ECE 232 Project 4 Report

July 24, 2024

Name: Kelly Furuya

1 Stock Market

1.1 Return Correlation

Q1: What are the upper and lower bounds on p_{ij}? Provide a justification for using log-normalized return $(r_i(t))$ instead of regular return $(q_i(t))$?.

The upper and lower bounds for p_{ij} are 1 and -1.

One reason to use log-normalized return instead of regular return is that using log allows us to still show the differences between the returns of each stock while constaining the values to a more manageable range.

1.2 Constructing correlation graphs

Q2: Plot a histogram showing the un-normalized distribution of edge weights.

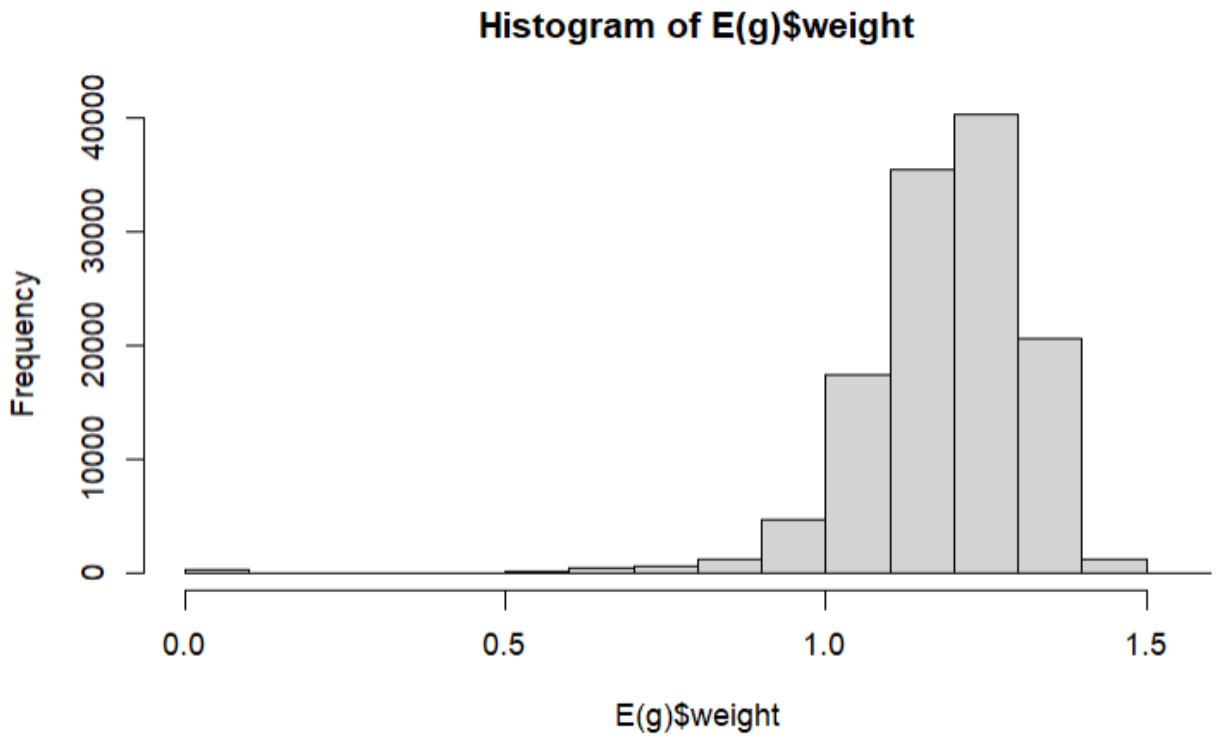


Figure 1: Un-normalized edge-weight distribution for stocks with complete data

We can see from Figure 1 that the edge-weight distribution follows a roughly binomial distribution with the highest number of edges with a weight around 1.25. There are no edges with a weight of 0 or with a negative weight. The correlation function w_{ij} makes use of p_{ij}'s upper and lower bounds to prevent trying to find a square root of a negative. Since p_{ij} is restricted to ± 1, the values of w_{ij} will range from 0 to 2. No edges with weight 0 indicates that there are no two stocks that are completely correlated or isolated from one another.

1.3 Minimum spanning tree (MST)

Q3: Extract the MST of the correlation graph. Plot the MST and color-code the nodes based o sectors. Do you see any pattern in the MST? Provide a detailed explanation about the pattern you observe.

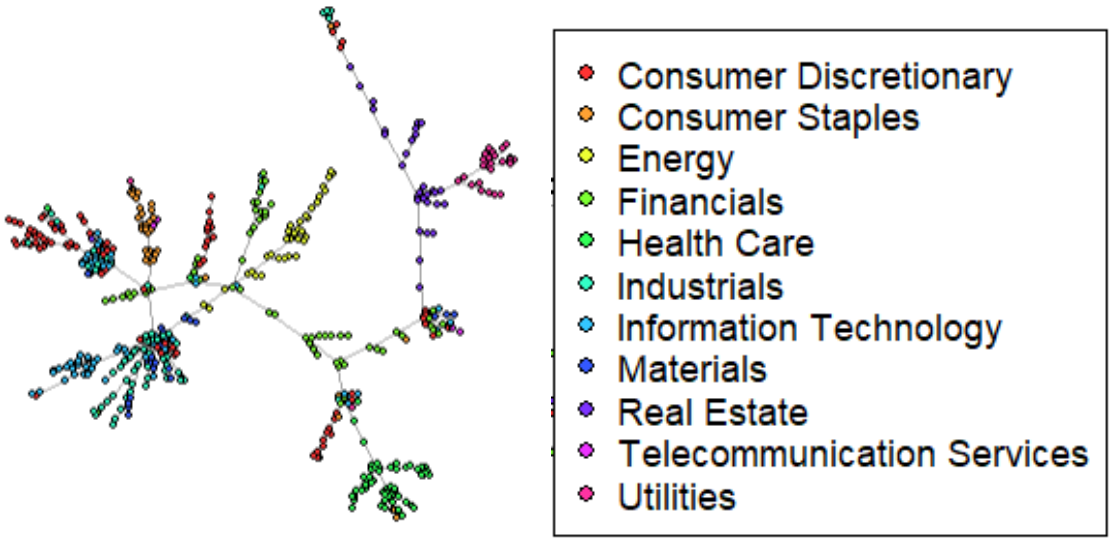


Figure 2: Color-coded MST with legend indicating which sector each color correlates to

We can see that the MST ends up roughly grouping each stock together based on its sector. For example, one of the vines near the center of the figure contains exclusively yellow nodes indicating that those are all energy related stocks. There are a couple of yellow nodes not on this individual "vine" but they are extremely close by. The grouping of stocks by sector make sense as the MST algorithm will try to connect nodes with edges of smaller weights to try to keep the overall edge-weight to a minimum. The stocks that are more closely correlated to each other wiill end up with a smaller weight as p_{ij} will have a value closer to 1 resulting in a lower weight. It also makes sense that there are a few "vines" with two or more sectors grouped together. For example, Industrials and Materials are grouped together as the two industries can have a large impact on one another. If there is an issue with the production or transporation of materials, then the industrial companies will struggle to generate product, resutling in a negative impact in both sectors. The relatively large gaps between each grouping is a nice visual representation of how much less of an effect poorly-correlated stocks/sectors have on each other.

Q4: Run a community detection algorithm on the MST obtained above. Plot the communities formed. Compute the homogeneity and completeness of the clustering.

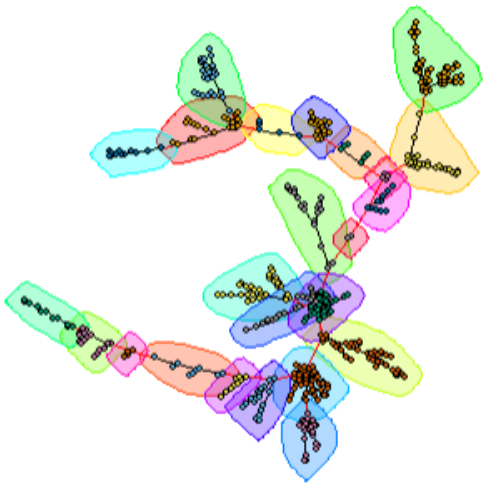


Figure 3: Communities within the MST

Intuitively, the walktrap community detection algorithm groups the nodes roughly based on sector as well. The homogeneity is 0.6808973 and the completeness is 0.5241226. The homogeneity score indicates that each community contains several nodes from different sectors and the homogeneity score indicates that not all nodes belonging to a sector are contained within the same community. This indicates that there are some stocks within given sectors that more closely correlate to stocks within different sectors resulting in branches of a "vine" with a mixture of stocks from different sectors.

1.4 Sector clustering in MSTs

Q5: Report the value of α for the mentioned two cases and provide an interpretation for the difference.

We are given the metric:

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

where S_i is the sector of node i. We are then asked to compare the two cases:

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

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

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.

In the first case, we get a score of 0.8287325. P is similar to the copmleteness score in that i is determining how well grouped a node is with its neighbors based on its sector. A high score for P means that more of its neighbors are in the same sector as node i, indicating a smaller weight

and a higher correlation. The resulting α, which we will call $\alpha 1$, would then also be higher as the summation would result in a higher score if more of the nodes within a stock

In the second case, which we will call $\alpha 2$, we get a score of 0.1141881. For this case, P is a comparison of all the nodes in the graph as opposed to a localized area. This more so focuses on what percentage of stocks fall into a given sector and is not as indicative of how well a random stock correlates to others.

The first method provides better results as it takes advantage of the "vine" structure of the MST to properly identify stocks that are more closely related to one another to help predict how one stock may affect another.

1.5 Correlation graphs for weekly data

Q6. Repeat questions 2, 3, 4, 5 on WEEKLY data

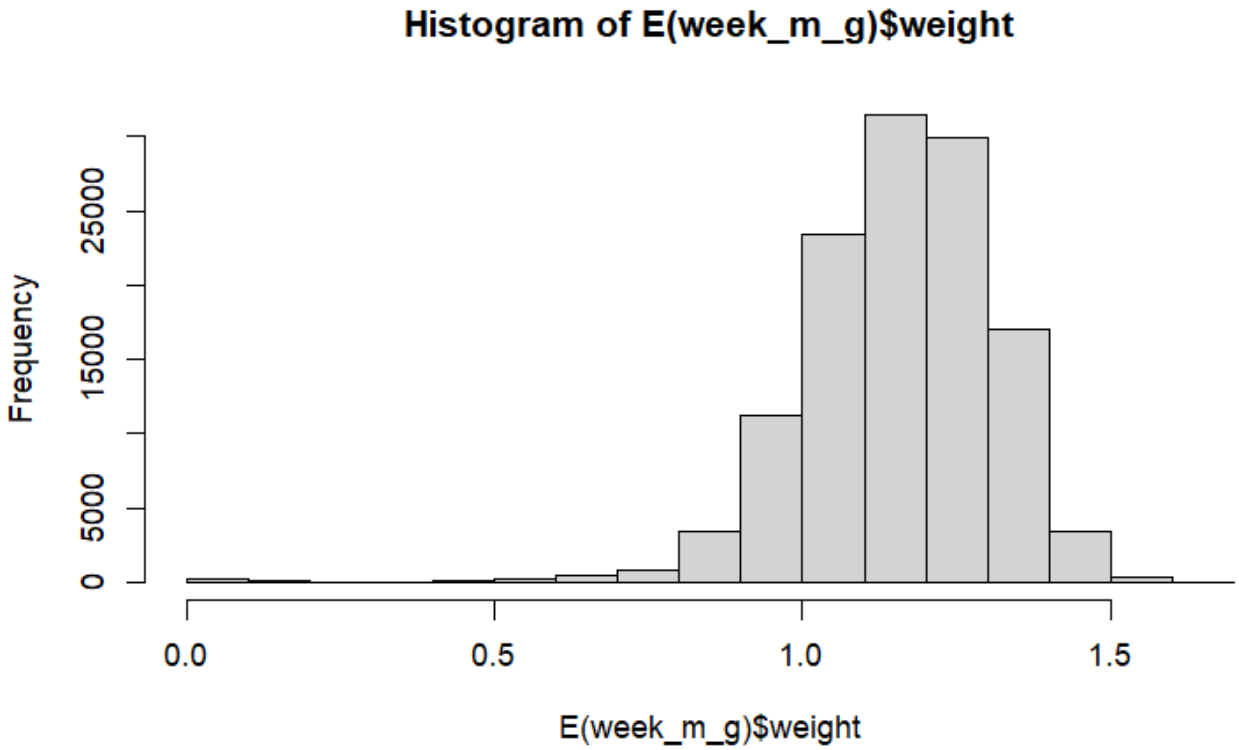


Figure 4: Un-normalized edge-weight distribution for stocks by week

We can see that the weight distribution is also roughlly binomial.

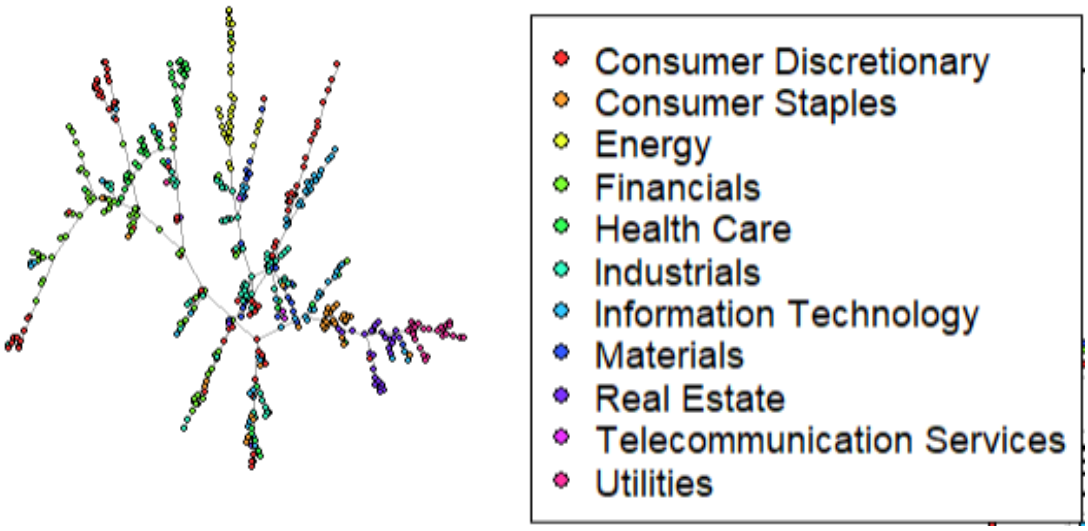


Figure 5: Color-coded MST with legend for weekly data

The MST also creates a "vine" structure with stocks grouped roughly by sector on each "vine".

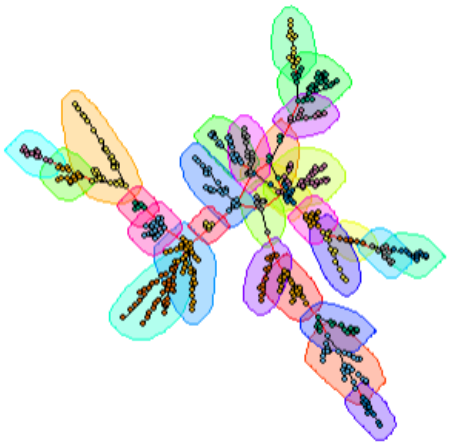


Figure 6: Communities within the weekly MST

This community algorithm has a homogeniety score of 0.5929015 and completeness score of 0.4189699.

Lastly, the weekly data gives an $\alpha 1$ score of 0.7441191 for the P that focuses on the nodes neighbors, and an $\alpha 2$ score of 0.1141881 for the P that compares the node's sector to all nodes.

Q7: Repeat questions 2, 3, 4, 5 on the MONTHLY data.

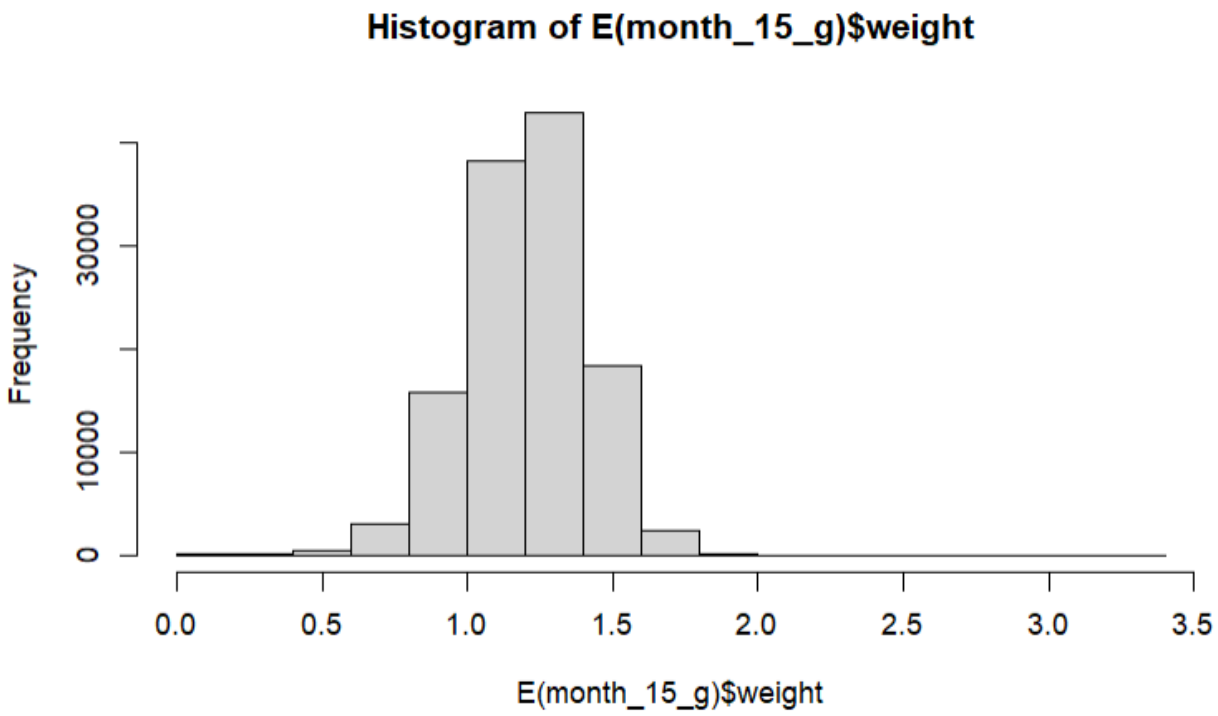


Figure 7: Un-normalized edge-weight distribution for stocks by month

We can see that the weight distribution is also roughlly binomial.

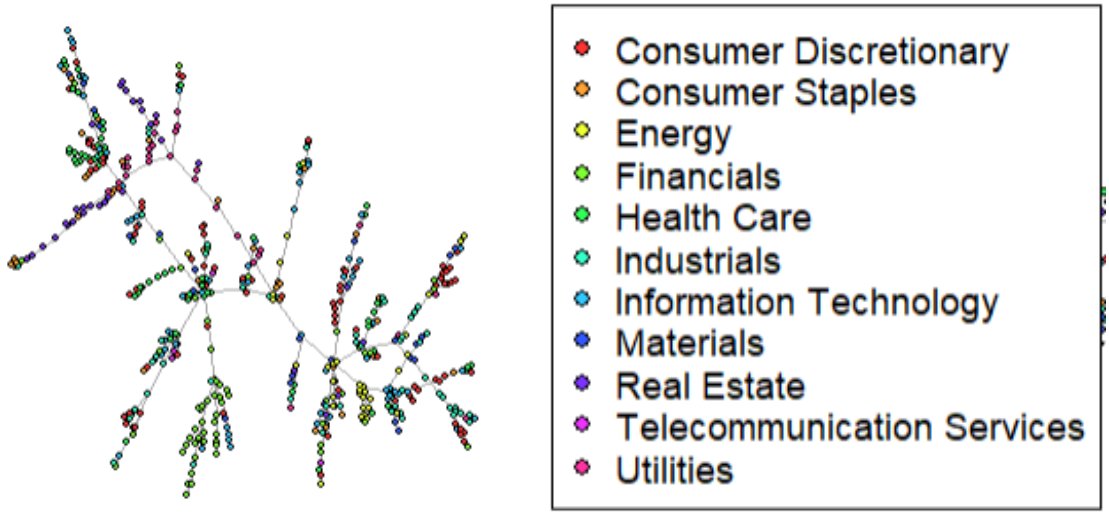


Figure 8: Color-coded MST with legend for monthly data

The MST also creates a "vine" structure with stocks grouped roughly by sector on each "vine".

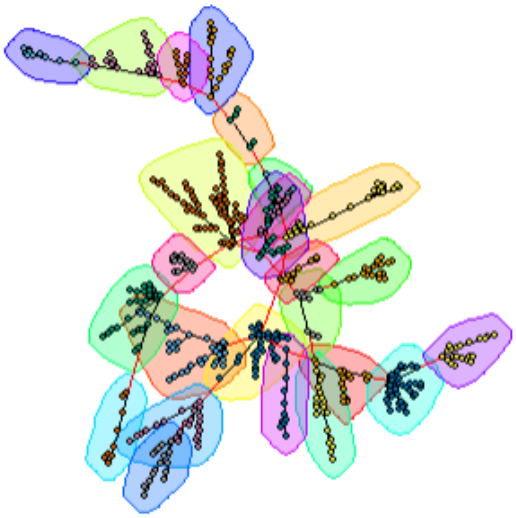


Figure 9: Communities within the weekly MST

The homogeneity score is 0.3367999 and the completeness score is 0.2485899.

The $\alpha 1$ score is 0.4673522 and the $\alpha 2$ score is 0.1141881.

Q8: Compare and analyze all the results of daily dta vs weekly data vs monthly data. What trends do you find? What changes? What remains similar? Give reasons for your observations. Which granularity gives the best results when predicting the sector of an unknown stock and why?

When comparing the edge weights and their distributions, we can see that all three data sets roughly follow a binomial distribution as there is a single distinct peak. All of the distributions' peak sits between1 and 1.5. Since we already know that w_{ij} will be between 0 and 2, this distribution is not too surprising. It also makes sense that all three versions are similar as, despite the decrease in information, the stocks are still correlated to each other strongly or poorly enough to keep the rough weight distribution the same.

Next, when looking at the MSTs, we can see that all three trees still contain at least one loop but the weekly and monthly trees both contain at least two loops. As we go from daily to weekly to monthly, we can see that the nodes are not grouped as closely together within their sectors. This is especially obvious when we compare the daily and monthly MSTs.

Table 1: Homogeneity and Completeness Scores

Time-Scale	**Homogeneity**	**Completeness**
Daily	0.6808973	0.5241226
Weekly	0.5929015	0.4189699
Monthly	0.3367999	0.2485899

Table 1 shows that the homogeneity and completeness scores get worse and worse as the time scale restricts the data further and further. As the amount of data is reduced by only selecting specific data points within the set of raw data, the computations and algorithms receive less and

less information to calculate how closely any two given stocks correlate. This results in edge weights that do not reflect the correlations as accurately as before, resulting in the MST not grouping all of the stocks of the same sector together, resulting in the walktrap algorithm not being able to group all of a sector's stocks together. This cascading effect directly causes the reduced homogeneity and completeness scores as the MST becomes less accurate.

Table 2: α scores

Time-Scale	$\alpha 1$	$\alpha 2$
Daily	0.8287325	0.1141881
Weekly	0.7441191	0.1141881
Monthly	0.4673522	0.1141881

Table 2 shows a similar pattern with the $\alpha 1$ score decreasing as the time scale increases. As expected, the $\alpha 2$ score remains the same across all of the tests as it does not depend on the way the MST is formed and instead relies on the number of stocks within a sector and the total number of stocks, both of which never change. The $\alpha 1$ score, however, relies on the neighbors of a given stock which is going to change as the MST changes. As previously noted, as we go from daily to weekly to monthly, the MSTs struggle to keep the stocks in a sector grouped together. This means that each stock's neighbor is less and less likely to be within the same sector as itself, reducing the $\alpha 1$ score.

Based on all of these observations, the daily breakdown provides the best results for predicting the sector of an uknown stock as it provided the highest homogeneity, completeness, and $\alpha 1$ scores, and the MST groups the stocks based on sector the best.

2 Let's Help Santa!

2.1 Download the Data

2.2 Build your Graph

Q9: Report the number of nodes and edges in G.

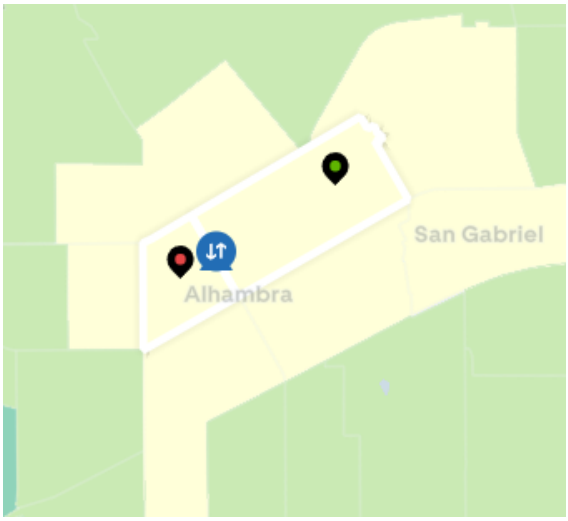
There are 2649 nodes and 1003858 edges.

2.3 Traveling Salesman Problem

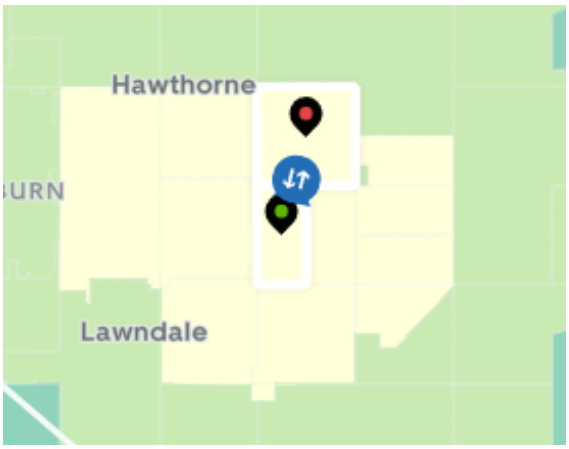
Q10: Build a minimum spanning tree (MST) of graph G. Report the street addresses near the two endpoints (the centriod locations) of a few edges. Are the results intuitive?

We sampled six edges spaced throughout all of the edges in the MST. First edge goes from 823 E Grand Ave, Alhambra, CA 91801 to 300 N 3rd St, Alhambra, CA 91801. The second edge goes from 3926 139th St, Hawthorne, CA 90250 to 13130 Florwood Ave, Hawthorne, CA 90250. The third edge goes from 12310 Sylvan St, North Hollywood, CA 91606 to 6240 Vantage Ave, North Hollywood, CA 91606. The fourth edge goes from 2302 S Gramercy Pl, Los Angeles, CA 90018 to 2069 S Oxford Ave, Los Angeles, CA 90018. The fifth edge goes from 5302 Lemoran Ave, Pico Rivera, CA 90660 to 6325 Pioneer Blvd, Whittier, CA 90606. The sixth edge goes from 9862 Ramm

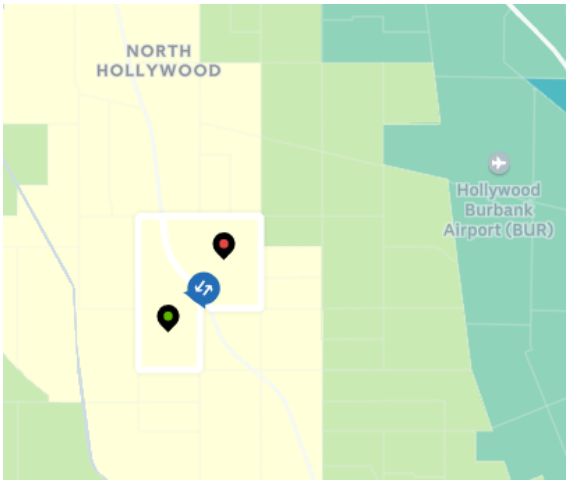
Dr, Anaheim, CA 92804 to 9671 W Colchester Dr, Anaheim, CA 92804. These trips are shown in the figures below.



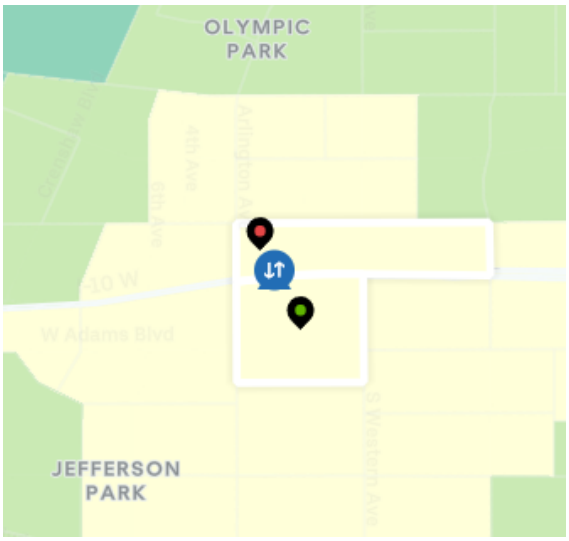
(a) Mapping of the first edge



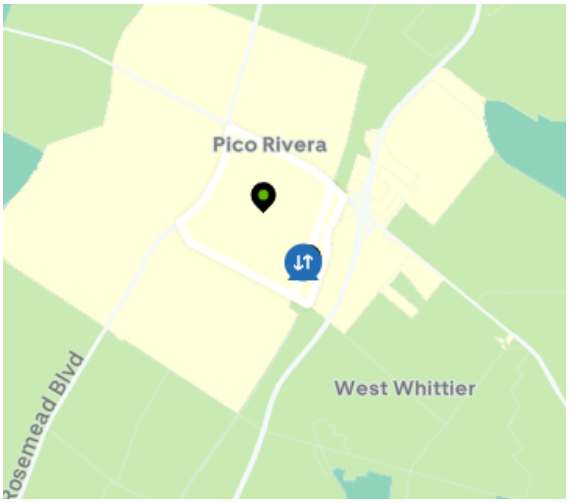
(b) Mapping of the second edge



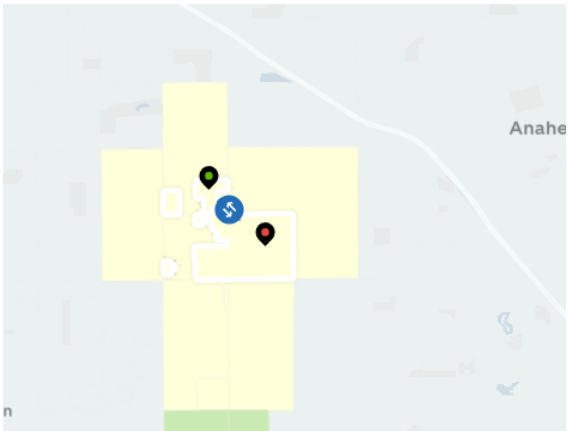
(c) Mapping of the third edge



(d) Mapping of the fourth edge



(e) Mapping of the fifth edge



(f) Mapping of the sixth edge

Figure 10: Mapping of the six selected edges

All of these trips took less than three minutes and seem to represent quick rides to and from various parts of a city all over Los Angeles County. This more or less follows what I would expect as longer trips would be less likely to appear as a result of the algorithm to create the minimum spanning tree.

Q11: Determine what percentage of triangles in the graph (sets of 3 points on the map) satisfy the triangle inequality. Estimate by randomly sampling 1000 triangles.

About 92% of the triangles in the graph satisfy the triangle inequality.

Q12: Find an upper bound on the empirical performance of the approximate algorithm:

$$p = \frac{ApproximateTSPCost}{OptimalTSPCost}$$

The 1-approximate algorithm instructs us to first generate a MST, then create a multigraph using two copies of each edge in the MST, and then find an eulerian walk and embedded tour in order to find a solution to the travelling salesman problem. We have already created a MST for this data, and once we create a multigraph and find a eulerian walk and embedded tour, we can try to find the approximate TSP cost. We tested 50 different starting nodes and selected the one that gave the lowest total cost. The lowest total weight we achieved was 425065.355. The provided reading indicates that the optimal TSP cost can be substituted for the total weight of the MST, so we will use that in this instance. The reading also indicates that the maximum upper bound of the approximate TSP cost will be twice the total weight of the MST. This means:

$$1 \leq \frac{ApproximateTSPCost}{OptimalTSPCost} \leq 2$$

When we try to calculate this ratio using our graphs, we get:

$$p = \frac{425065.355}{269084.545} = 1.5797$$

Q13: Plot the trajectory that Santa has to travel!

We can use the embedded tour from the calculations in the previous question to map the suggested optimal path and see if it intuitively makes sense to us.

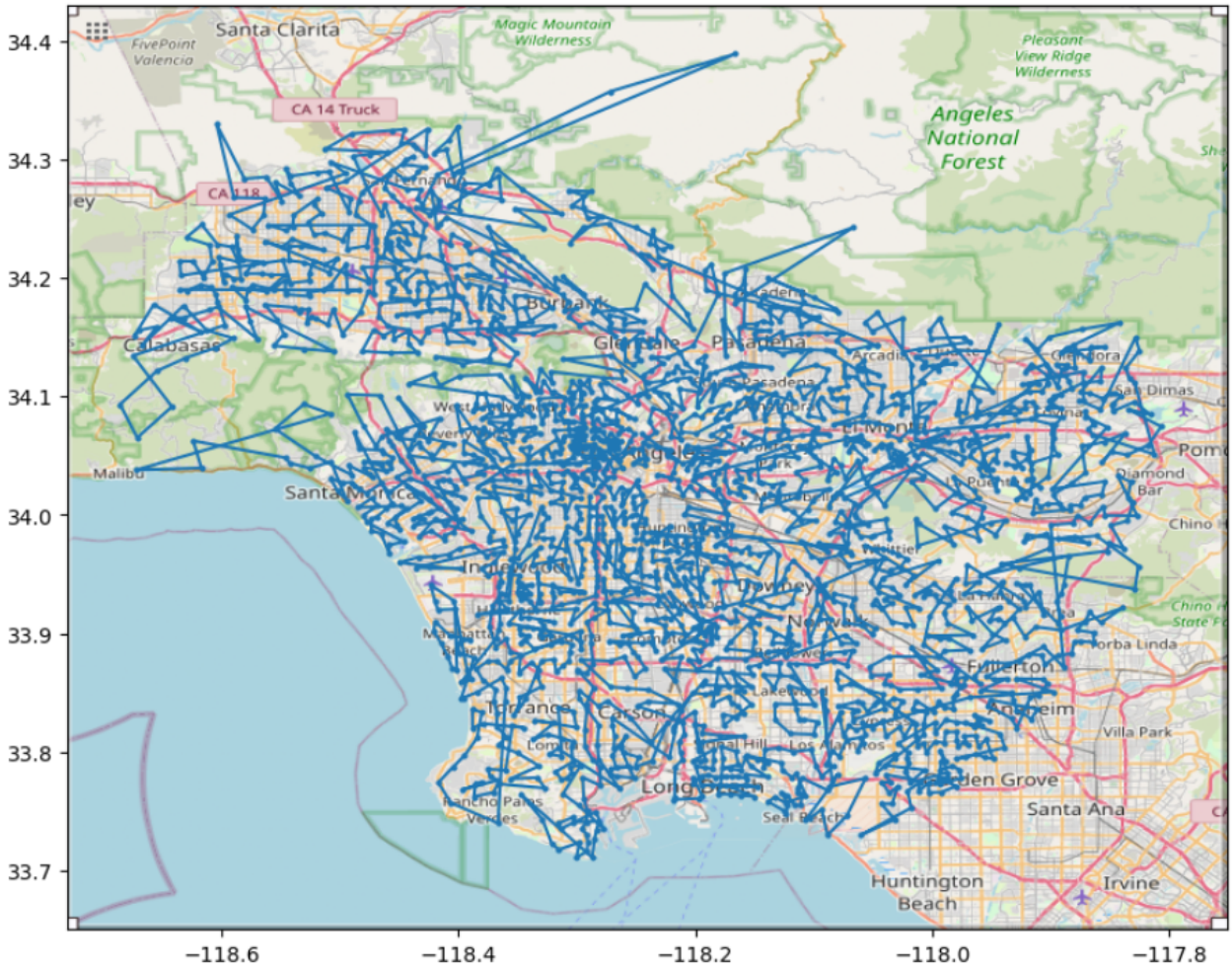


Figure 11: Optimal path for Santa to take

When we look at the map in Figure 11, we can see that path clearly snakes around the map, connecting each point. For the most part, the path seems to take as small of an edge as possible between nodes with the only noteable exceptions for nodes that are far away from the rest of the graph where the edge would necessarily be much longer. Intuitively, this path does seem to make sense as the path does not widly cross back and forth between distance nodes and tries to move from one node to a close neighbor.

2.4 Analysing Traffic Flow

2.5 Estimate the Roads

Q14: Plot the road mesh that you obtain and explain the result. Creat a graph $G\triangle$ whose nodes are different locations and its edges are produced by triangulation.

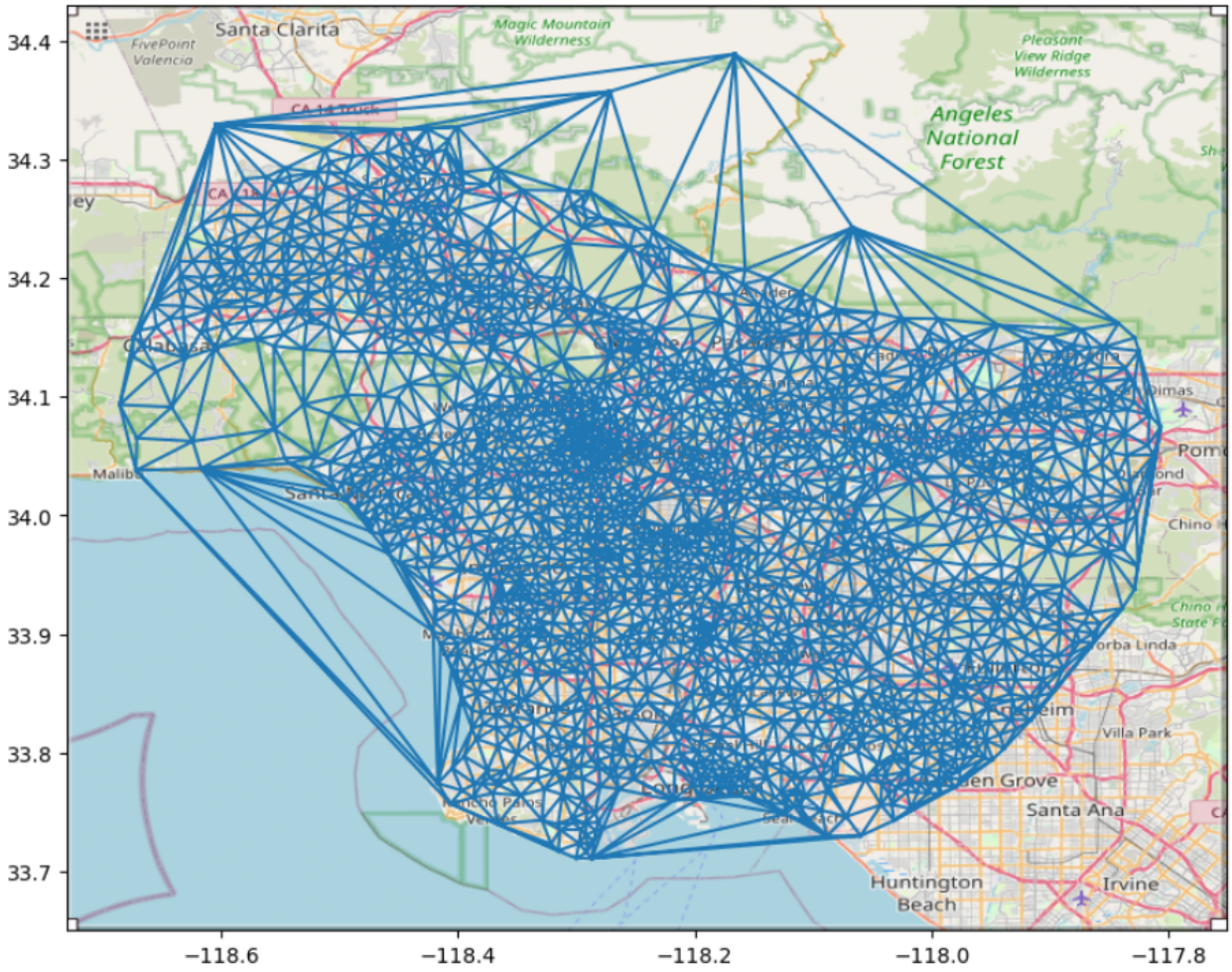


Figure 12: Road mesh produced from the Delaunay triangulation algorithm

Figure 12 shows the resulting road map. We can see that there is a very dense network of edges near the center of LA and in the surrounding cities. The less populated areas have much sparser edges with the outlier nodes connected by extremely long edges. The higher concentration of locations and nodes within cities causes the algorithm to generate more and more edges or "roads" in this instance.

We then create the graph $G\triangle$ from a subgraph of the GCC with the edges from the Delaunay triangulation algorithm, making sure to not create duplicate edges.

2.6 Calculate Road Traffic Flows

Q15: Using simple math, calculate the traffic flow for each road in terms of cars/hour. Report your derivation

The derivation to find the traffict flow of a given road is as follows:

- Determine the coordinates of the beginning and end of the road
- Calculate the distance in coordinate degrees the road covers by using $a^2 + b^2 = c^2$ where a is the differenc in latitude, b is the difference in longitude, and c is the hypotenuse of the resulting triangle.

- Multiply the distance in the previous step by 69 based on the assumption that 1 degree is equivalent to roughly 69 miles in order to obtain the length, D, of the road.
- Determine the weight of the corresponding edge in the graph G as this is the time, t, it took for a car to travel that distance in seconds.
- Determine the speed, s, the car was travelling at by dividing the length of the road by the time and then dividing by 3600 to convert the result into miles per hour.
- Based on the assumption that one car is 0.003 miles long and each car has a 2 second safety distance, sd, between each other, we can multiply the speed of the car times the 2 seconds converted into hours to determine this distance.
- Determine how many cars fit into one mile, cpm by dividing the length of the road plus the safety distance by the length of a car plust the safety distance.
- With the additional assumption that each road has 2 lanes, we can then calculate the number of cars that can travel on the road in one hour, cph, by multiplying the number of cars that fit in one mile, times the total length of the road, times 2, and then divide by the weight of the road converted into hours.

To convert this into equations with the corresponding variables would be:

- $D = \sqrt{(a^2 + b^2)} * 69$
- $s = \frac{D}{(t/3600)}$
- $sd = s * \frac{2}{3600}$
- $cpm = \frac{D+sd}{0.003+sd}$
- $cph = \frac{cpm*D}{t/3600}$

This process was then performed for each edge in $G\triangle$ and gave a total flow of 103301253.33 cars per hour over 7788 roads.

2.7 Calculate Max Flow

Q16: Calculate the maximum number of cars that can commute per hour from Malibu to Long Beach. Also calculate the number of edge-disjoint paths between the two spots. Does the number of edge-disjoing paths match what you see on your road map?

There are 4 edge-disjoing paths between Malibu and Long Beach meaning there are 4 different routes that any given person can take.

Table 3: Max flow for each path.

Path	Nodes	Total cars per hour
1	[1511, 1509, 1512, 1941, 1950, 1584, 1590, 2406, 1612, 1701, 1783, 1860, 1861, 1863, 1862, 1643, 1642, 1641, 1638, 684, 2373, 664, 663]	733390.52
2	[1511, 1510, 1700, 1950, 1583, 1582, 1581, 1588, 1590, 1701, 1704, 1783, 1859, 1860, 1658, 1660, 1661, 2422, 2373, 663]	790372.34
3	[1511, 1512, 1700, 1590, 1612, 324, 1702, 1704, 426, 1783, 1782, 1857, 1882, 1861, 1658, 1657, 1662, 1661, 2373, 653, 663]	885480.79
4	[1511, 1700, 1584, 1588, 1589, 2406, 1604, 1605, 1685, 319, 320, 1705, 1864, 562, 564, 2015, 2016, 2017, 2134, 2365, 163, 165, 170, 610, 647, 651, 652, 654, 663]	846284.60

Table 3 shows all four of the paths and their corresponding nodes and maximum flow per hour. We can see that Path 3 has the highest flow with 885480.79 cars per hour. All four paths combined provides a total of 3,255,528.25 cars per hour can travel from Malibu to Long Beach.

The number of edge disjoint paths makes sense as, when we check the degree of the Malibu and Long Beach nodes, we can see that they have degrees of 4 and 6, respectively. This means that there could only be, at most, 4 edge disjoint paths.

2.8 Prune Your Graph

Q17: Plot $\tilde{G}\triangle$ on actual coordinates. Do you think the thresholding method worked?

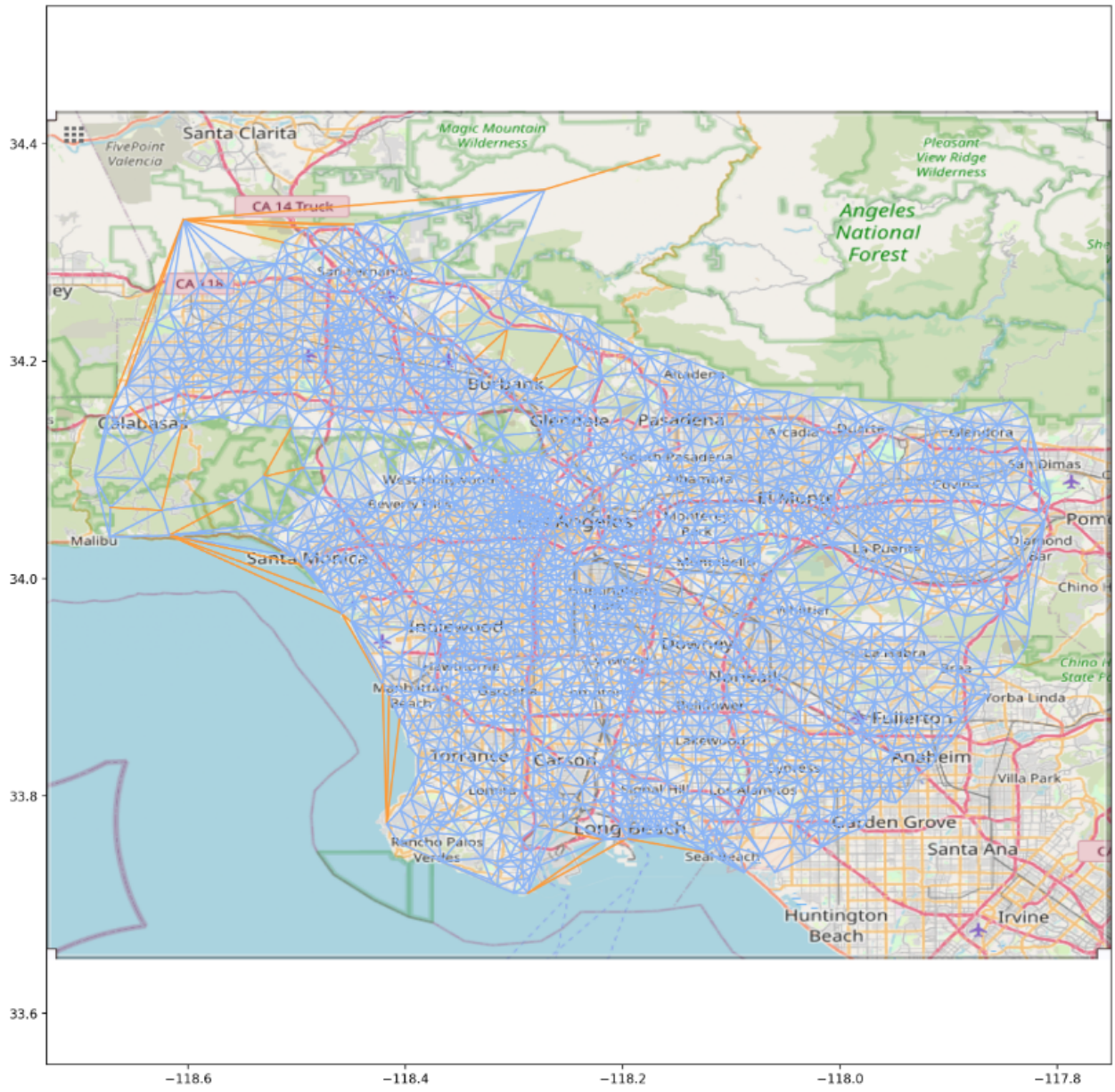


Figure 13: Graph $G\triangle$ without edges above the trim threshold of 800

Figure 13 shows the edges that would be trimmed in orange and the edges that would be kept in blue. We can see that it seems to work very well in removing the edges that travel across the ocean where roads clearly do not exist. It did, however, eliminate some of the roads that must exist for the outlier nodes.

Q18: Now, repeat question 16 for $\tilde{G}\triangle$ and report the results. Do you see any changes? Why?

Table 4: Max flow for each new path.

Path	**Nodes**	**Total cars per hour**
1	[1511, 1509, 1508, 1505, 1504, 1503, 1520, 1533, 1545, 1547, 1554, 1560, 244, 246, 251, 1600, 1609, 1610, 323, 1705, 1864, 562, 564, 2015, 2019, 2021, 2031, 1620, 2367, 158, 160, 166, 170, 1676, 684, 2373, 664, 663]	867508.97
2	[1511, 1510, 1509, 1507, 1506, 1504, 1502, 112, 1545, 1546, 1569, 1571, 249, 1592, 1599, 1607, 1611, 319, 323, 410, 409, 561, 560, 2399, 2015, 2016, 2017, 2134, 2365, 163, 165, 170, 610, 647, 651, 2373, 663]	722187.79
3	[1511, 1512, 1941, 1950, 1584, 1590, 2406, 1612, 1701, 1704, 426, 429, 1871, 1873, 1875, 1782, 1783, 1860, 1861, 1658, 1660, 1661, 1686, 2421, 2373, 653, 663]	783060.01
4	[1511, 1700, 1941, 1939, 1950, 1583, 1584, 1588, 1589, 2406, 1604, 1612, 324, 320, 1705, 1706, 1866, 1868, 2020, 2021, 2030, 2032, 1621, 2368, 159, 162, 1627, 1636, 1638, 684, 651, 652, 654, 663]	920679.37

After removing the roads over a certain length, the new paths travell through significantly more nodes than before. The highest max flow for a path, Path 4, is now 920,679.37 cars per hour and all the paths have a combined total of 3,293,436.15 cars per hour. Both of these numbers are higher than the previous results. If we take a look and compare the two sets of paths, we can make a few more observations.

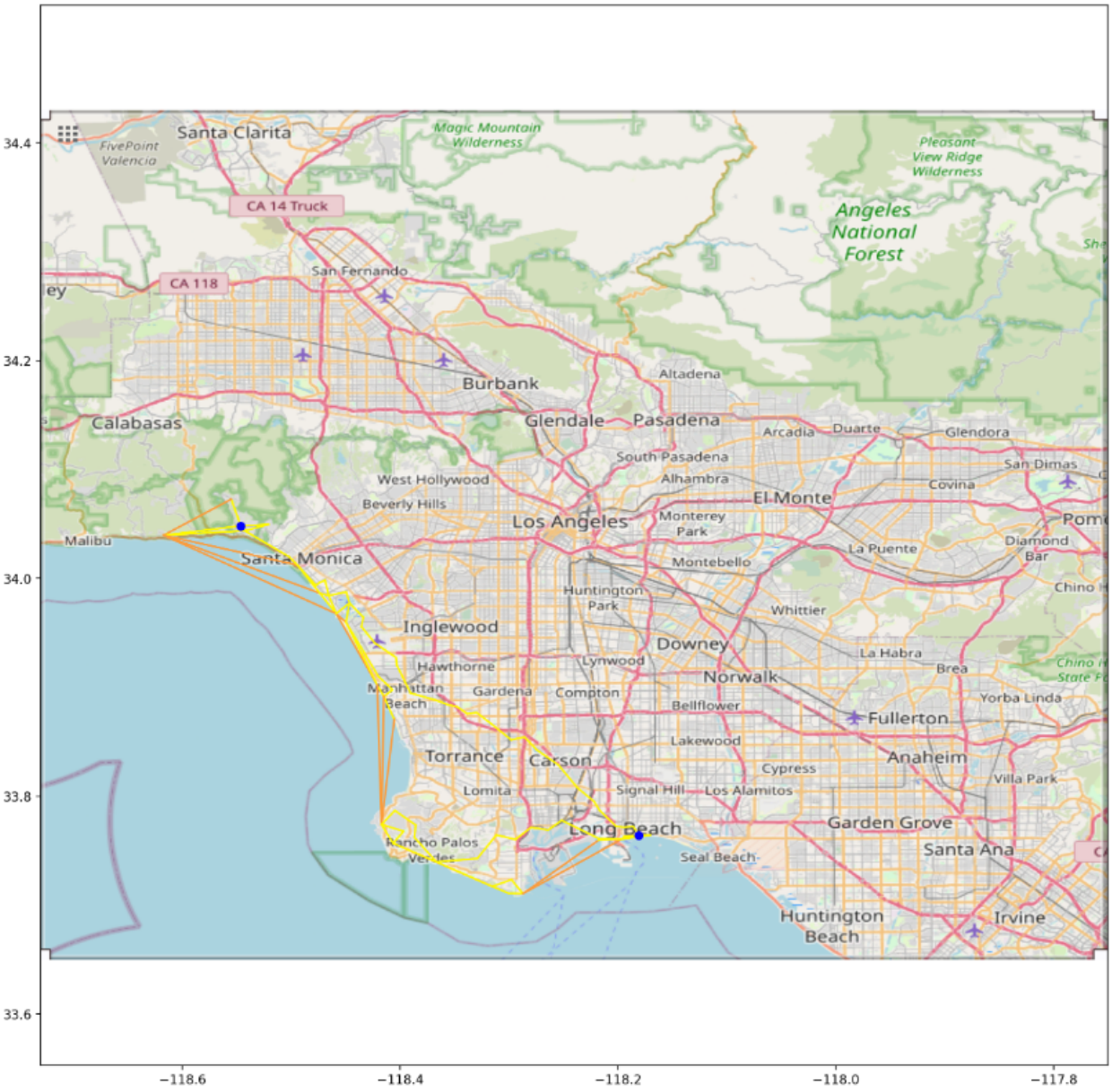


Figure 14: Edge disjoint paths prior to removing threshold roads

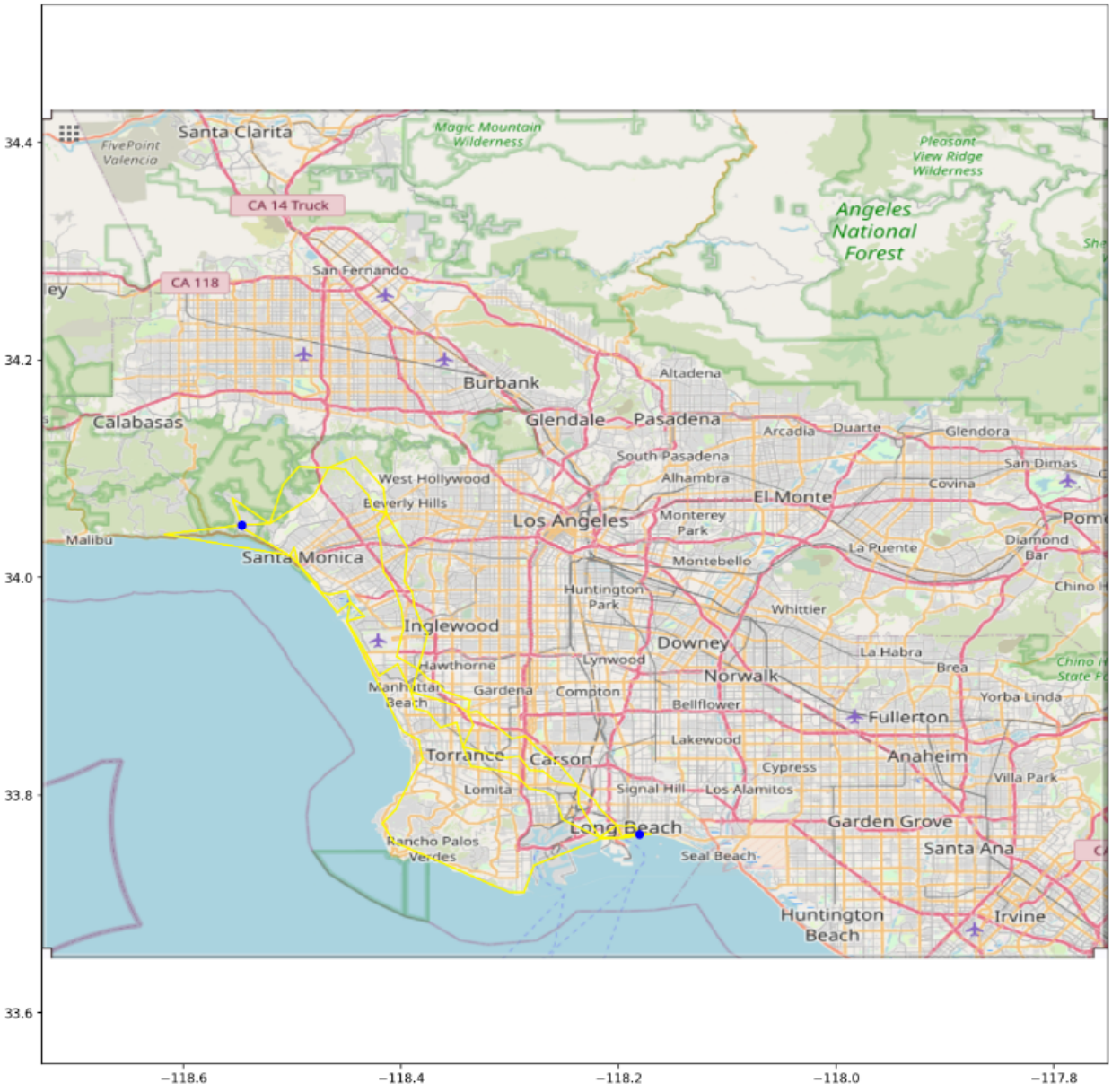


Figure 15: Edge disjoint paths after removing threshold roads

Figure 14 shows the original paths prior to removing the longer roads. The yellow indicates roads that would not be removed by the threshold and the orange represents roads that would be removed. Figure 15 shows the new paths after the threshold roads are removed. We can clearly see that these paths are fairly different with only one matching path between the two figures. The new paths now travel northward first before moving south towards Long Beach instead of making use of the non-existent roads across the ocean. As a result, the paths appear somewhat less direct which helps explain the overall higher total max flow as a longer travel path means more cars can fit.

These maps also show why the number of edge disjoint paths did not change as the roads directly connected to both the source and destination nodes do not change as they do not exceed the threshold limit.

2.9 Construct New Roads

Q19: Strategy 1 (geo_distance, static): Reduce the maximum extra travelling distance. The extra travelling distance between 2 locations is the difference of the shortest traveling distance and the straight distance. i.e. extra_distance(v, s) = distance_of_shortest_path(v,s) - euclidean_distance(v, s)

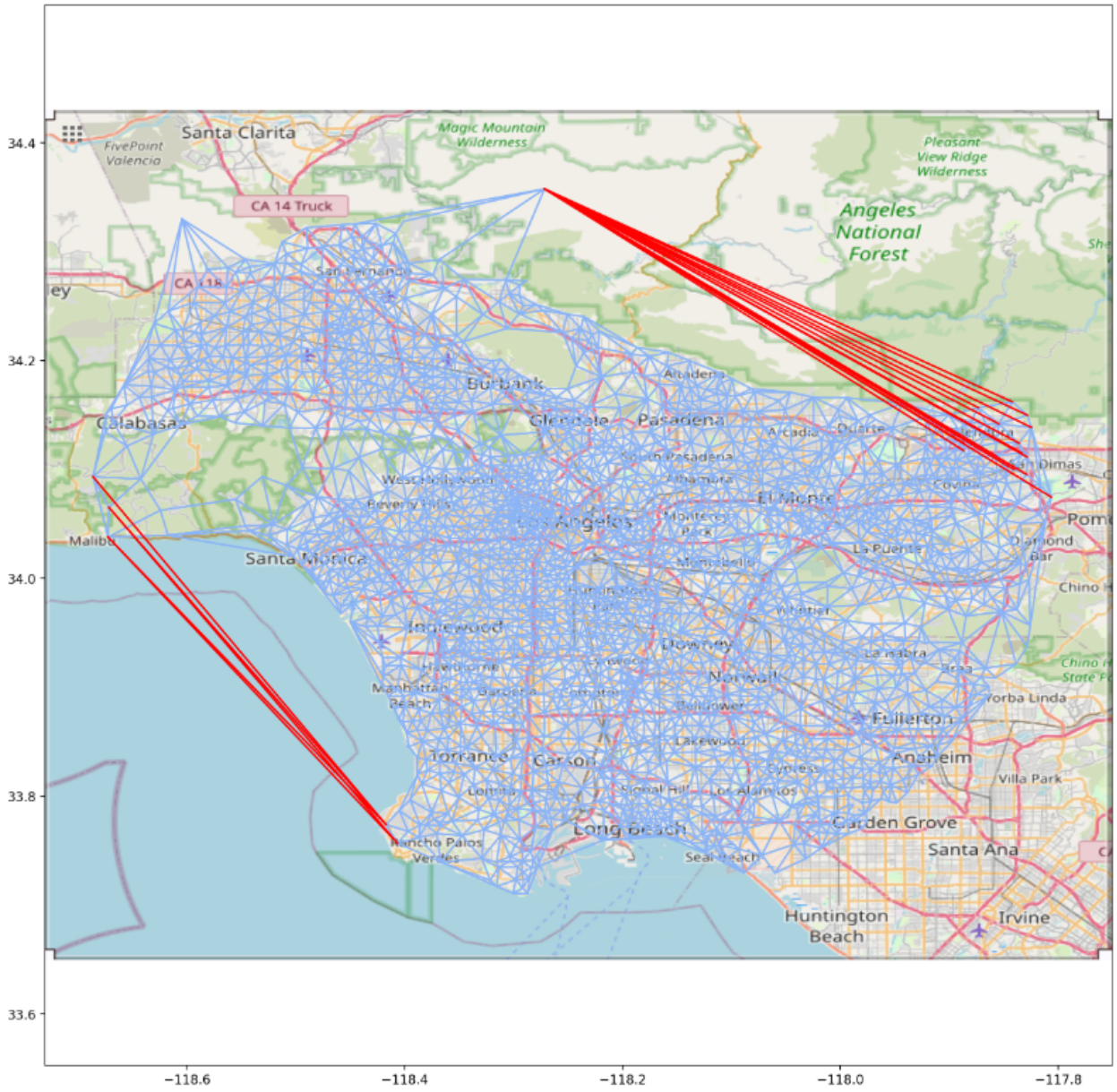


Figure 16: New map with the added edges colored in red and the original edges colored in blue.

Figure 16 shows the 20 roads added by finding the largest difference between the shortest path and euclidean distance between two nodes. These added roads connect nodes on the outer edges of the map that would have a more roundabout route between each other. This makes sense as these nodes would have much longer paths than a straight line between them. This process, however, also added back in roads that go across the ocean and logically do not exist.

Table 5: Each new edge's start and end nodes.

Edge	**Nodes**
1	[392, 2419]
2	[1902, 2419]
3	[2158, 2419]
4	[1906, 2419]
5	[1783, 2413]
6	[45, 2419]
7	[1904, 2419]
8	[390, 2419]
9	[386, 2419]
10	[2163, 2419]
11	[1699, 1860]
12	[381, 2419]
13	[1901, 2419]
14	[391, 2419]
15	[1699, 1783]
16	[383, 2419]
17	[382, 2419]
18	[1860, 2416]
19	[2140, 2419]
20	[1783, 2416]

This strategy requires us to first, find the shortest path between all of the nodes based on the existing edges. This process, Dijkstra's algorithm in this case, typically takes $O(V^2)$ time where V is the number of nodes. Next, we need to calculate the euclidean distance between every node which would take us $O(V * (V-1)/2)$ time as we do not need to compute the distance between a node and itself and the distance remains the same regardless of the order of the nodes. We necessarily must loop through each node in the graph, meaning we end up with a final time complexity of $O(V^3 * (V-1)/2)$.

Q20: Strategy 2 (geo_distance, static, with_frequency): Use the frequency as the weight to multiply the difference

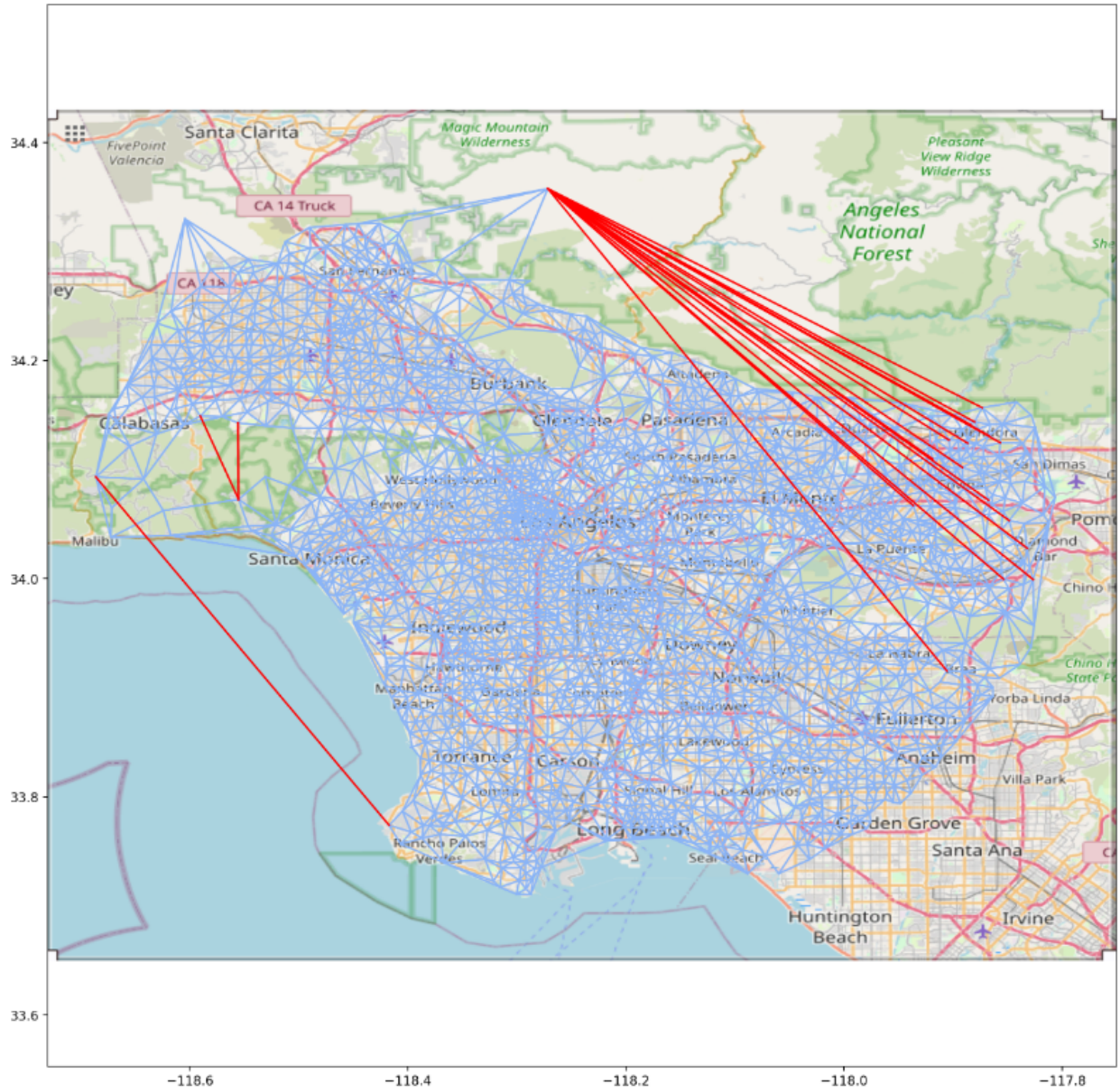


Figure 17: New map with the added frequency-based edges colored in red and the original edges colored in blue.

This new map has similar edges to those in Figure 16 with the outer nodes created edges between each other. In this case, however, more edges form from a single outlier node with few edges forming over the ocean. The frequency multiplication appears to have given more weight to certain edges which likely had an original difference that was only marginally smaller than the other paths created in Figure 16.

Table 6: Each new edge's start and end nodes.

Edge	**Nodes**
1	[2419, 2464]
2	[384, 2419]
3	[1005, 1510]
4	[2072, 2419]
5	[252, 2419]
6	[231, 2419]
7	[2164, 2419]
8	[388, 2419]
9	[2142, 2419]
10	[2141, 2419]
11	[2052, 2419]
12	[257, 2419]
13	[383, 2419]
14	[2057, 2419]
15	[49, 2419]
16	[2148, 2419]
17	[989, 1510]
18	[2163, 2419]
19	[389, 2419]
20	[1783, 2413]

This version has a very similar time complexity as it requires the difference data from Question 19. The only change is that there is an additional step where each difference now needs to be multiplied by a random number. Multiplication can end up being very resourse expensive as it typically has a time complexity of $O(n^2$ where n represents the number of digits of the largest number being multiplied. So, if we have to perform the multiplication $V * (V-1)/2$ times, the same number of loops as the previous question, we can simply multiply the multiplication time complexity with the time complexity from question 19, giving us $O(n^2 * V^3 * (V-1)/2)$.

Q21: Strategy 3 (geo_distance, dynamic): Create the roads one by one. Repeat 20 times: i) Compute extra_distance between all the pairs in the graph. ii) Create a road between the pair with highest extra_distance and update the graph. iii) Print the source and desitnation of this new edge. Plot the final graph on actual coordinates. What is the time complexity of this strategy?

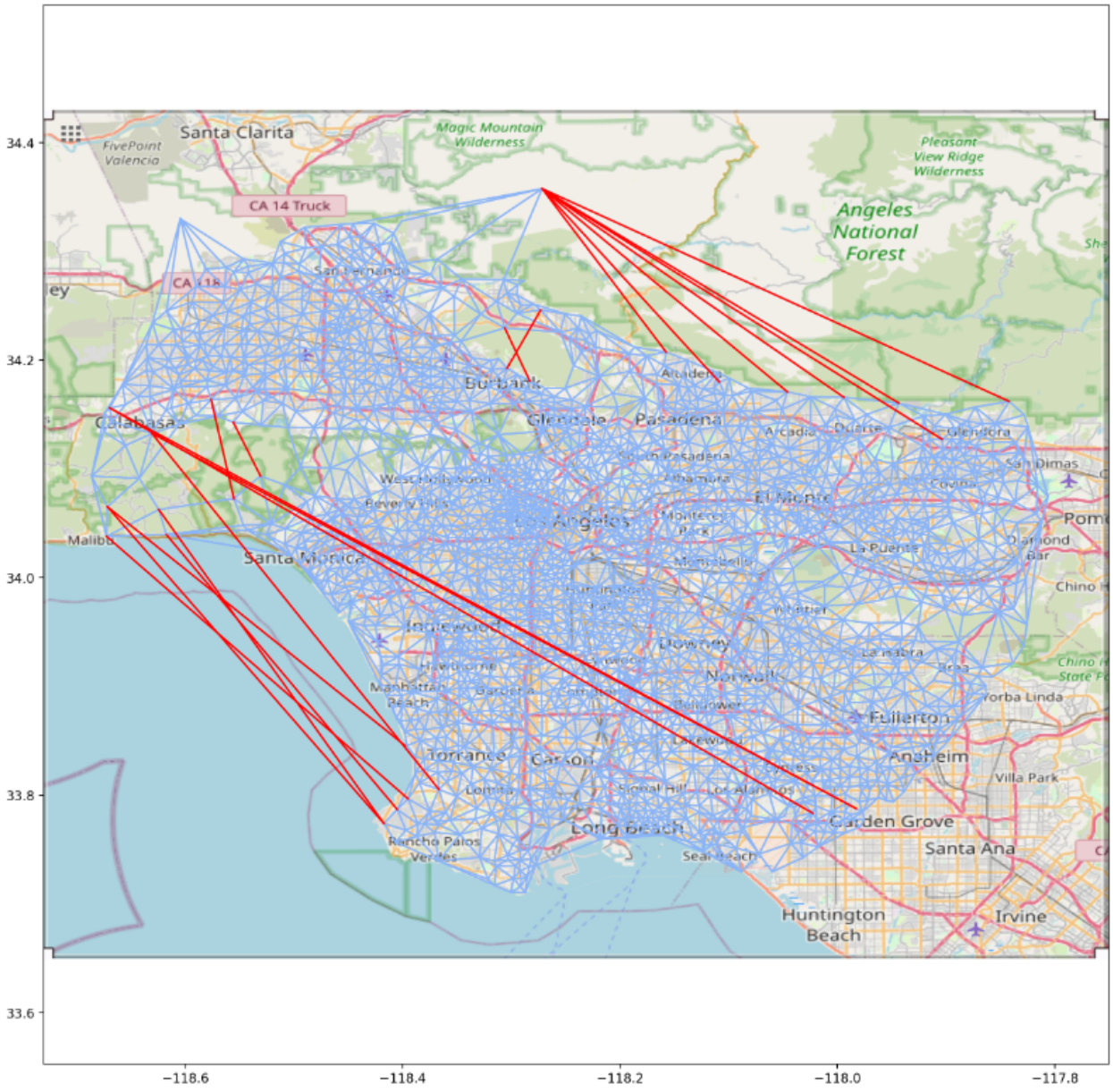


Figure 18: Dynamically created roads are colored in red and the original roads are colored in blue.

When we compare this map with the one created in Figure 16, we can see that there are a few edges that both methods produced, [1783, 2416] and [2140, 2419]. There are also several edges in the top right and bottom left quadrants of the map that connect different nodes in the two maps, but serve a similar purpose in providing shorter routes for outlier nodes. The main difference between the two is the production of edges spanning the diagonal of the map to connect nodes on opposite ends. These roads would be similar to freeways or highways which enable drivers to traverse long distances and avoid a lot of the traffic and congestion of surface roads. The dynamic generation also created a few shorter edges to fill in gaps where there are blank areas of the map.

Table 7: Each new edge's start and end nodes.

Edge	**Nodes**
1	[1783, 2416]
2	[2140, 2419]
3	[986, 1510]
4	[49, 2419]
5	[285, 2419]
6	[1717, 2419]
7	[1783, 2417]
8	[1956, 2419]
9	[2247, 2419]
10	[1005, 1678]
11	[2414, 2619]
12	[121, 689]
13	[2242, 2419]
14	[1699, 1781]
15	[144, 2619]
16	[2402, 2416]
17	[144, 2040]
18	[1700, 1782]
19	[356, 1679]
20	[2414, 2559]

This strategy takes significantly longer than the previous ones as the entire process is essentially repeated 20 times. The time complexity is equivalent to the time complexity from Question 19 times 20. This would give us $O(20 * V^3 * (V-1)/2)$, or, if we want to generalize this, $O(X * V^3 * (V-1)/2)$ where X is the number of iterations.

Q22: Strategy 4 (Travel time, static): We want to optimize to reduce the maximum extra travelling time. Use the coordinates of v and s to get the euclidean distance betwen them. Calculate the extra time between all pairs of points. Print the source and destination of the top 20 pairs and plot them. What is the time complexity?

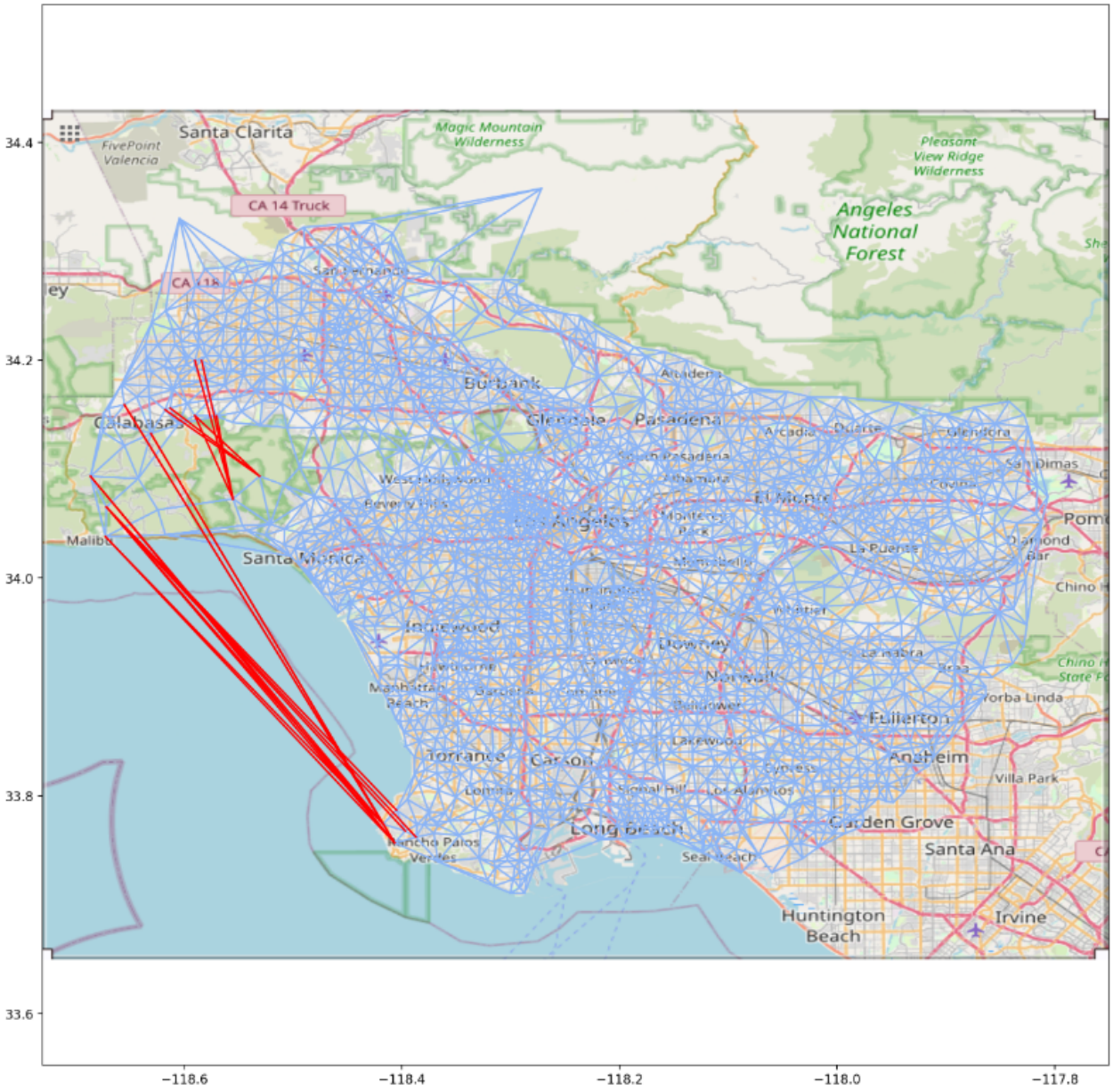


Figure 19: Map showing the edges produced based on travel time in red with the original edges in blue.

This mapping in Figure 19 shows that this version prioritizes creating edges across the ocean and to cut straight through the small mountain range separating Calabasas/Tarzana from Westwood/Pacific Palisades.

Table 8: Each new edge's start and end nodes for the travel time map

Edge	**Nodes**
1	[984, 1678]
2	[951, 1510]
3	[430, 1860]
4	[1882, 2416]
5	[989, 1678]
6	[950, 1510]
7	[430, 1783]
8	[144, 1860]
9	[144, 1783]
10	[989, 1510]
11	[1782, 2416]
12	[988, 1510]
13	[985, 1678]
14	[1699, 1860]
15	[1699, 1783]
16	[1860, 2413]
17	[1859, 2416]
18	[1783, 2413]
19	[1860, 2416]
20	[1783, 2416]

The time complexity for this version would be similar to the static distance version, however, we must now also compute the path with the shortest travel time. The reason we still need to compute the shortest distance path is in order to calculate the speed for that trip. This means that we will need to run the Dijkstra algorithm twice, once for distance as the weight and once for time as the weight. This will increase the time complety to $O(V^5 * (V-1)/2)$

Q23: Strategy 5 (Travel time, dynamic): Repeat 20 times: i) Compute extra_time between all the pairs in the graph. ii) Create a road between the pair with highest extra_time and update the graph. iii) Print the source an desitnation of this new edge. Plot the final coordinates. What is the time complexity?

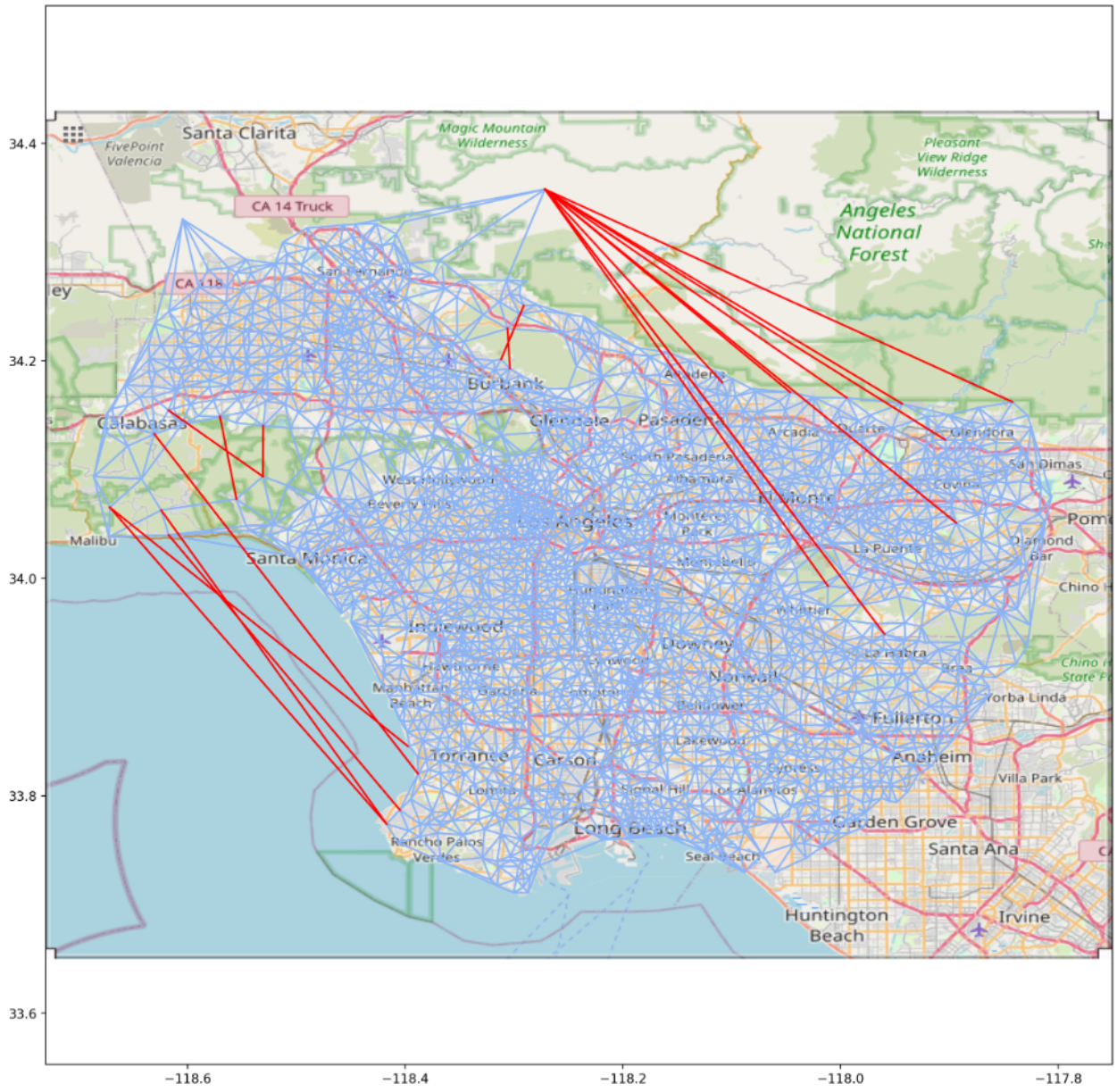


Figure 20: Map showing the edges dynamically produced based on travel time in red with the original edges in blue.

This new mapping is similar to the dynamic distance one without a few of the edges spanning the entire diagonal of the map. It still creates a diverse array of roads to help alleviate a wider range of traffic problems.

Table 9: Each new edge's start and end nodes for the dynamic travel time map

Edge	Nodes
1	[1783, 2416]
2	[985, 1678]
3	[988, 1510]]
4	[2140, 2419]]
5	[1783, 2417]
6	[2081, 2419]
7	[49, 2419]
8	[285, 2419]
9	[2262, 2419]
10	[2273, 2419]
11	[1717, 2419]
12	[2402, 2416]
13	[1003, 1678]
14	[1964, 2419]
15	[1956, 2419]
16	[1700, 1782]
17	[120, 687]
18	[2247, 2419]
19	[121, 1679]
20	[144, 1875]

The time complexity for this method would follow a simlar pattern as the dynamic distance one in that it is the time complexity of the static one times the number of times that the process is run. This means that it would be $O(20 * V^5 * (V-1)/2)$

Q24: Strategy comparison. Compare the following strategies:

2.9.1 a) 1 vs 2 - compare and analyze the results. Which is better? Why?

Strategy 1 focuses on solely the distance to determine which roads to construct while Strategy 2 also takes into account the frequency of how often a given road is used. Although this frequency is a random number, it can help highlight shorter potential roads that could help alleviate dense areas of traffic. In this case, the algorithms ended up creating most of the roads to cut across the Angeles National Forest to connect a singular outlier node. It also created a few additional edges on the other side of the map across the ocean and across Topenga State Park. In terms of creating a more idea road netowrk, Strategy 2 is somewhat better than Strategy 1 if only because it creates a variety of roads to serve different locations and groups of people.

2.9.2 b) 1 vs 3 - compare and analyze the results. Which is better? Why?

Strategy 3 uses distance like Strategy 1 but generate the roads dynamically instead of statically. This means that the algorithm re-evaluates which road would be best to form after each iteration. In theory, this should generate a better road network as it can take into account how a new road will affect the travel distance for a given trip. The resulting mapping of this strategy provides a

much more diverse selection of roads than Strategy 1 and 2 and is consequently better than both strategies purely in terms of reducing the overall distance needed to travel. Much like Strategy 1 and 2, though, it does create roads across the ocean which, in real life, are impractical to build over such a long distance.

2.9.3 c) 1 vs 4 - compare and analyze the results. Which is better? Why?

Strategy 4 uses time to determine which roads to construct which, in theory, can be a better measure of actual impact on traffic and traffic flow. The mapping for this strategy, however, only creates edges across the ocean and Topenga State Park. This is a result of a driving having to take a longer, more time-consuming path to reach the southern parts of the map from the north-western parts. Although time can be a good measure of areas of high traffic that need to be alleviated, it can also be skewed by instances where this only a long, roundabout path between two nodes. For this reason, Strategy 4 is not necessarily better than Strategy 1 which generated roads only across the ocean and the Angeles National Forest. Strategy 1 provided more roads to serve a wider range of people, making it slightly better.

2.9.4 c) 1 vs 5 - compare and analyze the results. Which is better? Why?

Lastly, we will compare Strategy 1 and Strategy 5. Strategy 5 dynamically generates roads based on travel time. The resulting mapping creates a much more diverse selection of roads than the static time strategy did, making a more usefull road network. This also means that it creates a much better road network than Strategy 1 as it not only connects the outlier nodes, it also generates a few roads to bridget other gaps in the map where additional roads and paths would help reduce the distance and time needed to travel between nodes.

2.9.5 d) Statics vs dynamic - Considering we want to improve the overall road network by constructing new roads, is either of them the optimal strategy?

Strictly in terms of improving the overall road network, dynamic methods are much better than static methods. By creating a road and then seeing the affect it has on traffic, travel times, and travel distances, we can create more roads to resolve more problems. In terms of time complexity, however, dynamic methods are extremely costly. Since the entire process must be repeated for however many roads need to be produced, the amount of time it takes constantly increases. Regardless, some combination of dynamic time, distance, and frequency would likely be more ideal than any singular static strategy alone. As mentioned, the dynamic strategies are able to adapt to the changes caused by creating new roads to be able to solve more problems in more areas of a map.

2.9.6 e) Open ended question: Come up with a new strategy.

On paper, the most naive and simplistic (and most unrealistic) way to optimize traffic and reduce travel time and distance is to create roads connecting every single node to every other node on the map such that each node's neighbors consists of every single other node. Obviously, in real life this would be incredibly impractical and extremely cost prohibitive.

A more interesting and potentially better method would be to dynamically generate roads based on the average speed for a given path. This would be somewhat similar to the time methods, however, basing the decision off of speed instead would help pinpoint areas with dense traffic that have a very slow average speed. It is likely that this method would not generate all of the same roads as the tested methods such as the ones crossing the ocean or national forests as it would, in

theory, target denser, traffic heavy areas. Dynamically producing these roads would allow us to target more areas of dense traffic

3 Define Your Own Task

In this section, we will examine the effects that the Covid-19 shutdowns had on delivery services such as UPS, FedEx, etc. More specifically, we want to see if the reduced amount of traffic could have allowed these services to alter their routes to be more efficient.

For context, package delivery services often route their deliveries to specifically avoid left turns whenever possible. The reasoning behind this is not only to avoid potential accidents but also because left turns, on average, have significantly longer wait times than right turns. For example, a delivery truck will opt to take three rights instead of a left, or the route will be designed such that left turns are not needed in order to create a route that spends less time waiting at a signal to turn. The Covid-19 shutdowns significantly reduced the amount of vehicles on the road during working hours, potentially reducing the amount of time a truck might have to wait at a signal. We do not have the specific data indicating how these shutdowns affected wait times for turns, but we will make a few basic assumptions for the sake of experimentation.

First, the Los Angeles Department of Transporation indicates that the average signal cycle is 90-120 seconds in the city. We will take a value between these two averages, 100 seconds, and apply that to all intersections. For simplicity, we will add a wait time of 100 seconds for each left turn, 50 seconds for going straight, and 50 seconds for turning right.

Second, an "intersection" will be defined as a node in a graph, as that is where the edges, or "roads", connect. Unfortunately, there will be several nodes with degrees above and below 4. We will simplify this by generalizing the roads as either straight, left, or right based on the edge the driver is coming from.

Third, the map we will be using will be a smaller area focusing primarly on the center of the city, Los Angeles, using the same triangulation method as the earlier parts to generate the roads.

3.1 Create the maps

To create the map, we will select nodes with coordinates in the range (-118.309486, 34.090574):(-118.212078, 34.018242) and then generating a subgraph of $\tilde{G}\triangle$ from the December 2019 data. We then repeat this process for the data for March 2020.

We end up with two graphs consisting of 146 nodes and 393 edges.

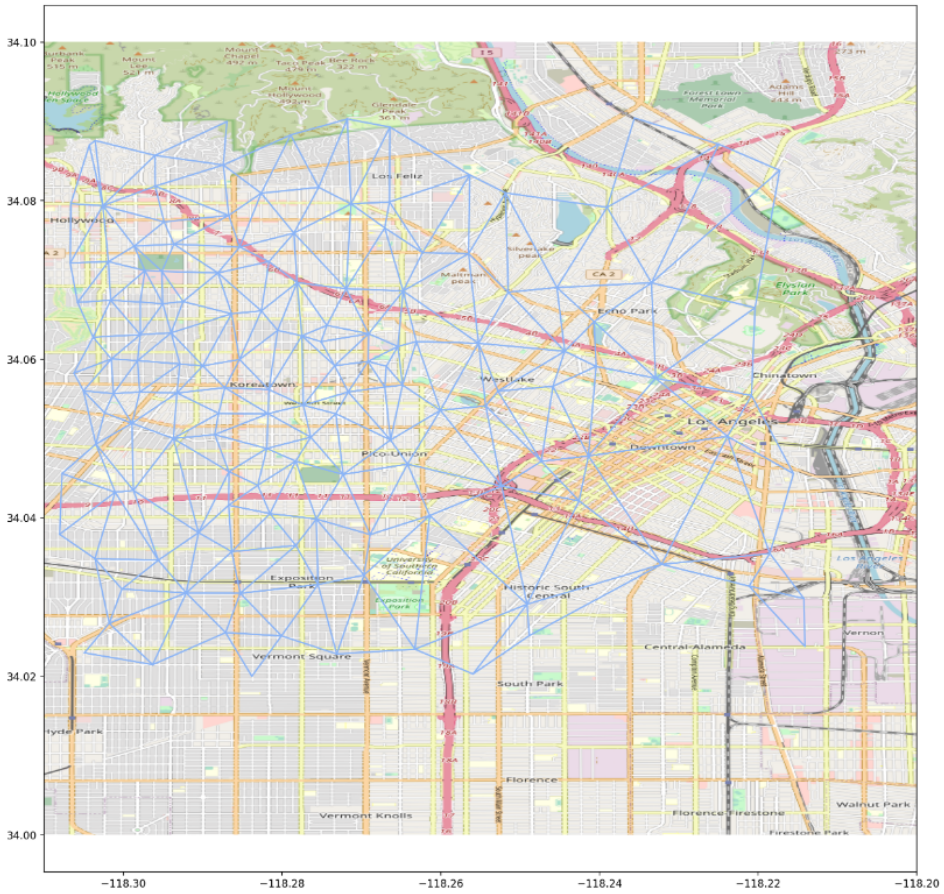


Figure 21: Map showing the nodes and roads for the reduced map of the December 2019 data

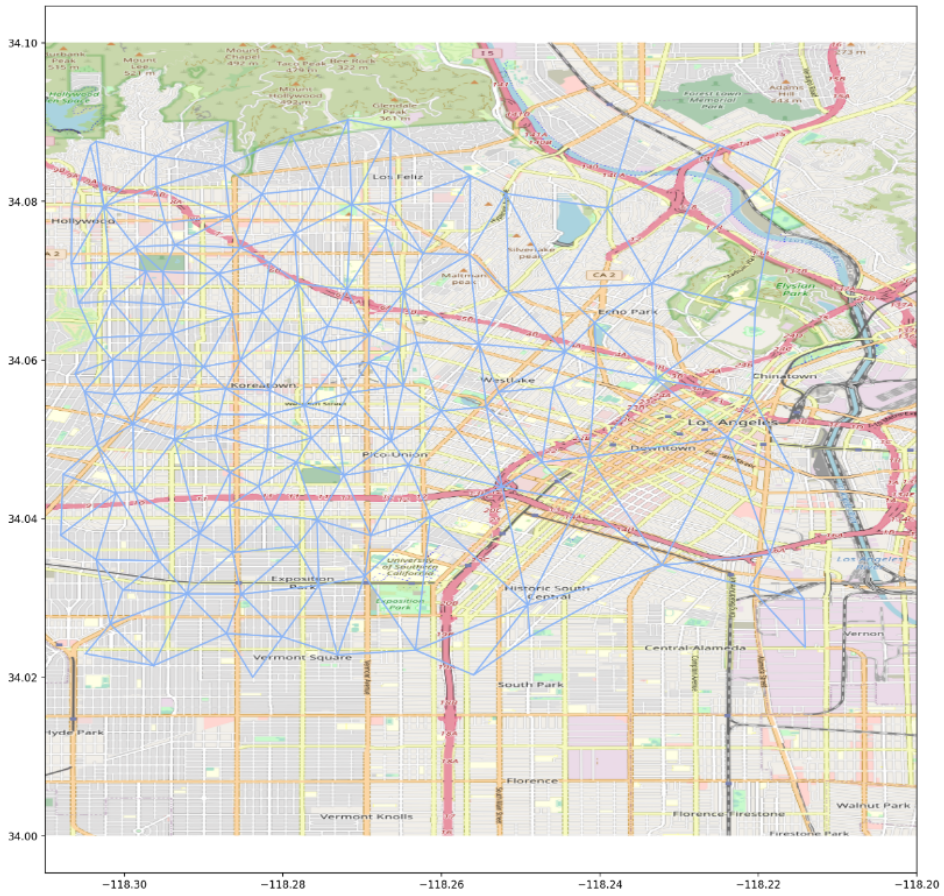


Figure 22: Map showing the nodes and roads for the reduced map of the March 2020 data

We can see from the above mappings that the two timeframes produce identital maps.

3.2 Determing turn direction

In order to keep track of whether a movement from one node to another is a left turn, right turn, or going straight, we will use the formula: $cos^{-1}(\frac{u \cdot v}{||u|| * ||v||})$ where u represents the vector for the edge the driver is coming from when reaching the intersection, and v is the vector representing the road the driving takes after the intersection. This will give us the degree between these two lines in a range between 0 and 180. We will say that if the degree is less than 135 degrees, then the truck has turned and if it is equal to or greater than 135, the truck is going straight. This is needed since the triangulation method generates several nodes with more and 4 neighbors and several with less than 4 neighbors, all at various angles.

We can then use a 2D cross product: $(v_2.x - v_1.x) * (v_3.y - v_1.y) - (v_2.y - v_1.y) * (v_3.x - v_1.x)$ where v_1 is the node prior to the current intersection, v_2 is the current intersection, and v_3 is the node the driver is going towards. If the resulting value is positive, then the outgoing vector is to the left relative to the incoming vector, and if the value is less than zero, then the outgoing vector is to the right.

When we combine the information from both of these values, we can determine if the truck has turned left, turned right, or went straight. U-turns are treated like left turns and receive the same time penalty.

3.3 Generating the delivery points

We will generate the nodes the truck must travel to by randomly sampling the common nodes of the two maps, the map from the December 2019 data and the map from the March 2020 data. We will call this set of nodes D. We then arbitrarily assign the first node in the randome sample as the start and end point for the delivery loop.

In this case, we are working with 10 nodes. [125, 112, 88, 18, 24, 144, 53, 41, 33, 27].

3.4 Algorithms

For this task, we will explore two different methods. Both will use a modified version of Dijkstra's algorithm but will vary in how the final path is generated. The modified Dijkstra's algorithm is nearly the same as the original except it takes into account the additional time penalty for turning when calculating the minimum path value. We then can generate the shortest paths for every point in D for each map from the two timeframes.

The first version will use a naive brute force method to generate an optimal path. We will take all of the nodes in D, except the start node, and generate alll possible permutations. We then use the shortest paths generated earlier to plot a full path for each permutation with turn penalties in mind. We then take the path with the smallest time and label that as the ideal path.

The second version will use a method similar to the first section of this project with the Approximate-1 method. Instead of making a multigraph of all the nodes and edges in each graph's MST, we instead only make a multigraph of the nodes in D. The edges that we add are representations of the shortest paths created by Dijkstra's algorithm. We then find an Euler circuit from this reduced graph and remap this to the original map with the full paths.

3.5 Results

As a quick note, the traditional Travelling Salesman Problem restricts the algorithm from repeating nodes. In this case, we will have the algorithm try to avoid repeating delivery nodes but will allow it to repeat intermediary nodes as it is not uncommon for a delivery truck to have to backtrack after delivering to a neighborhood.

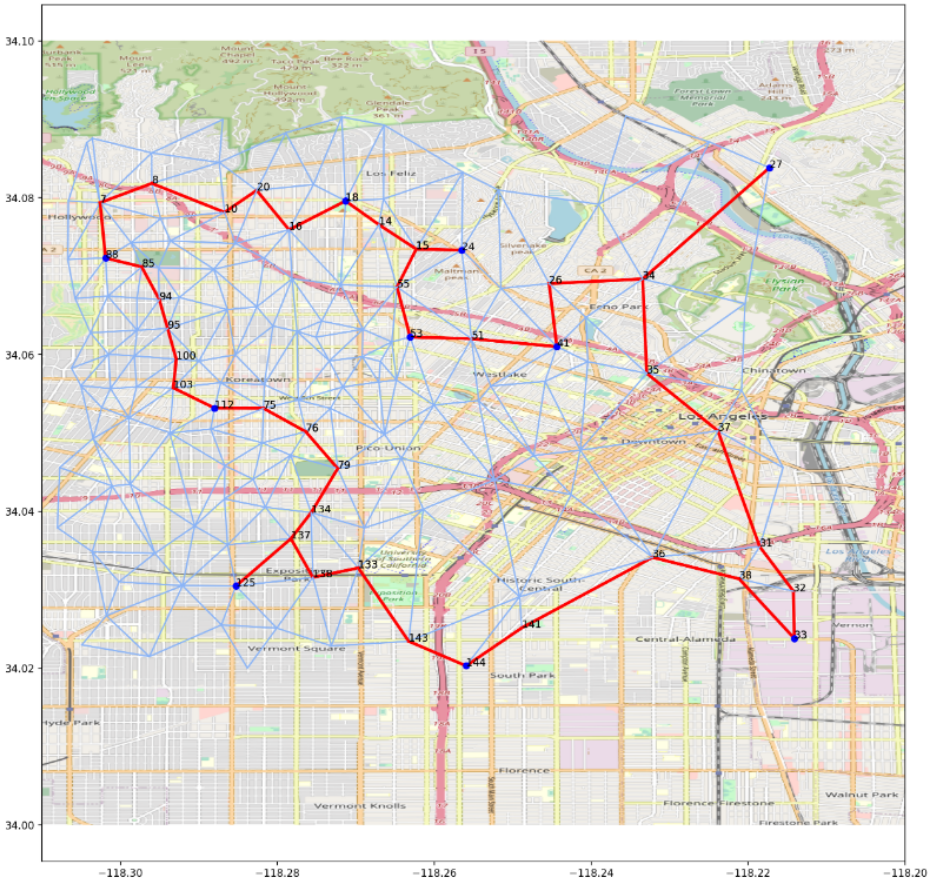


Figure 23: Resulting route from permutation method for both time frames

For the brute force permutation method, both time frames generated the same path when the turn penalties were included. The delivery node route in these cases is [125, 112, 88, 18, 24, 53, 41, 27, 33, 144, 125]. We can clearly see that there are three points where the driver makes a U-turn and goes back along an intermediary node to travel to the next node in the shortest time possible. As mentioned, we are allowing the truck to go back along nodes as it is still avoiding returning to a delivery location.

The pre-Covid route took 15240.32 seconds (254.01 minutes or 4 hours and 14.01 minutes). The Covid route took 14894.14 seconds (248.24 minutes or 4 hours and 6.24 minutes). These results indicate that despite the lockdowns, the reduction in traffic did not create too significant of a difference in travel times.

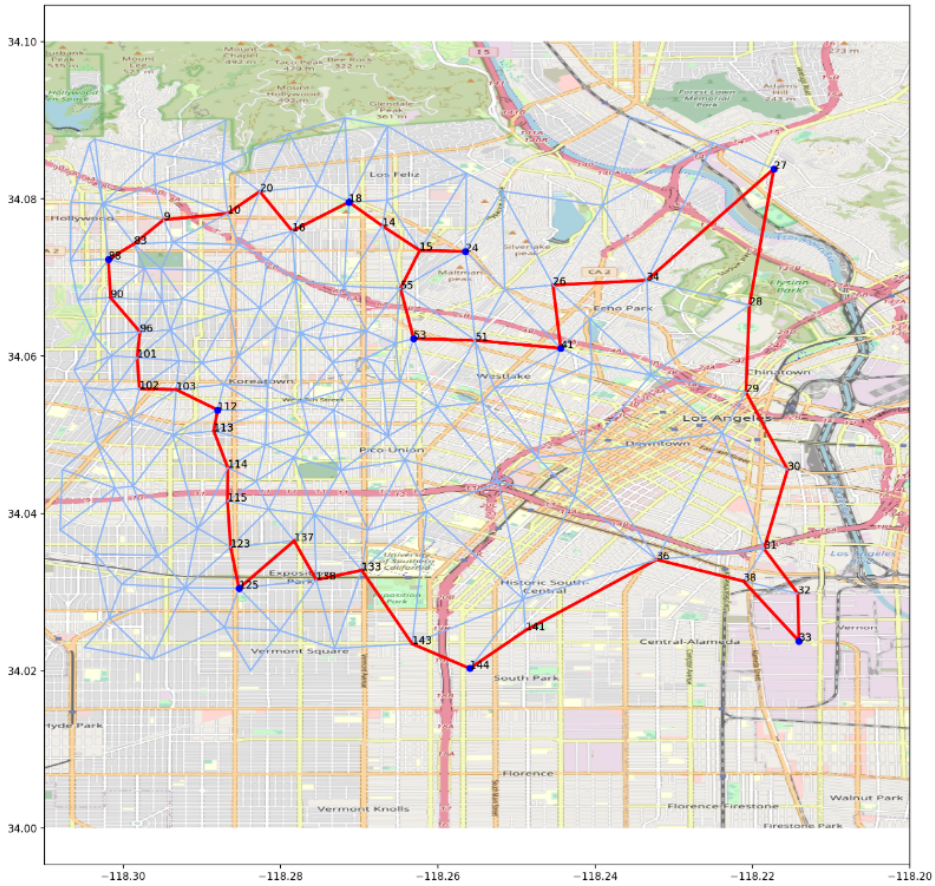


Figure 24: Resulting route from permutation method for both time frames without any turn penalties.

If we take a look at what route the algorithms would take without any turn penalties (e.g. the route a regular person might take), we get a path that looks much more similar to a traditional TSP solution with the exception of a single point where the truck doubles back along its route. The route in this case is [125, 112, 88, 18, 24, 53, 41, 27, 33, 144, 125] which is a slightly different route.

The pre-Covid route took 12116.72 seconds (201.95 minutes or 3 hours and 21.95 minutes) and the Covid route took 11790.72 seconds (196.512 minutes or 3 hours and 16.512 minutes). We still see a very small difference in total delivery time between the two time frames. If we add back in the minimum turn penalty for each intersection, we instead get the times 14266.72 seconds (237.78 minutes or 3 hours and 57.78 minutes) and 13940.72 seconds (232.345 minutes or 3 hours and 35.345 minutes) which is a fairer comparison between the two sets of times.

Next, we will discuss the results of the Eulerian circuit algorithm.

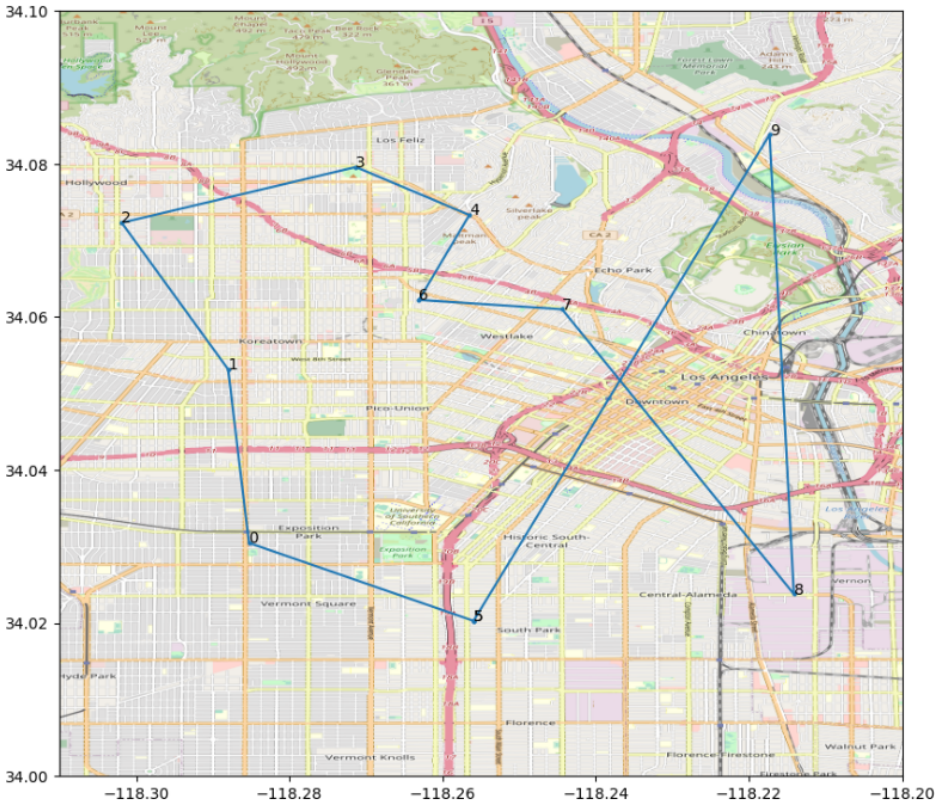


Figure 25: Reduced mapping of the eulerian circuit with just the delivery points for both time frames

Figure 25 shows a reduced version of the mapping of the truck's path. Each edge between each node represents the shortest path by Dijkstra's algorithm and shows the general intention of the algorithm. The intended path for this reduced mapping is [5, 9, 8, 7, 6, 4, 3, 2, 1, 0, 5]

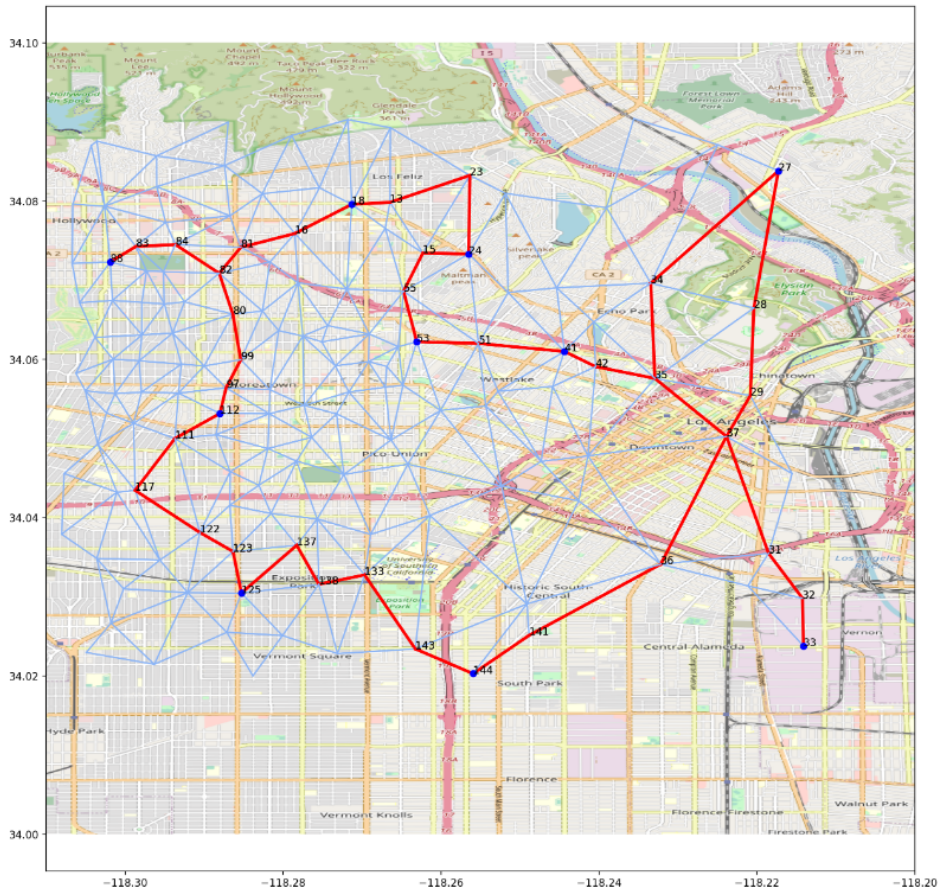


Figure 26: Resulting route from Euler circuit method for pre-Covid traffic

Figure 26 shows the full path that this algorithm tries to take. Again, we see that the truck makes two U-turns and travels back along intermediary nodes in an attempt to travel along the shortest path. The intended delivery path is [125, 144, 27, 33, 41, 53, 24, 18, 88, 112, 125]. This path takes 16691.524 seconds (278.192 minutes or 4 hours and 38.2 minutes) for this time frame.

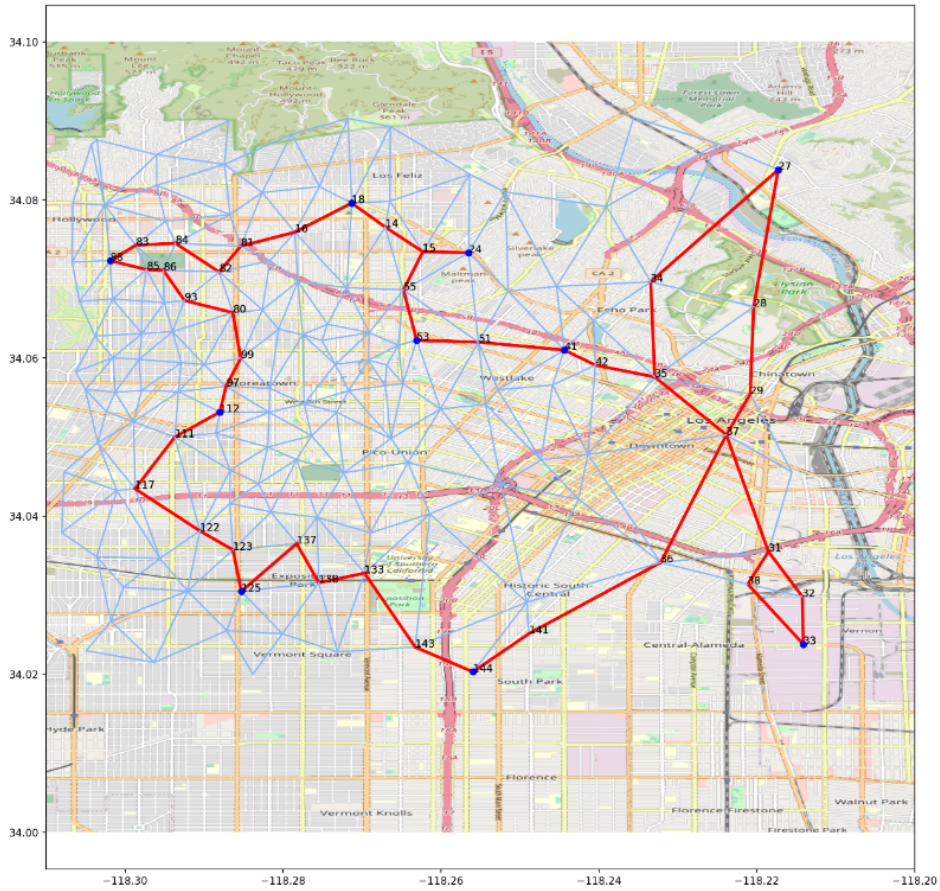


Figure 27: Resulting route from Euler circuit method for Covid levels of traffic.

Since the reduced mapping was the same for both time frames, the Covid route follows the same path of [125, 144, 27, 33, 41, 53, 24, 18, 88, 112, 125]. However, noteably, the truck only makes a single U-turn and instead opts to take a more roundabout path to avoid the turn penalty. Again, the path does visit intermediary nodes more than once. The total time for this path is 16159.665 seconds (269.33 minutes or 4 hours and 29.33 minutes)

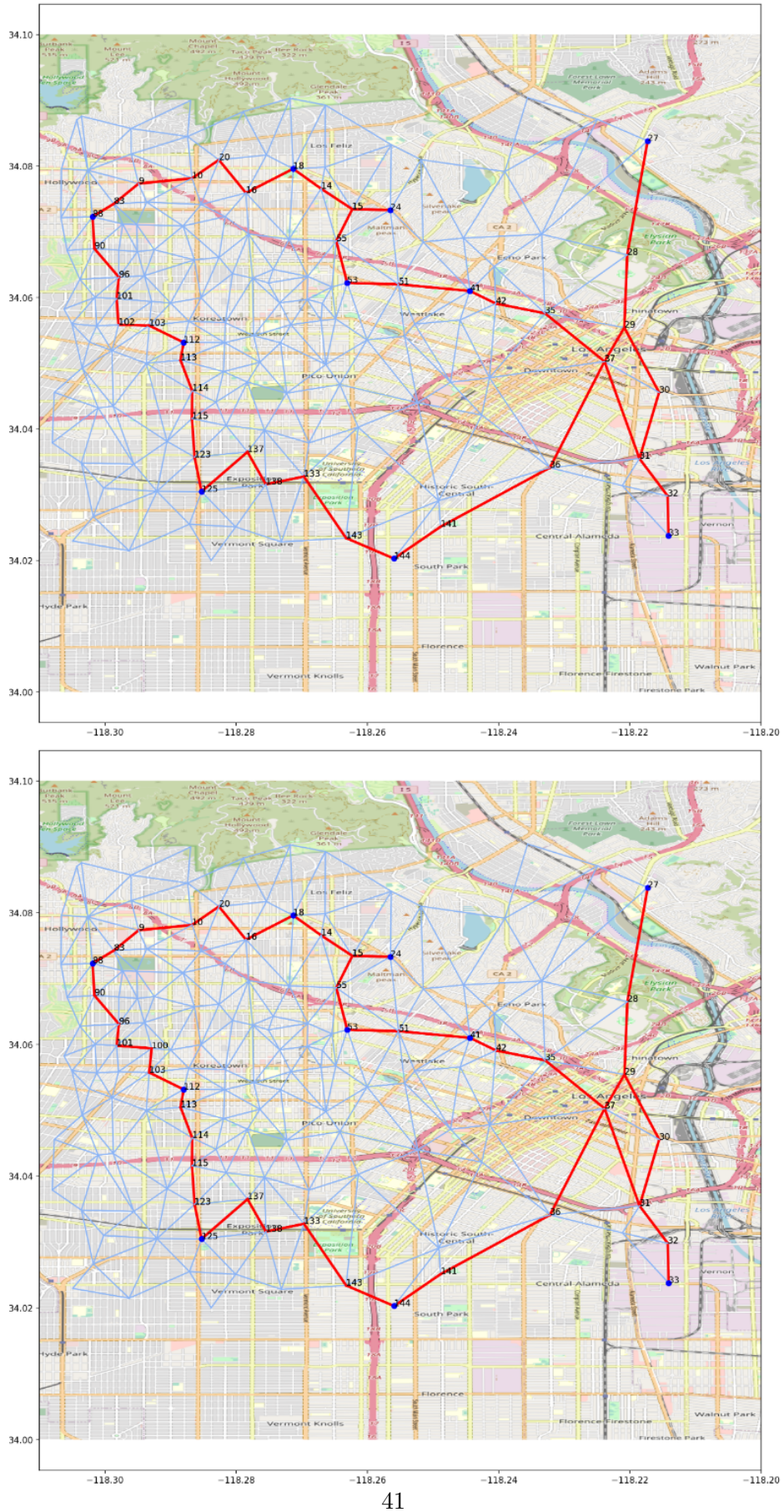


Figure 28: Pre-Covid and Covid delivery routes, respectively, without turn penalties

We again check to see what the path would be if there were no turn penalties for both time frames. The first mapping shows pre-Covid and the second shows during Covid lockdowns. The two mappings are nearly identical except for a small difference on the left hand side of the map where the pre-Covid travels the path [101, 102, 103] while the Covid path is [101, 100, 103]. The pre-Covid path takes 13515.835 seconds (225.26 minutes or 3 hours and 45.26 minutes) and the Covid path takes 13058.01 seconds (217.63 minutes or 3 hours and 37.63 minutes). Once the minimum turn penalty is added back in for each intersection, we get 15965.835 seconds (266. 097 minutes or 4 hours and 26.097 minutes) for pre-Covid and 15508.01 seconds (258.47 minutes or 4 hours and 18.47 minutes) for Covid.

Table 10: Path time for each time frame and algorithm.

Algorithm	**Pre-Covid**	**Covid**
Permutation w/ Turn Penalty	15240.32 seconds	14894.14 seconds
Permutation w/o Turn Penalty w/ Min Penalty	14266.72 seconds	13940.72 seconds
Euler Circuit w/ Turn Penalty	16691.524 seconds	16159.665 seconds
Euler Circuit w/o Turn Penalty w/ Min Penalty	15965.835 seconds	15508.01 seconds

We can see from Table 9 that the Covid times were consistently less than the pre-Covid times which was expected. We can also see that the permutation algorithm produced the best results in both time frames which is understandable as it loops through literally every possible delivery node order and then selects the smallest path. The Euler circuit method stilll produced good results, but it does not guarantee to find the best possible path. The major downside of the permutation method though is the time complexity. It is mostly driven by the number of permutations that must be tested which is why this task uses such a small delivery node size. If there are N nodes in D, then the algorithm must loop through $(N-1)!$ permutations since the start node is already determined. For smaller values of N, the algorithm is manageable and reasonable; however, as N gets larger, the time complexity rapidly increases. The general time complexity for the permutation algorithm is $O((N-1)!*V*N)$ where V is the number of nodes in the graph. The time complexity for the Euler circuit method is $O(V^2*E*N)$ where V is the number of nodes in the graph, E is the number of edges, and N is the number of delivery nodes. When examining the time complexities, we can see that when N is larger, the Euler circuit is better despite not always producing the optimal solution.

We can also conclude from Table 9 that, during Covid lockdowns, delivery trucks favoring right hand turns made their delivery route take only a few minutes longer than it would have taken them to travel the same delivery path pre-Covid without favoring right turns.