and create the graph `g_m`. We find that there are in total of 25 trading 15th day in 765 days. We ignore 13 companies with invalid data. The resulting plot and metrics are given above.

## Question 8

We can see a clear trend in the results above as the time interval between two sample points increase:

- From Figure 1, we can see that the edge weights distribution becomes more spread out, meaning a more complicated and diverse relationship between a pair of stock prices.
- From Figure 2, a wider range of weights results in a more mixing vine structure, the vertices are not forming a clearly separable regions.
- From Figure 3, there are more clusters in the community structure, each with a smaller amount of vertices. This can be proved by Table 1, we can see that the homogeneity and completeness score both decrease. This indicates that clustering is worse with higher time intervals.
- From Table 1 we can also see that, $\alpha 1$ drops dramatically while $\alpha 2$ remains unchanged. As we explained in Question 5, $\alpha 2$ is only a global estimation of sector clustering, it is not influenced by local graph structure. The decrease of $\alpha 1$ shows the deterioration of the community structure. This means that the stocks from the same sector lose their correlation when the time-scale increases, making it harder to assign a sector to an unknown stock.

From the discussion above we can reach the conclusion that daily data gives the best prediction results.

## Question 9

We read the edges of the graph from the csv file and the vertices of the graph from the json file. We construct the graph `g` with the data set and extract its giant connected component (GCC). We then simplify the graph by merging duplicate edges. Our later questions are based on this simplified graph `gcc`.

There are 2649 vertices and 1003858 edges in `gcc`. The vertices have 3 attributes: name (each polygon's original id minus 1), display (each polygon's display name), centloc (each polygon's central location, this is calculated by taking a mean of all the corners' coordination. The edges also have 3 attributes: time (mean travel time), distance (geographical distance between two central locations), mark (used for plotting).

## Question 10

We calculate the MST from `gcc`, choosing the time attribute as its weight. The MST is shown in Figure 4

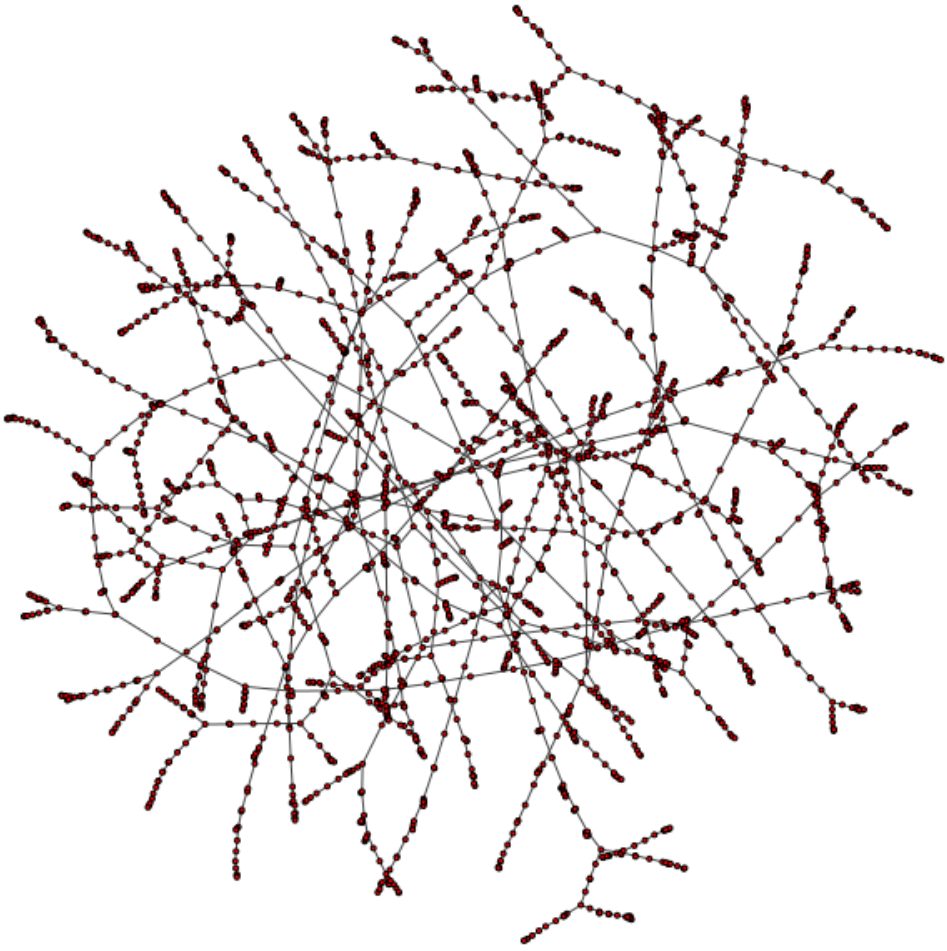


Figure 4: MST for the Uber graph

We choose the first 5 edges in the MST to check our result in Table 2. The unit of distance and time is mile and second, respectively. We can see that the two endpoints of an edge in MST is geographically close to each other. This makes intuitive sense because the MST is supposed to connect all the vertices (locations) such that the lowest cumulative weight of edges (mean travel time) is achieved. Therefore, Prim's algorithm will choose the two endpoints of an edges so that they have a small pairwise travel time and distance.

Table 2: Information for 5 edges in MST

| Source | Target | Distance | Time |
|---|---|---|---|
| [-118.1205,34.1030] | [-118.1313,34.0962] | 0.8846 | 129.765 |
| [-118.1205,34.1030] | [-118.1165,34.0958] | 0.5698 | 118.335 |
| [-118.1378,34.0964] | [-118.1313,34.0962] | 0.4465 | 90.235 |
| [-118.1378,34.0964] | [-118.1322,34.1034] | 0.6210 | 126.475 |
| [-118.1378,34.0964] | [-118.1418,34.0853] | 0.8116 | 125.675 |

## Question 11

The triangle inequality states that the sum of any two sides of a triangle must always be greater than the third side. In our graph, it means that the mean travel time between two addresses will always be smaller than the mean travel time between those addresses via a third address. After randomly sampling 1000 triangles, we found out that 92.2% of the triangles satisfy the triangle inequality.

## Question 12

The travelling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" TSP is a NP-hard problem, we use the 2-approximation algorithm given as follows:

1. Find the MST by Prim's algorithm. The cost of MST is $c(T)$
2. Randomly choose a vertex as root, then pre-order traverse the MST. This traverse goes though each edges exactly twice, its cost is thus $c(C) = 2c(T)$
3. Create a path by connecting deduplicated vertices from the traverse sequence. If a pair of vertices are not connected, find the shortest path between the two vertices and append to the whole path.
4. Calculate the approximate TSP cost as the cost of the path derived above as $c(C^{'})$. We can tell from the deduplication that $c(C^{'}) \leqslant c(C)$

Now we can prove that this algorithm is a 2-approximation rather than 1: If we remove an edge from the optimal path of TSP, we will get a spanning tree. Its cost is guaranteed to be larger than the MST: $c(C^*) \geqslant c(T)$. Therefore from the above discussion, $c(C^{'}) \leqslant 2c(C^*)$.

Therefore we can get the following inequality:

$$1 \leq \frac{c(C^{'})}{c(C^{*})} = \rho \leqslant \frac{c(C^{'})}{c(T)} \leq 2 \tag{1}$$

We iterate 30 times to find different roots for the pre-order traversal and then find the minimum approximate cost $c(C^{'}) = 455450$ seconds. As $c(T) = 269084$, $\rho \leqslant c(C^{'})/c(T) = 1.692$.

## Question 13

We plot the trajectory Santa needs to take in Figure 5.

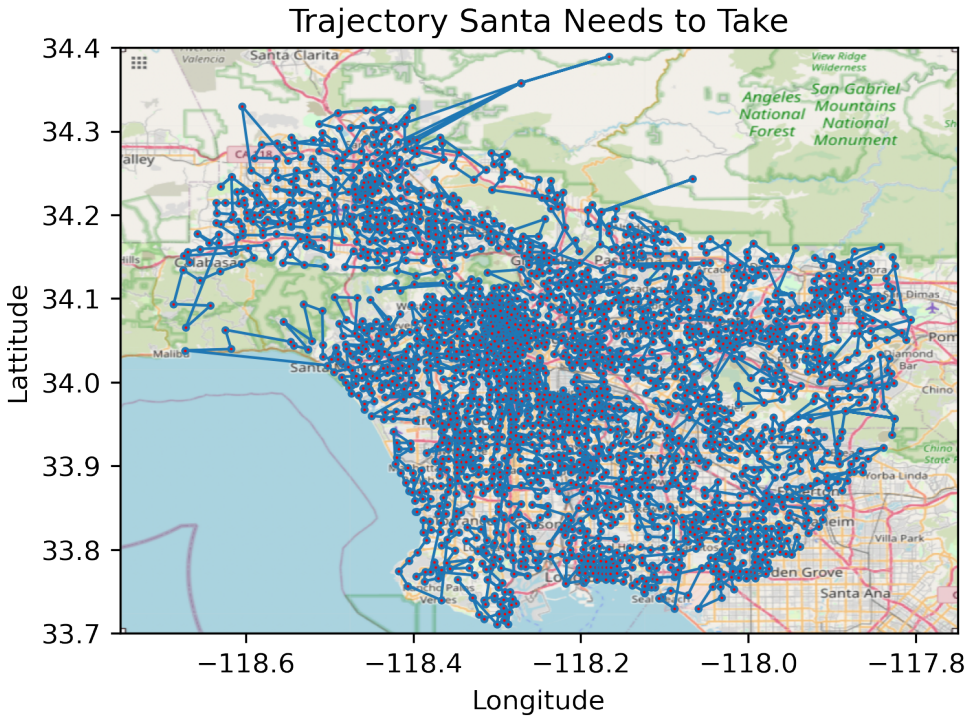



Figure 5: Trajectory Santa needs to take

## Question 14

Using the `Delaunay` function from `scipy.spatial`, we take a subgraph `g_d` from `gcc` by keeping only the edges that are in the Delaunay triangles. We plot the road mesh in Figure 6

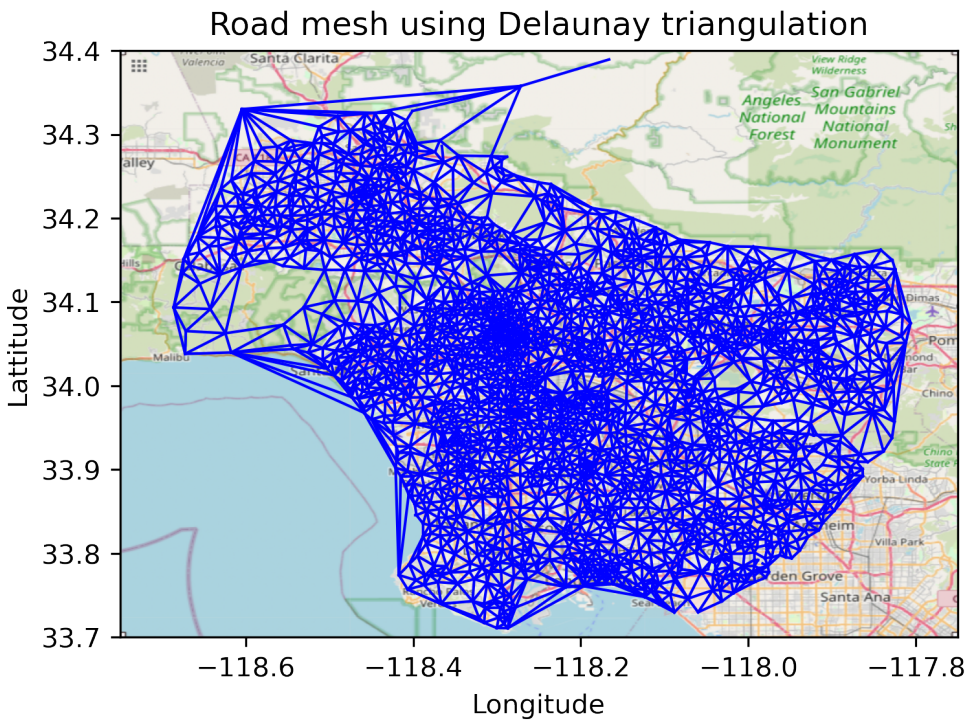



Figure 6: Road mesh using Delaunay triangulation

## Question 15

To calculate the traffic flow for each road, we take the following steps:

1. Calculate the distance attribute for each edge in the graph. This was already done when we construct the original graph `gcc`.

$$distance = 69 \times \sqrt{(\text{Latitude}_{\text{point1}} - \text{Latitude}_{\text{point2}})^2 + (\text{Longitude}_{\text{point1}} - \text{Longitude}_{\text{point2}})^2} \tag{2}$$

2. Calculate the velocity for each edge: $v = distance/time \times 3600$ miles/hour

3. Calculate the time each car takes to travel through a point:

$$\Delta t = \frac{0.003}{v} + \left(\frac{2}{3600}\right) \text{ hours/car} \tag{3}$$

   Where the car length is 0.003 miles and the safety gap is 2 seconds.

4. Calculate the traffic flow for two lanes in each direction:

$$\text{flow} = 4 \times \frac{1}{\Delta t} \text{ cars/hour} \tag{4}$$

Assuming no traffic jam, consider the calculated traffic flow as the max capacity of each road, we create a new capacity attribute for the edges in `g_d` with our results.

## Question 16

We use [-118.62, 34.02] as Malibu's coordinates instead. We mark the two vertices closest to the coordinates on the road map in Figure 7. We also plot the local road map near the two vertices in Figure 8

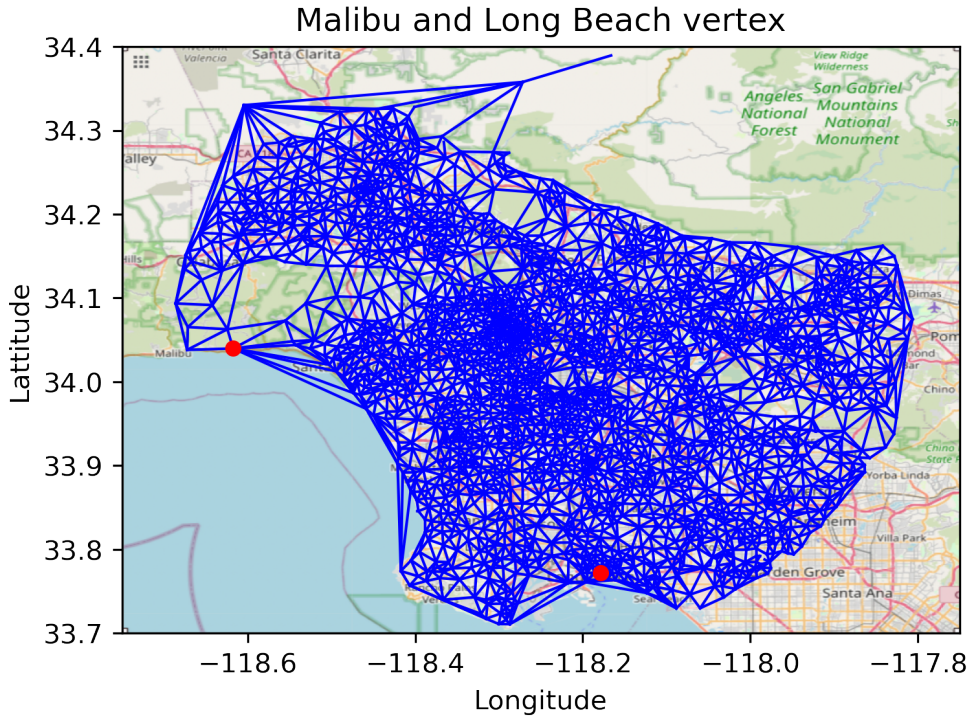
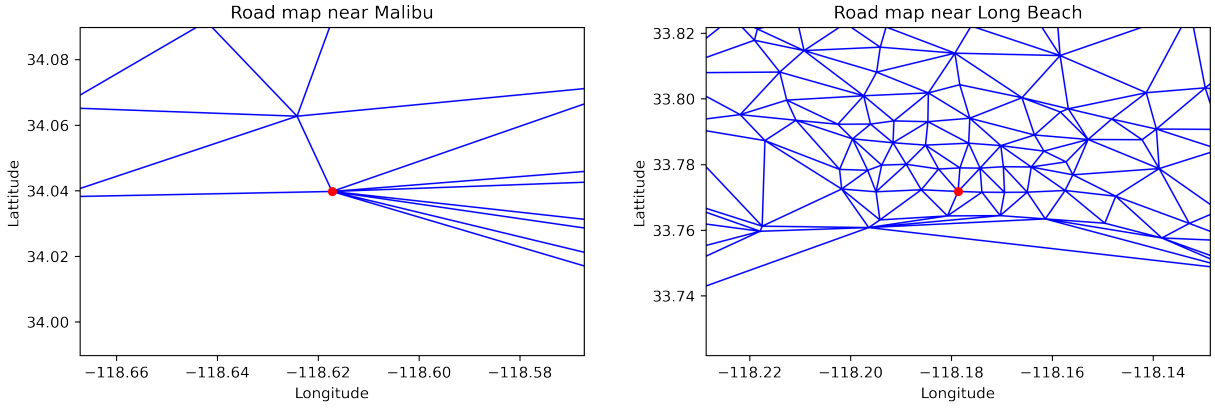



Figure 7: Malibu and Long Beach vertex




(a) Malibu

(b) Long Beach

Figure 8: Local road map near the two vertices

Using the `maxflow_value` function from `iGraph`, we get the max flow of 30642 (assuming four lanes per road). From 8 we know that the degree of Malibu and Long Beach vertices are 9 and 6. Not surprisingly, the number of edge-disjoint paths between them is 6.

## Question 17

We apply a threshold on the time attribute of all the edges. The red lines in Figure 9 have a mean travel time larger than 800 seconds. They are removed from `g_d` to create a subgraph `g_trim`. We can see that by applying this threshold we effectively removed the fake roads in the ocean and the hills.

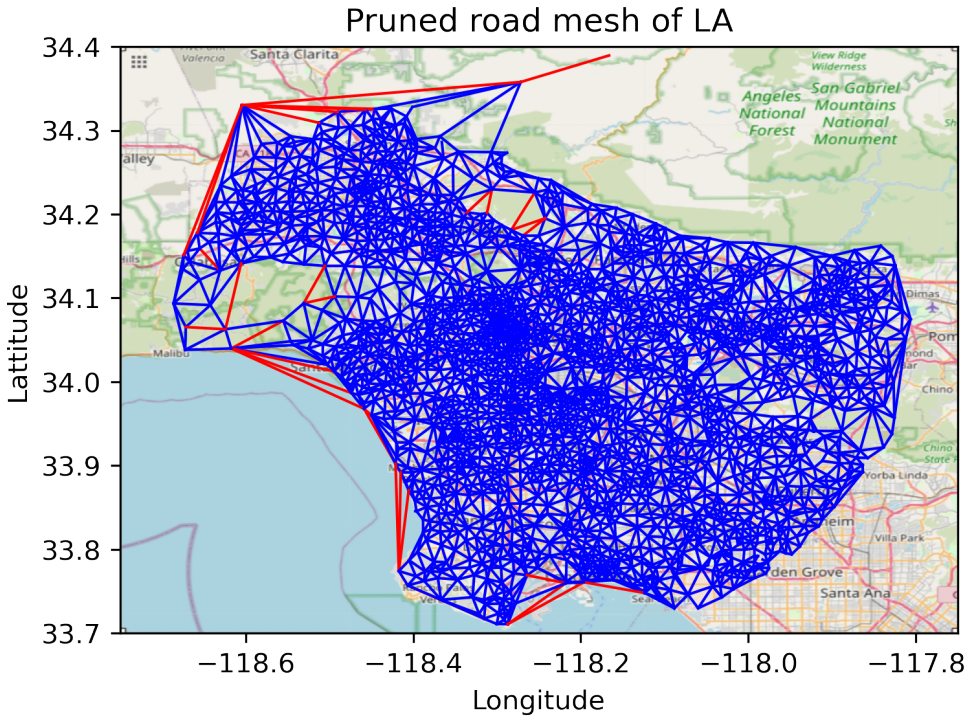
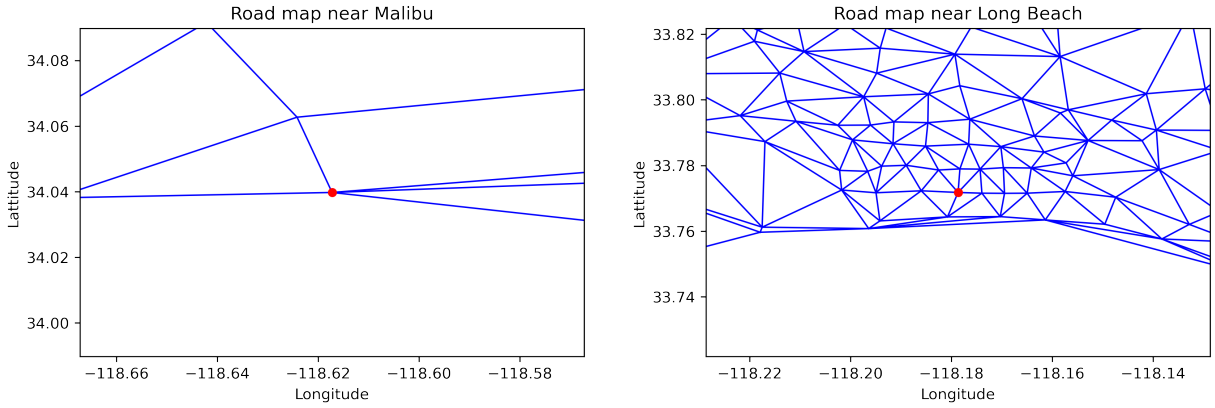



Figure 9: Pruned road mesh of LA

## Question 18

We plot the the local road map near the two vertices again in Figure 10.

After the trimming, the degree of Malibu and Long Beach vertices are now 5 and 6. The number of edge-disjoint paths between them is therefore 5. However, the max flow between them is still 30642. This is because there still exists many paths that cars can take to reach Long Beach from Malibu after the trimming. In other words, the capacity of all the roads leading from Malibu to Long Beach is still larger than 30642 cars, leaving the max flow value unchanged.




(a) Malibu

(b) Long Beach

Figure 10: Local road map near the two vertices

# Question 19

### Strategy 1

We know that:

$$\text{extra distance(v,s)} = \text{distance of shortest path(v,s)} - \text{euclidean distance(v,s)}$$

Our steps to pick the 20 edges are therefore as follows:

1. Calculate the $2647 \times 2647$ distance matrix $D$, where $D_{vs}$ is the geographical euclidean distance between vertex v and s. The distribution of the distances are shown in the histogram in Figure 11.
2. Use the `shortest_path` method from `iGraph` to get the shortest distance between any two vertices as the $2647 \times 2647$ matrix $S$. The time complexity of this step is $O(|V| \times |E|)$
3. Calculate the extra distance matrix and keep its upper triangle.
4. Get the top 20 extra distances and add new edges with the respective distances to the graph. We use `np.argsort` to sort the results, the time complexity is $O(V^2 \times \log(V^2))$. Note that using `np.argmax` 20 times might lower the time complexity theoretically but not empirically.

The edges are shown in Table 3 and Figure 12. Based on the discussion above, the time complexity of the algorithm we use is $O(|V| \times |E|)$.

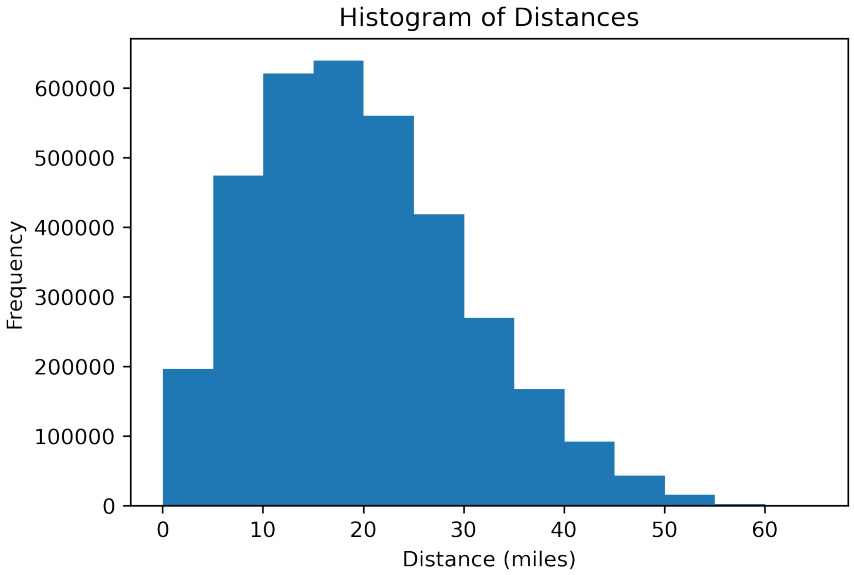



Figure 11: Histogram of pairwise distances

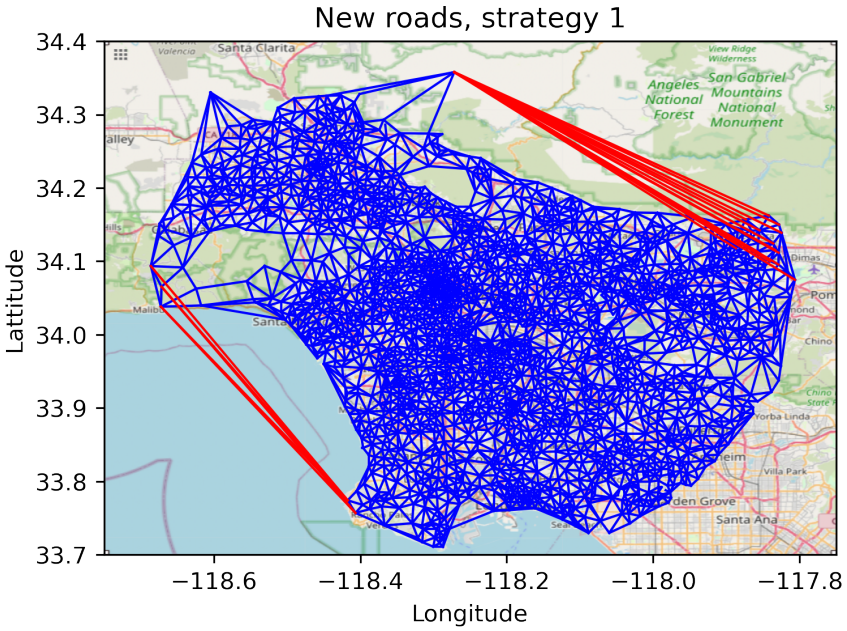



Figure 12: New roads, strategy 1

Table 3: 20 new edges for strategy 1

| Source | Target |
|---|---|
| -117.85748312987019,34.112945454545475 | -118.27158981797648,34.35782482248528 |
| -117.80629424922105,34.074753919003115 | -118.27158981797648,34.35782482248528 |
| -117.84598029213484,34.103187044943816 | -118.27158981797648,34.35782482248528 |
| -117.84047893749997,34.100165062500004 | -118.27158981797648,34.35782482248528 |
| -118.41714115454548,33.774103109090916 | -118.68599468727271,34.09347549818183 |
| -117.88700062790699,34.11772844186047 | -118.27158981797648,34.35782482248528 |
| -117.82879819999992,34.0955664090909 | -118.27158981797648,34.35782482248528 |
| -117.83602374666668,34.124341866666676 | -118.27158981797648,34.35782482248528 |
| -117.85454409302326,34.142120790697675 | -118.27158981797648,34.35782482248528 |
| -117.88665511320752,34.12443260377358 | -118.27158981797648,34.35782482248528 |
| -118.67225393333337,34.03809095384618 | -118.40648371022725,33.75689578693178 |
| -117.82467288479266,34.13885138248847 | -118.27158981797648,34.35782482248528 |
| -117.82866074137928,34.111707206896554 | -118.27158981797648,34.35782482248528 |
| -117.83331135955059,34.11532583146067 | -118.27158981797648,34.35782482248528 |
| -118.67225393333337,34.03809095384618 | -118.41714115454548,33.774103109090916 |
| -117.8732378168498,34.15698468864466 | -118.27158981797648,34.35782482248528 |
| -117.82810402090584,34.1499674041812 | -118.27158981797648,34.35782482248528 |
| -118.40648371022725,33.75689578693178 | -118.67145511923503,34.06533878290219 |
| -117.84197423684212,34.16224818421053 | -118.27158981797648,34.35782482248528 |
| -118.41714115454548,33.774103109090916 | -118.67145511923503,34.06533878290219 |

# Question 20

### Strategy 2

We know that:

$$\text{weighted extra distance(v,s)} = \text{extra distance(v,s)} * \text{frequency(v,s)}$$

So we simply generate a $2647 \times 2647$ frequency matrix $F$ where $F_{vs} \in [1, 1000]$ and then element-wise multiply with the extra distance matrix that we generated in Question 19. The edges are shown in Table 4 and Figure 13. Based on the discussion above, the time complexity of the algorithm is still $O(|V| \times |E|)$.

Table 4: 20 new edges for strategy 2

| Source | Target |
|---|---|
| -118.27158981797648,34.35782482248528 | -117.83861802211293,33.92170087223584 |
| -117.9770713023256,34.096261674418606 | -118.27158981797648,34.35782482248528 |
| -117.98386745205488,34.14503512328768 | -118.27158981797648,34.35782482248528 |
| -117.90265488571433,34.11780265714285 | -118.27158981797648,34.35782482248528 |
| -118.27158981797648,34.35782482248528 | -117.90945247933884,33.93498581818182 |
| -118.65489981553401,34.15918811650484 | -118.39683944230771,33.7679271025641 |
| -117.8802836388889,34.14393644444445 | -118.27158981797648,34.35782482248528 |
| -117.85366782967031,34.028488450549446 | -118.27158981797648,34.35782482248528 |
| -118.27158981797648,34.35782482248528 | -117.89388585074627,33.908821776119396 |
| -117.91658202173915,34.05489234782609 | -118.27158981797648,34.35782482248528 |
| -117.85623433333333,34.044350606060604 | -118.27158981797648,34.35782482248528 |
| -117.89826721875,34.05742703125 | -118.27158981797648,34.35782482248528 |
| -117.99399975280905,34.16600654494379 | -118.27158981797648,34.35782482248528 |
| -118.40648371022725,33.75689578693178 | -118.68599468727271,34.09347549818183 |
| -117.82657713380283,33.99942785211269 | -118.27158981797648,34.35782482248528 |
| -117.85758971428571,34.08924028571428 | -118.27158981797648,34.35782482248528 |
| -118.39683944230771,33.7679271025641 | -118.67145511923503,34.06533878290219 |
| -117.82866074137928,34.111707206896554 | -118.27158981797648,34.35782482248528 |
| -117.89532858333337,34.12746769444445 | -118.27158981797648,34.35782482248528 |
| -118.41714115454548,33.774103109090916 | -118.67145511923503,34.06533878290219 |


Figure 13: New roads, strategy 2


Figure 14: New roads, strategy 3

# Question 21

### Strategy 3

For the dynamic strategy, we use the same algorithm in Question 19 20 times. The only difference is that we use `np.argmax` this time to find the maximum extra distance every iteration. The edges are shown in Table 5 and Figure 14. Based on the discussion above, though we have to multiply the time complexity by a constant coefficient 20, the time complexity of the algorithm is still $O(|V| \times |E|)$.

Table 5: 20 new edges for strategy 3

| Source | Target |
|---|---|
| -118.41714115454548,33.774103109090916 | -118.67145511923503,34.06533878290219 |
| -117.84197423684212,34.16224818421053 | -118.27158981797648,34.35782482248528 |
| -118.57609893103452,34.163914112068966 | -118.55510068503939,34.07289098425196 |
| -117.90384881034481,34.12744793103448 | -118.27158981797648,34.35782482248528 |
| -117.94347146212125,34.160677848484866 | -118.27158981797648,34.35782482248528 |
| -117.99399975280905,34.16600654494379 | -118.27158981797648,34.35782482248528 |
| -118.41714115454548,33.774103109090916 | -118.62418590723568,34.06269522170691 |
| -118.046042942029,34.17109230434782 | -118.27158981797648,34.35782482248528 |
| -118.10868277868856,34.18016231147541 | -118.27158981797648,34.35782482248528 |
| -118.5556897357724,34.143095296747944 | -118.53067983104125,34.09368131827117 |
| -118.66929280689664,34.156163006896556 | -117.98288185714287,33.787883428571426 |
| -118.3036932608696,34.1928858115942 | -118.2728725826087,34.24626368695651 |
| -118.15749682068964,34.20758387586207 | -118.27158981797648,34.35782482248528 |
| -118.67225393333337,34.03809095384618 | -118.39491433093522,33.796855824940074 |
| -118.63048660074622,34.1327155932836 | -117.98288185714287,33.787883428571426 |
| -118.39748945762716,33.84561491864407 | -118.67145511923503,34.06533878290219 |
| -118.63048660074622,34.1327155932836 | -118.36653025668451,33.80541077540108 |
| -118.61724170318024,34.03975774911659 | -118.40485093013112,33.78694638427947 |
| -118.28301819999999,34.18119773333334 | -118.3059771739131,34.22977666086955 |
| -118.66929280689664,34.156163006896556 | -118.02266135185184,33.78354657407407 |

# Question 22

**Strategy 4**

We know that:

travel speed(v,s) = distance of shortest path(v,s) / travel time of shortest path(v,s)
extra time(v,s) = travel time of shortest path(v,s) - euclidean distance(v,s)/travel speed(v,s)

All the entries above are $2647 \times 2647$ matrices. We calculate the travel speed first and then the extra time. We then swap the extra time with the extra distance in Question 19. The new edges are shown in Table 6 and Figure 15. Based on the discussion above, though we have to multiply the time complexity by a constant coefficient 2, the time complexity of the algorithm is still $O(|V| \times |E|)$.

Table 6: 20 new edges for strategy 4

| Source | Target |
|---|---|
| -118.61281342424239,34.15615184848486 | -118.53067983104125,34.09368131827117 |
| -118.58392439285714,34.19973064285714 | -118.55510068503939,34.07289098425196 |
| -118.65489981553401,34.15918811650484 | -118.40648371022725,33.75689578693178 |
| -118.3868687012987,33.76222715584416 | -118.67145511923503,34.06533878290219 |
| -118.59018528030296,34.14922015909091 | -118.53067983104125,34.09368131827117 |
| -118.59007249999999,34.199754199999994 | -118.55510068503939,34.07289098425196 |
| -118.65489981553401,34.15918811650484 | -118.41714115454548,33.774103109090916 |
| -118.63048660074622,34.1327155932836 | -118.40648371022725,33.75689578693178 |
| -118.63048660074622,34.1327155932836 | -118.41714115454548,33.774103109090916 |
| -118.59018528030296,34.14922015909091 | -118.55510068503939,34.07289098425196 |
| -118.40485093013112,33.78694638427947 | -118.67145511923503,34.06533878290219 |
| -118.57017334070797,34.14872431415929 | -118.55510068503939,34.07289098425196 |
| -118.6174550722892,34.15459418072292 | -118.53067983104125,34.09368131827117 |
| -118.67225393333337,34.03809095384618 | -118.40648371022725,33.75689578693178 |
| -118.67225393333337,34.03809095384618 | -118.41714115454548,33.774103109090916 |
| -118.40648371022725,33.75689578693178 | -118.68599468727271,34.09347549818183 |
| -118.39683944230771,33.7679271025641 | -118.67145511923503,34.06533878290219 |
| -118.41714115454548,33.774103109090916 | -118.68599468727271,34.09347549818183 |
| -118.40648371022725,33.75689578693178 | -118.67145511923503,34.06533878290219 |
| -118.41714115454548,33.774103109090916 | -118.67145511923503,34.06533878290219 |


Figure 15: New roads, strategy 4


Figure 16: New roads, strategy 5

# Question 23

### Strategy 5

Similar to Question 21, for the dynamic strategy, we iterate the algorithm in Question 22 20 times. The new edges are shown in Table 7 and Figure 16. Based on the discussion above, though we have to multiply the time complexity by a constant coefficient 40, the time complexity of the algorithm is still $O(|V| \times |E|)$.

Table 7: 20 new edges for strategy 5

| Source | Target |
|---|---|
| -118.41714115454548,33.774103109090916 | -118.67145511923503,34.06533878290219 |
| -118.6174550722892,34.15459418072292 | -118.53067983104125,34.09368131827117 |
| -118.57017334070797,34.14872431415929 | -118.55510068503939,34.07289098425196 |
| -117.84197423684212,34.16224818421053 | -118.27158981797648,34.35782482248528 |
| -118.41714115454548,33.774103109090916 | -118.62418590723568,34.06269522170691 |
| -117.89365872839508,34.05103139506174 | -118.27158981797648,34.35782482248528 |
| -117.90384881034481,34.12744793103448 | -118.27158981797648,34.35782482248528 |
| -117.94347146212125,34.160677848484866 | -118.27158981797648,34.35782482248528 |
| -117.95946314285713,33.94862083333332 | -118.27158981797648,34.35782482248528 |
| -118.01154852195125,33.99293787317071 | -118.27158981797648,34.35782482248528 |
| -117.99399975280905,34.16600654494379 | -118.27158981797648,34.35782482248528 |
| -118.39748945762716,33.84561491864407 | -118.67145511923503,34.06533878290219 |
| -118.53047408659789,34.14032269896909 | -118.53067983104125,34.09368131827117 |
| -118.04268341379307,34.0580242413793 | -118.27158981797648,34.35782482248528 |
| -118.046042942029,34.17109230434782 | -118.27158981797648,34.35782482248528 |
| -118.61724170318024,34.03975774911659 | -118.40485093013112,33.78694638427947 |
| -118.31185060431655,34.20150541726619 | -118.29105503125,34.250763375 |
| -118.10868277868856,34.18016231147541 | -118.27158981797648,34.35782482248528 |
| -118.3036932608696,34.1928858115942 | -118.3059771739131,34.22977666086955 |
| -118.63048660074622,34.1327155932836 | -118.38821546987946,33.82029325301204 |

# Question 24

### Strategy 6

The new strategy we come up with is related to the capacity attribute that we later add to the project. From our calculation of the traffic flow we can tell that the faster the speed on a road, the larger its capacity. In our previous strategies, there are mainly two issues:

- We did not consider traffic jam. In this strategy we assume that the larger the capacity of a road, the easier it is to be jammed (which is consistent from our real life experience).
- The road we suggested are too long to be built practically. This can be solved by given a boundary to the distance attribute so that we can only add more practical edges. We did not implement this here because it takes too long to run.

Here we first find the shortest path between any two vertices by distance, and then sum the capacity of the edges in the path. We take the top 20 paths that yield the maximum sum, which are potentially easier to be jammed. The new edges are shown in Figure 17.

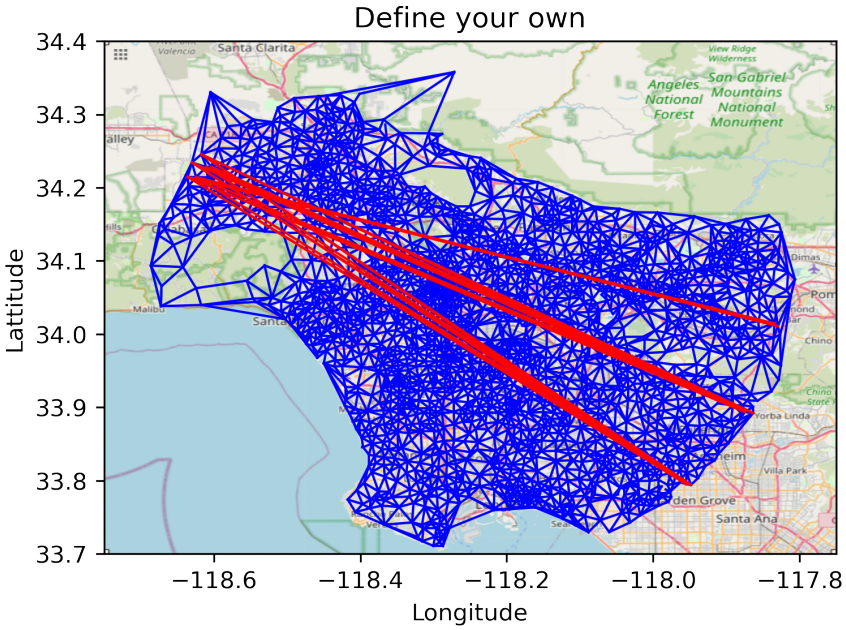



Figure 17: New roads, strategy 6

# Question 25

**Define your own task**

Task Statement:

Recall the name of our Uber project: 'Let's help Santa', it's essentially a TSP problem that we have looked into in Question 12. However, since TSP is a NP-Hard problem, we cannot find a polynomial-time solution. Instead, we use an approximation algorithm to approach the optimal result. For this task, we will look into another solution of the TSP problem by taking the following steps:

- First we find the MST of the road network we build in Question 14, we then find its community structure by walktrap through the distance attribute then we calculated from the coordinates.
- We compare the community structure with the one we get from walktrap through the time attribute that we get from Uber data.
- We then contract each cluster into a single vertex, we also combine the edges at the same time.
- Finally we find the optimal TSP solution in the contracted graph. We also find the major LA traffic lines at the same time.

Result:

First, we find the MST of the road network `g_trim`, then we use `community_walktrap` to extract its community structure. We plot the vertices in different clusters in Figure 18.

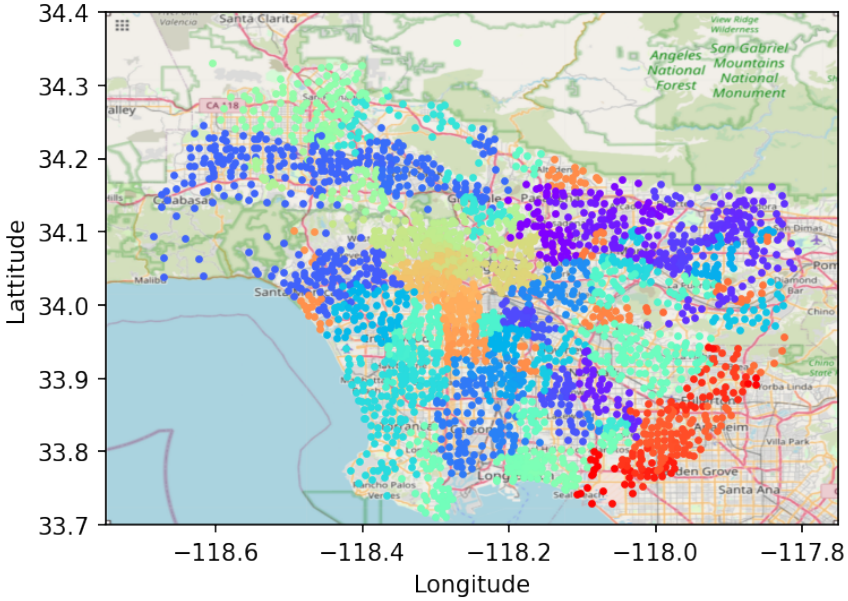



Figure 18: Location community structure from distance

We get a total of 164 clusters, comparing the clusters with the real LA community structure in Figure 19, we can see that the clusters partially align with the real world situation.

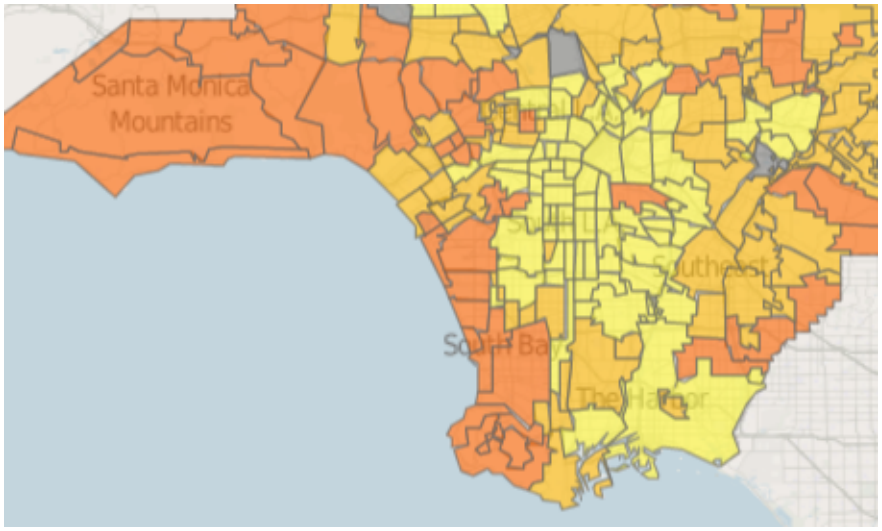



Figure 19: Real LA community structure

In order to analysis the connections between the clusters, we classify all the edges to be within-clusters and inter-clusters. In Figure, the grey lines represent inter-clusters connection. We can see a clear boundary that is formed by the edges.

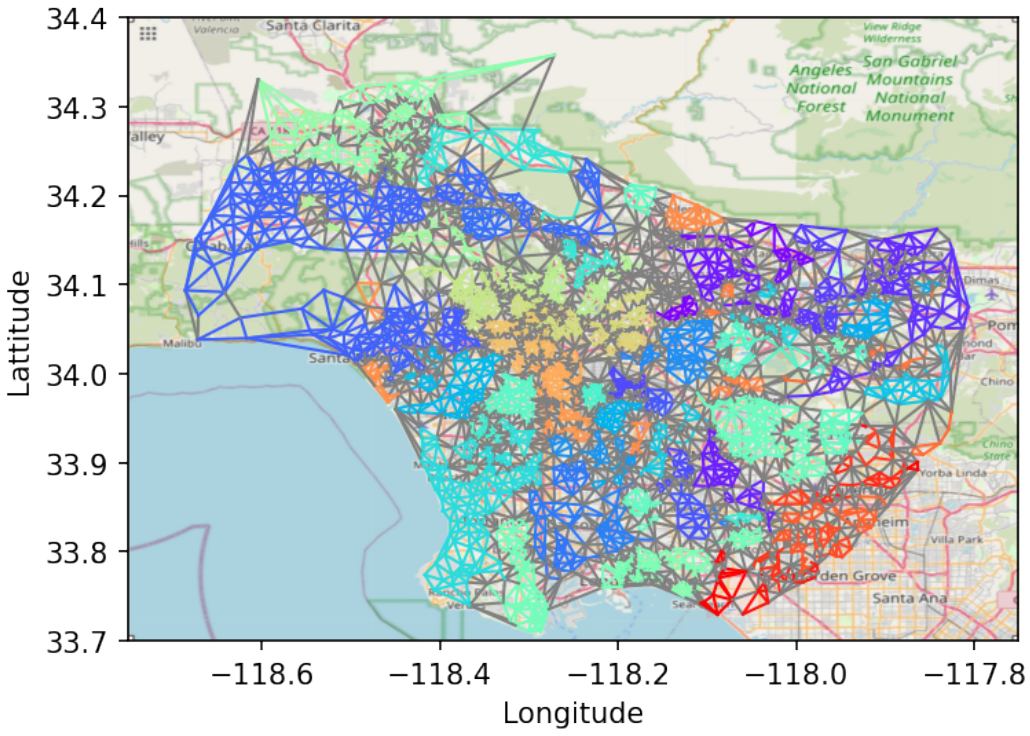



Figure 20: Location community structure from distance

We then try to contract each cluster into one vertex, and simplify the resulting graph at the same time. For the edge attributes, we take a mean value of time and distance, take a sum of the internal traffic flow. The resulting graph is shown in Figure 21.

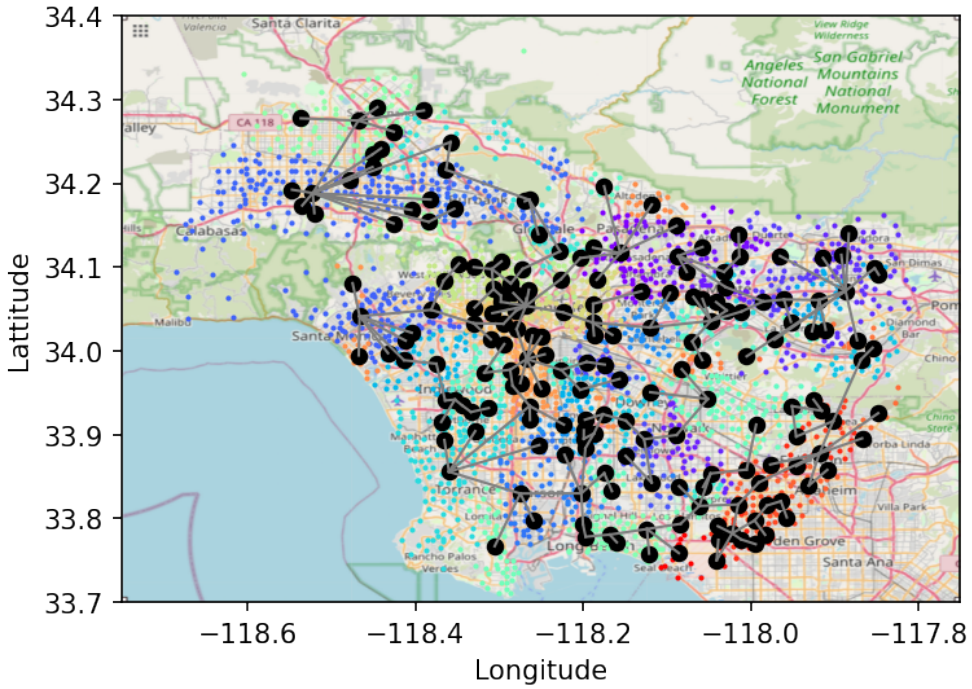



Figure 21: Contracted graph from distance clusters

The black dots are all the representative vertices that we calculated by taking the mean of all the vertices in a cluster. The grey lines are the edges between them. However, we can see that the graph is not well-connected, most edges are concentrated in traffic hubs like downtown LA, Santa Monica, etc.

Now we try to utilize our Uber data: this time we walktrap directly on the graph by its time attribute. The community structure we find are shown in Figure 22.

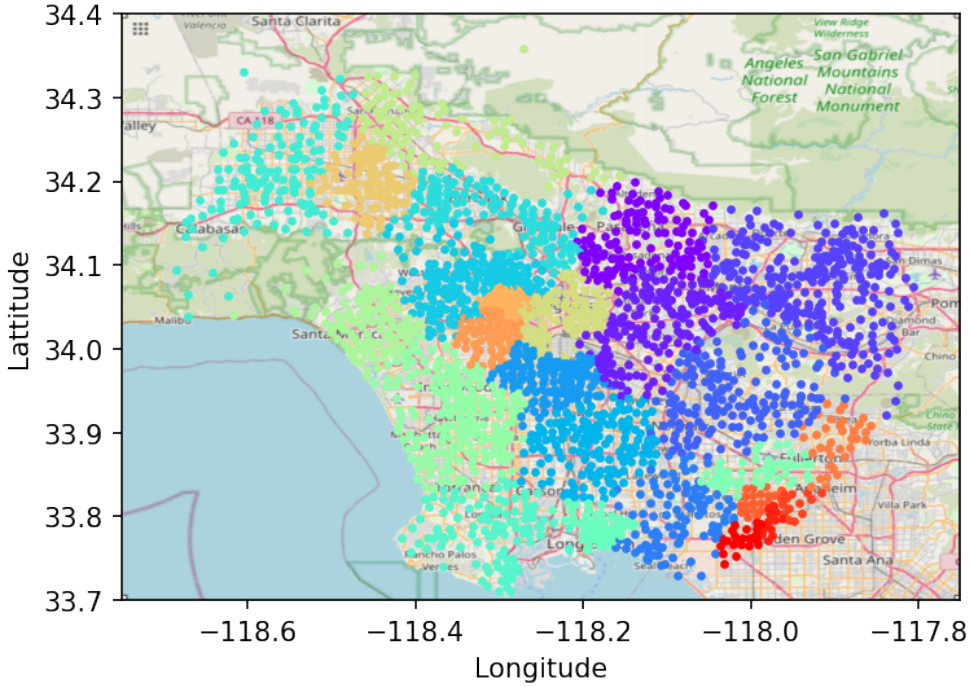



Figure 22: Location community structure from time

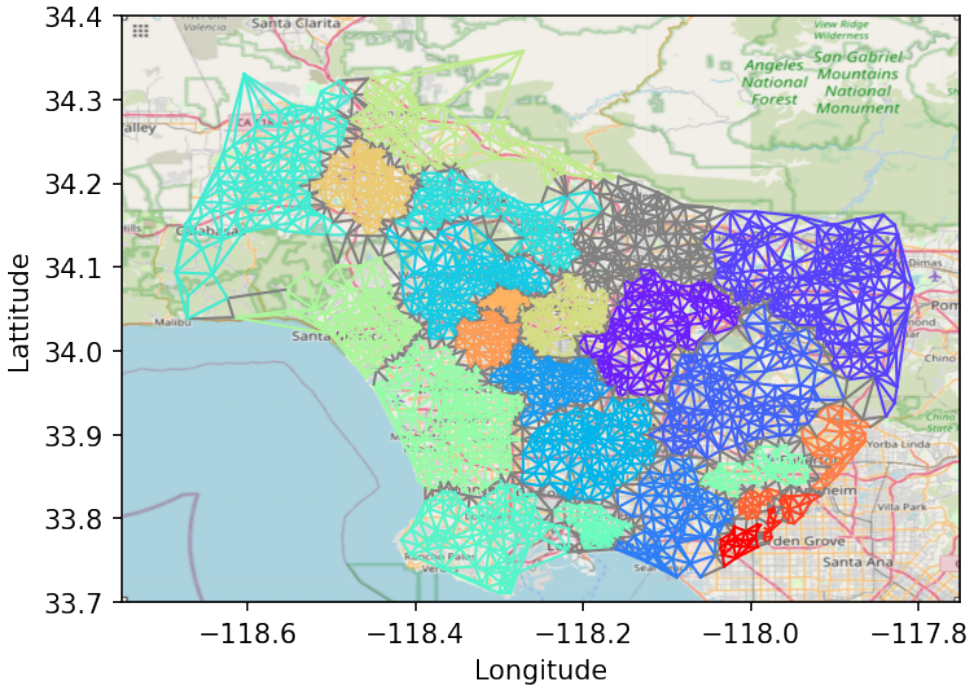



Figure 23: Location community structure from time

We then plot the edges in Figure 23. This time we can see a clearer community structure and border edges. We can see from 19 that the size of Santa Monica follows closely to the real world situation. This means that the travel time data brought by Uber data sets also provides us information on a city's community structure. We use the similar method to contract the graph, the result are shown in Figure 24

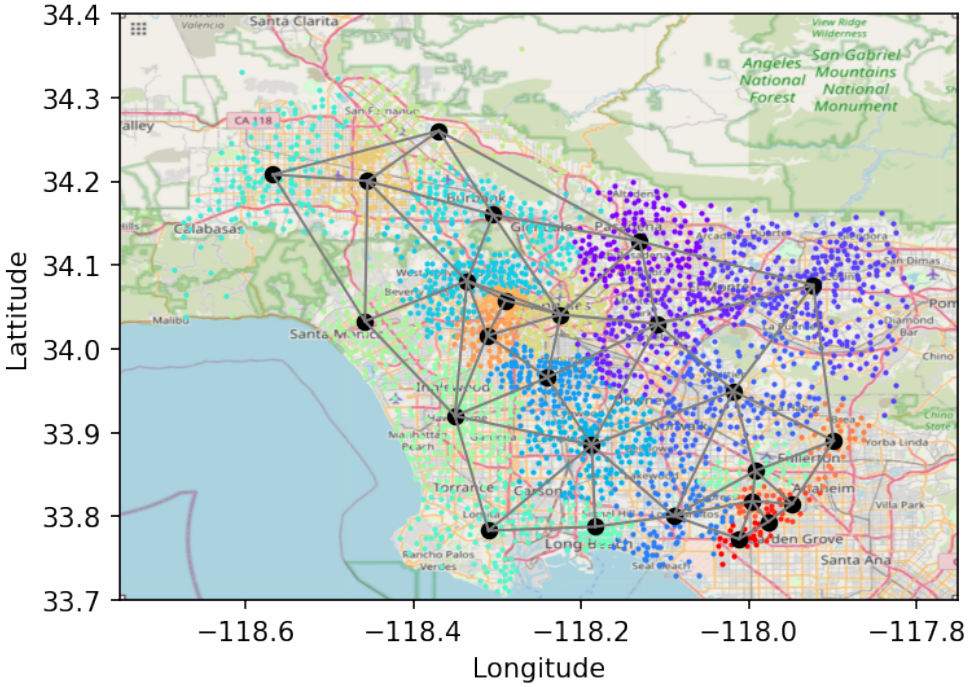



Figure 24: Contracted graph from time clusters

This time the representative vertices are well-connected to each other, there are only 25 clusters in the graph. We can now use a simple enumerating approach to calculate the optimal TSP solution. Moreover, in order to connect the central areas shown in the graph, we detect major traffic lines in LA, they have a large capacity and are thus crucial to the road network. It's also interesting to see how this map relates to the actual LA metro map in Figure 26

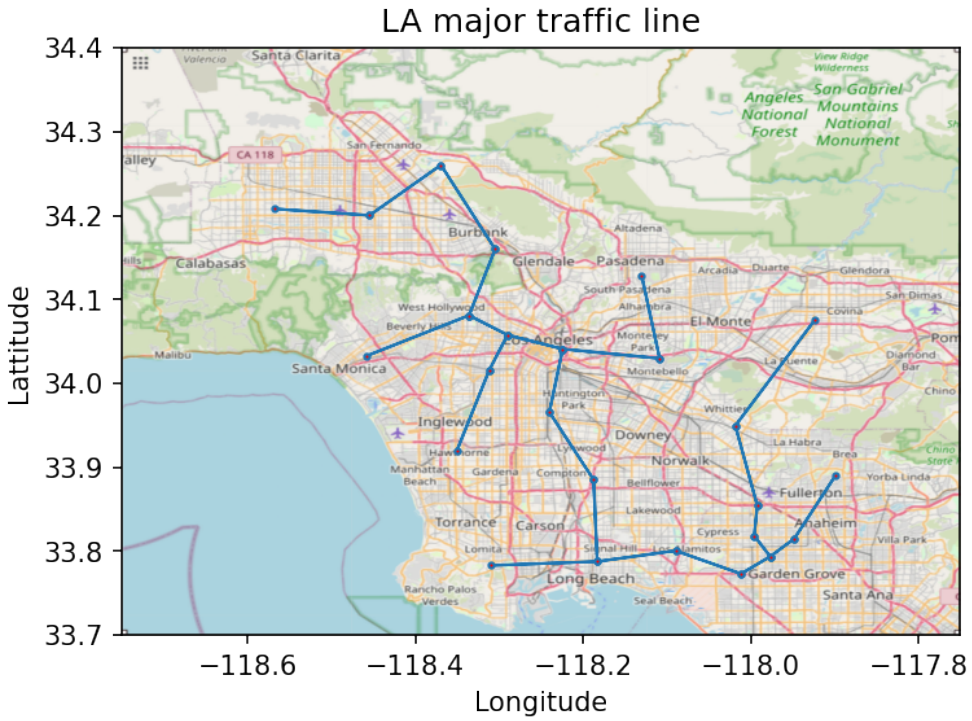



Figure 25: Major traffic lines in LA

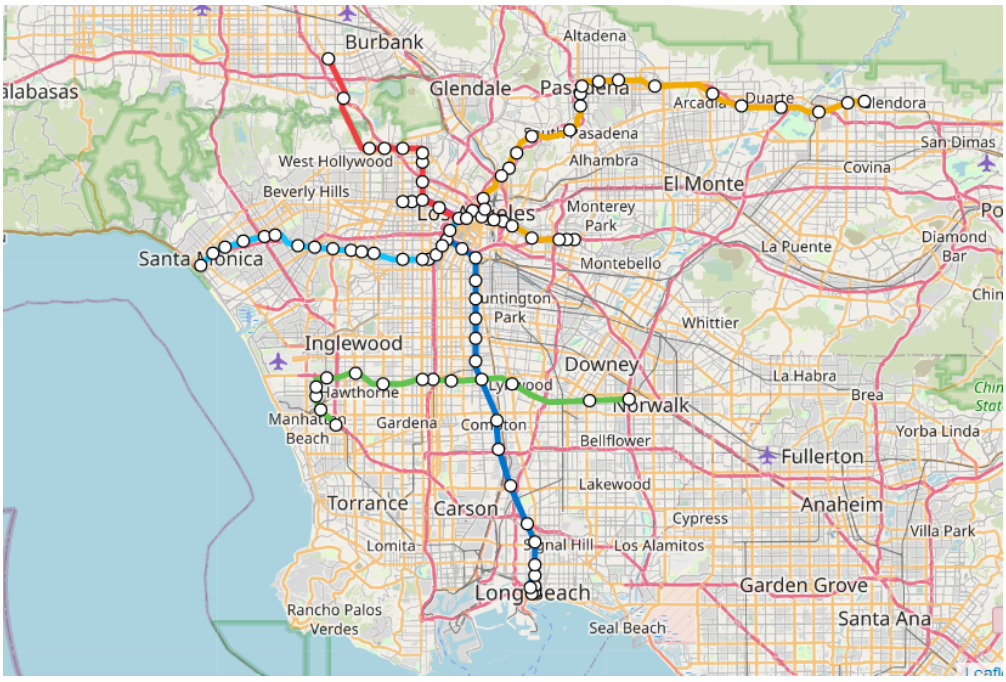



Figure 26: Metro lines in LA