UCLA

232E - Large-Scale Social and Complex Networks: Design and Algorithms

Report on

Project 4: Graph Algorithms

Authors
Jayanth Shreekumar (805486993)
Leo Ly (805726182)
Yuheng He (505686149)

Spring 2022

Part 1 - STOCK MARKET

Introduction

In this part of the project, we used a dataset from the stock market to study correlation structures among fluctuation patterns of stock prices using tools from graph theory. We utilized time series data to construct different based on similarities among the returns that investors got for different time scales(day, week, and month). After construction these graphs, we studied their properties.

The correlation among log-normalized stock-return time series data is given by:

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

where
$\langle . \rangle$ is the temporal average on the given time regime,
$r_i(t)$ is the log-normalized return stock i over a period of $[t-1, t]$ given by

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

$q_i(t)$ is the return of stock i over a period of $[t-1, t]$ given by

$$q_i(t) = \frac{p_i(t) - p_i(t-1}{p_i(t-1}$$

and $P_i(t)$ is the closing price of stock i at the t^{th} day.

After computing the log-normalized return between each pair of stocks, we constructed an undirected correlation graph consisting of stocks as nodes and the edges as having weights taken from the formula:

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

Finally, we evaluate two clustering techniques with a higher value of alpha indicating a better clustering technique where:

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

The two clustering techniques are:

$$P(\nu_i \in S_i) = \frac{|Q_i|}{|N_i|} \text{ and } P(\nu_i \in S_i) = \frac{|Q_i|}{|V|}$$

where Q_i is the set of neighbours of node i that belong to the same sector as node i, N_i is the set of neighbours of node i, and V is the set of all nodes.

Question 1

We observed that the denominator for ρ_{ij} is the standard deviation formula and acts as a normalizing factor when calculating the log-normalized return. Therefore, ρ_{ij} is always a number between -1 and +1. When the number is positive, it indicates that the two stocks are positively correlated, and when it is negative, the two stocks are negatively correlated.

It is always a good idea to normalize when finding out correlation so that we reduce the effect of outliers in the data as well as take care of skewness and relative values between stocks rather than absolute values which can vary widely.

Question 2

As the log-normalized return ρ_{ij} is a value between [-1, +1], the weight of edges, that was calculated using:

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

has to be in the range [0, 2] where 0 indicates that the stocks had perfect positive correlation, and 2 indicates that the stocks had perfect negative correlation.

However, as we tried to calculate the edge weights, we realized that there were a few stocks that did not have data for all the time stamps. In fact, we noticed that 11 of them had incomplete data. Due to this, our log-normalized return matrix was incomplete and so, we removed the stocks that had incomplete data to get a complete matrix. Using this, we created the required correlation graph and computed the histogram of its edge weights that is displayed below:

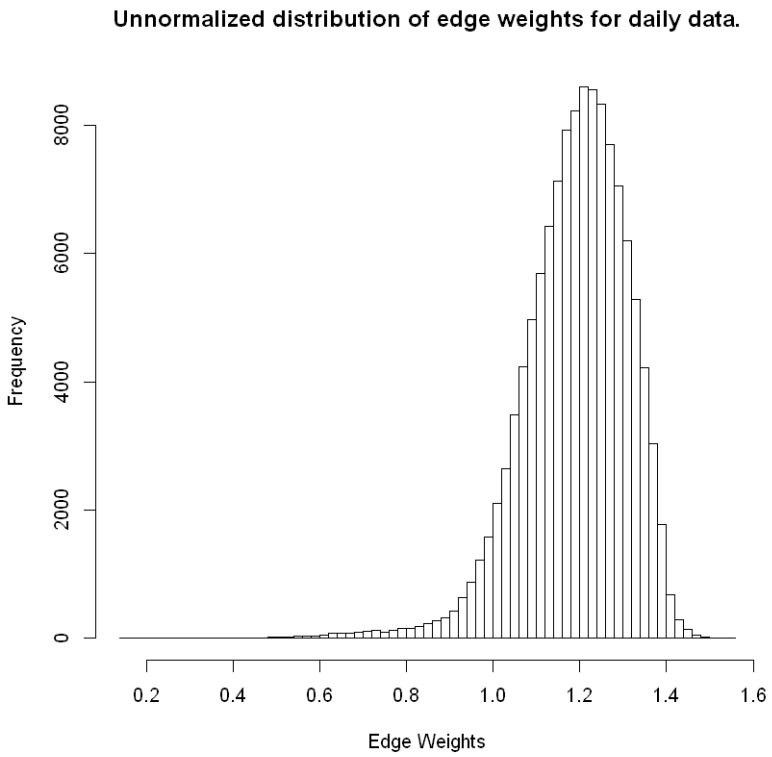


Figure 1: Histogram of edge weights for Daily Data

We observe that most of the edges have weights between 1 and 1.5, and almost all of the weights are between 0.5 and 1.5. This suggests there are no stocks that are strongly positively or negatively correlated. Also, when ρ_{ij} is 0, then it means there is no correlation between the two stocks, and this yields a value of 1.414 for the edge weight. Therefore, a value greater than 1.414 means that the two stocks are negatively correlated, and a value lesser than 1.414 means that the stocks are positively correlated. Thus, we conclude that most stocks are weakly positively correlated with each other in this dataset.

Question 3

After creating the correlation graph, we used the *mst* function in igraph to obtain the minimum spanning tree. A minimum spanning tree is a subset of the edges of a connected, weighted, undirected graph that connects all the nodes, without creating any cycles and with the minimum possible total cumulative edge weight.

As required, we color coded the nodes in the minimum spanning tree based on the sector the stock was associated with in the dataset. The MST graph is displayed below:

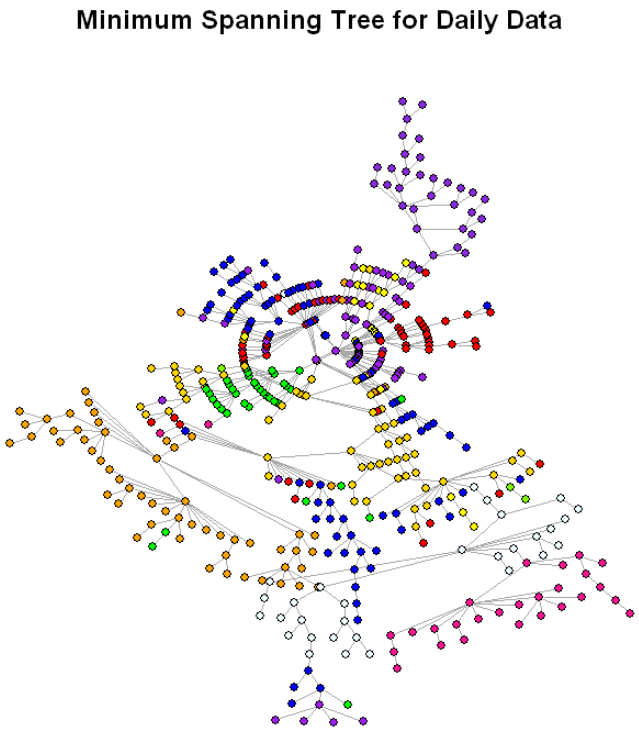


Figure 2: Minimum Spanning Tree for Daily Data

We clearly see that nodes (stocks) that belong to the same sector(same color) tend to form clusters. These clusters are called vine clusters. This suggests that stocks that are in the same sector are correlated to each other much more than to stocks belonging to other sectors. There are a few edges that connect nodes of different colors. We speculate that these stocks belong to sectors that

are not completely independent of each other. For example: Real Estate and Financials might have correlation depending on the company that the stock belongs to.

Another take-away from the MST is that as MST always strives to have a tree with the least possible total edge weight, and stocks that are in the same sector are connected, the edges between correlated stocks have a smaller weight than the edges between uncorrelated stocks.

Question 4

Graph nodes often form groups that are almost independent from the rest of the graph, with which they share a few edges, but the edges between nodes in that small group are denser. Such groups of nodes are called communities, and the community structure describes the communities existing in a given graph. As required, we ran the walktrap community detection algorithm on our MST using the *cluster_walktrap* function in igraph. The plot is displayed below:

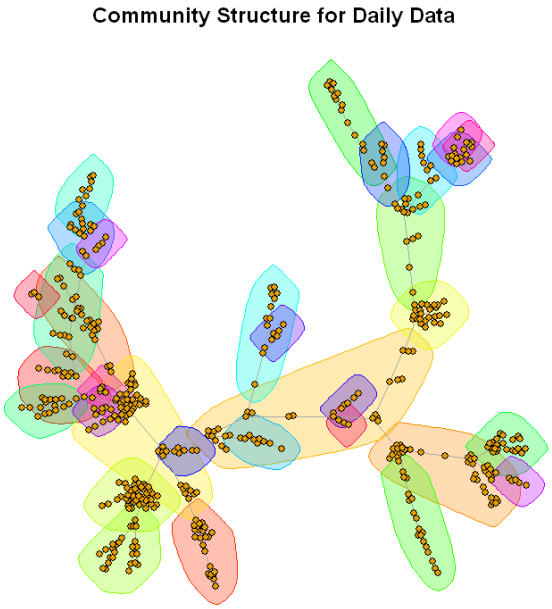


Figure 3: Community Structure for Daily Data using Walktrap Community Detection Algorithm

Homogeneity and completeness are measures to evaluate clustering techniques. Homogeneity evaluates if a community has users belong to the same circle. A perfect score of 1 means each community has its members share exactly one circle. A low score in homogeneity means each community has its members are equally distributed to all possible circles.

On the other hand, completeness metric considers each circle and evaluates the variation in community assignment. A perfect score of 1 in completeness means each given circle has its members share exactly one community.We used the clevr library in R to compute the homogeneity and completeness of this clustering and obtained the following results:

Homogeneity: 0.6826
Completeness: 0.4793

Question 5

As described in the Introduction, we calculated the value of alpha for two different sector clustering techniques and the results are as follows:

$P(\nu_i \in S_i) = \frac{|Q_i|}{|N_i|}$: $\alpha = 0.828930$

$P(\nu_i \in S_i) = \frac{|Q_i|}{|V|}$: $\alpha = 0.114188$

We see that the clustering technique that utilizes the neighbour information from the graph has a much higher value for α. This is as expected, as utilizing local information and connectivity in a graph to make decisions is much more intuitive and reasonable than simply utilizing only the global graph without any local information.

Question 6

In this part, we repeated the above procedure of graph analysis for the same dataset for weekly regimes. To do this, we extracted the subset of the data that was taken on Mondays and found that 13 stocks had incomplete data, and so we ignored those. After that, we performed the same procedure. The results are displayed below:

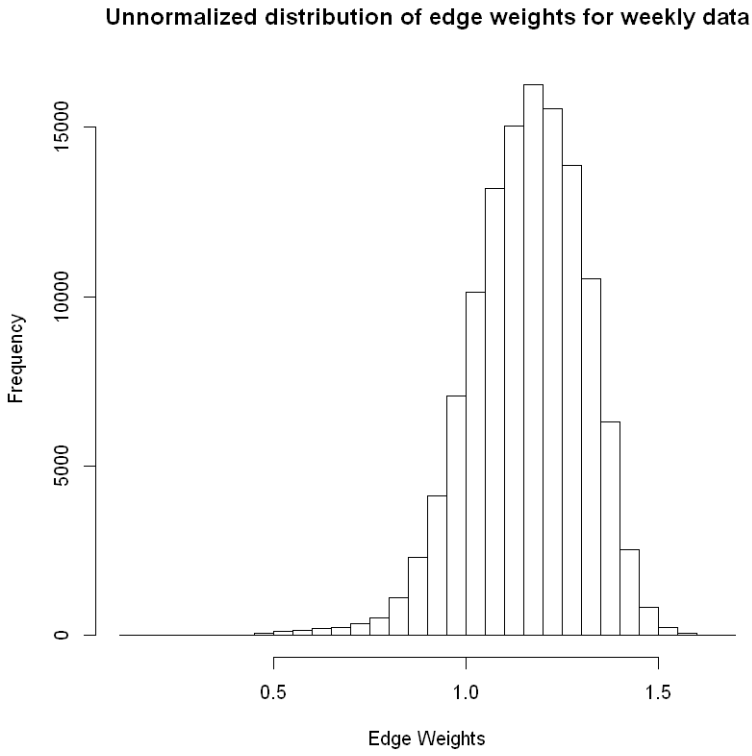


Figure 4: Histogram of edge weights for Weekly Data

Above is the histogram we obtained for edge weights for weekly data. We see that it is very similar to the histogram of weights for daily data, with a similar distribution between 0.5 and 1.5, and most of the stocks are still weakly positively correlated with each other.

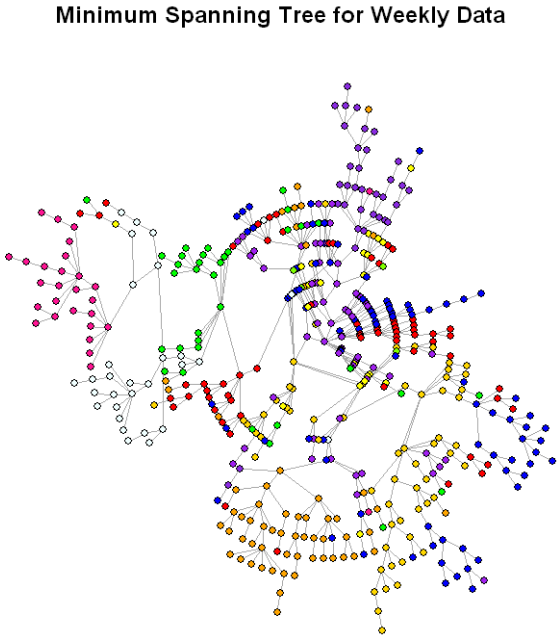


Figure 5: Minimum Spanning Tree for Weekly Data

Above is the MST that we obtained for weekly data. We observe that while there are still vine clusters that group nodes of the same sector together, there are a lot more anomalous nodes when compared to the MST of daily date. This indicates when the level of granularity decreases from daily to weekly data, clustering also decreases. This means the correlation between stocks decreases as well.

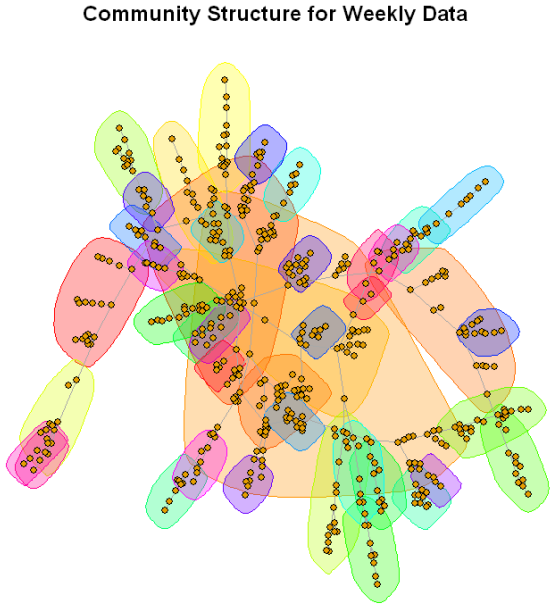


Figure 6: Community Structure for Weekly Data using Walktrap Community Detection Algorithm

The community structure of the weekly data obtained using the Walktrap community algorithm is shown above. We clearly see that there are a lot fewer communities in the weekly data. This

concurs with our analysis of the MST that the communities are not as tightly clustered, leading to uncertainty, and thus more number of communities. We recorded the homogeneity and completeness scores as follows:

Homogeneity: 0.5811
Completeness: 0.39004

Then,we calculated the value of alpha for two different sector clustering techniques and the results are as follows:

$P(\nu_i \in S_i) = \frac{|Q_i|}{|N_i|}$: $\alpha = 0.743957$

$P(\nu_i \in S_i) = \frac{|Q_i|}{|V|}$: $\alpha = 0.114309$

We see that the first value decreases when compared to that of daily data as expected. This indicates again that sector clustering has weakened. However, the second value is about the same, with a slight increase that is due to the number of nodes in the graph decreasing (as we ignored 13 for weekly data and 11 for daily data).

Question 7

In this part, we are asked to repeat the above procedure of graph analysis for the same dataset for monthly regimes. To do this, we extracted the subset of the data that was taken on the 15th of every month and found that 13 stocks had incomplete data, and so we ignored those. After that, we performed the same procedure. The results are displayed below:

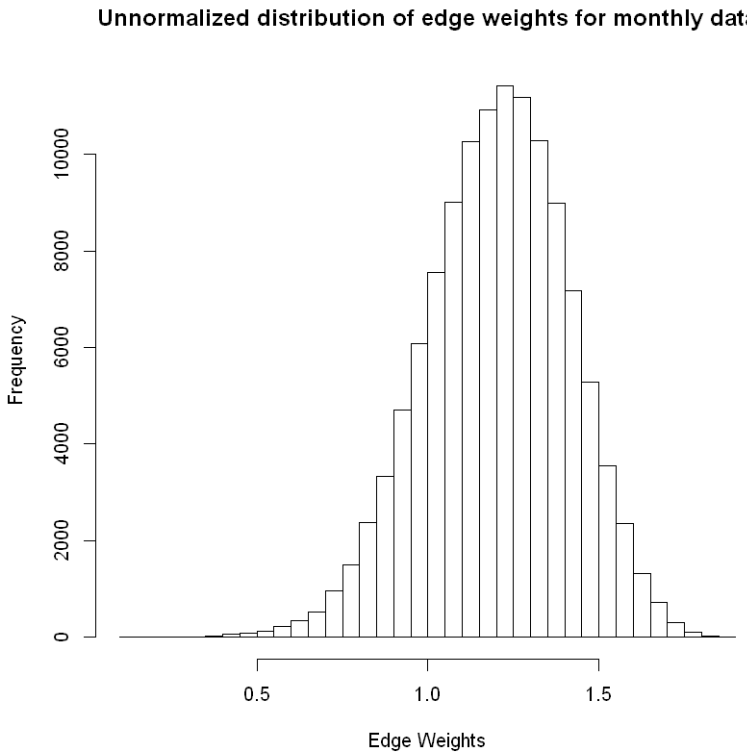


Figure 7: Histogram of edge weights for Monthly Data

Above is the histogram we obtained for edge weights for monthly data. We see that it is very

similar to the histogram of weights for daily data, with a similar distribution between 0.5 and 1.5, and most of the stocks are still weakly positively correlated with each other.

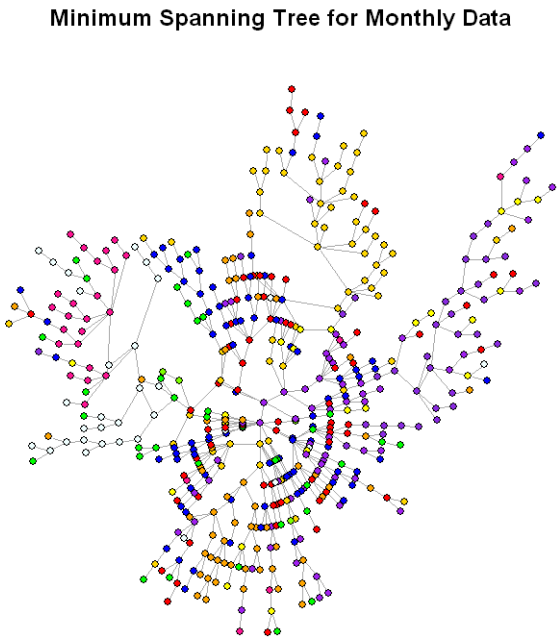


Figure 8: Minimum Spanning Tree for Monthly Data

Above is the MST that we obtained for Monthly data. We observe that while there are still vine clusters that group nodes of the same sector together, there are a lot more anomalous nodes when compared to the MST of daily data or weekly data. This indicates that when the level of granularity decreases from daily to weekly data and then to monthly data, clustering also decreases. This means that the correlation between stocks decreases as well.

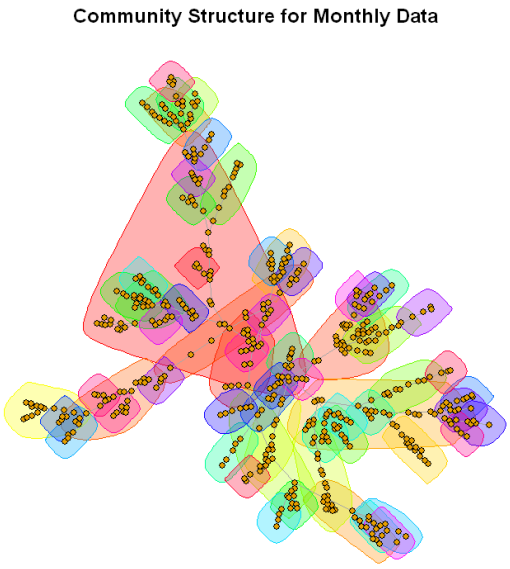


Figure 9: Community Structure for Monthly Data using Walktrap Community Detection Algorithm

The community structure of the weekly data obtained using the Walktrap community algorithm is shown above. We clearly see there are a lot fewer communities in the monthly data. This concurs with our analysis of the MST that the communities are not as tightly clustered, leading to uncertainty, and thus more number of communities. We recorded the homogeneity and completeness scores as follows:

Homogeneity: 0.47945
Completeness: 0.27755

Then, we calculated the value of alpha for two different sector clustering techniques and the results are as follows:

$P(\nu_i \in S_i) = \frac{|Q_i|}{|N_i|}$: $\alpha = 0.484446$

$P(\nu_i \in S_i) = \frac{|Q_i|}{|V|}$: $\alpha = 0.114309$

We see the first value decreases when compared to that of daily data as expected. This indicates again that sector clustering has weakened. However, the second value is about the same, with a slight increase that is due to the number of nodes in the graph decreasing (as we ignored 13 for monthly data and 11 for daily data).

Question 8

We make the following observations based on our results for comparison:

1. The histogram shape values remain similar across granularity.
2. As the granularity decreases from daily to weekly and finally to monthly data, we see that the MST becomes less and less clustered, and more anomalies start showing up. This indicates that the communities are becoming smaller and more uncertain. Thus, we are losing correlation between stocks in monthly data, that was present in daily data.
3. In the community structure graphs, we clearly see an increase in the number of communities, which again indicates that we are losing correlation between stocks.
4. As we go from daily to weekly to monthly data, we see that both homogeneity and completeness values decreases, indicating that the communities are becoming sparser and weaker.
5. As we go from daily to weekly to monthly data, we see that the value of alpha1 also decreases, again indicating a decrease in clustering certainty and strength.
6. As we go from daily to weekly to monthly data, we see that the value of alpha2 remains constant, indicating that it is not a reliable method for clustering and prediction.

Therefore, we conclude that using the granularity level of daily data provides the best results when predicting the sector of an unknown stock as the clustering score of alpha1 is much higher for daily data, which gives more certainty for your prediction.

Part 2 - UBER DATASET

Question 9

In this part, we built graph with the Uber data where nodes relate to locations and the weights f edges relate to the mean traveling times between each pair of nodes. We only used the December data, and the following properties are observed.

- Nodes: 2649
- Edges: 1004955

Question 10

In this part, we built a minimum spanning tree (MST) of graph G and the results are presented in Fig. 10. We also obtained the street address of a few edges and the results are given in Table 1. It is observed from the table that the endpoints are close to each other and this is intuitive in that MST constructs the tree where two nodes has the lowest mean travel time and the way to do it is to pair the nodes close to each other together, resulting in a MST.

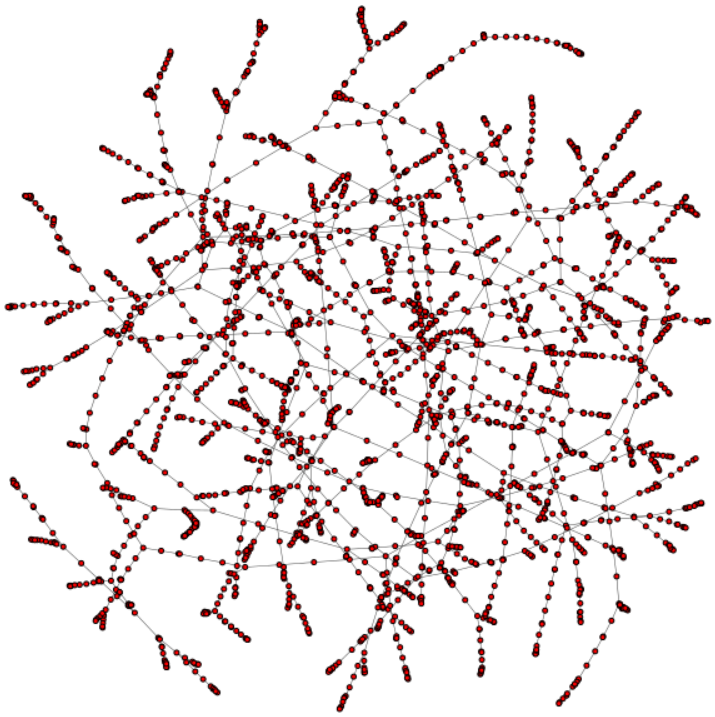


Figure 10: Minimum Spanning Tree of Graph G

Endpoint 1			Endpoint 2		
Census Tract	**Coordinates**	**Address**	**Census Tract**	**Coordinates**	**Address**
55400	[-118.133298 33.904119]	Rosecrans Avenue, Bellflower, CA 90706	554002	[-118.141448 33.896526]	Rancho Tire, Somerset Boulevard, Bellflower, CA 90706
461700	[-118.159325 34.153604]	500 Prospect Boulevard, Pasadena, CA 91103	460800	[-118.172425 34.180286]	Brookside Golf Club, Rosemont Avenue, Pasadena, CA 91109
302201	[-118.251349 34.146334]	134 North Kenwood Street, Glendale, CA 91206	302202	[-118.248811 34.142665]	East Colorado Street, Glendale, CA 91205
407101	[-117.967696 34.04101]	619 North Sunset Avenue, La Puente, CA 91744	407002	[-117.980572 34.051745]	652 Puente Avenue, El Monte, CA 91746
433401	[-118.043316 34.058606]	10468 Fern Street, South El Monte, CA 91733	433402	[-118.038157 34.056957]	2433 Continental Avenue, El Monte, CA 91733
543603	[-118.28871 33.803659]	Harbor Freeway, West Carson, CA 90710	294410	[-118.289861 33.797764]	828 Lomita Boulevard, West Carson, CA 90710

Table 1: Address of a few edges

Question 11

By randomly sampling 1000 triangles, we obtained a result of 90.909% of the triangles that satisfies the triangle inequality.

Question 12

To find the upper bound of the empirical performance, we first calculated t$\rho = 1.5663852972306513$. We know that the following holds.

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

To approximate the optimal TSP cost, we can use the MST cost.

$$\rho = \frac{\text{Approximate TSP Cost}}{\text{MST cost}}$$

Theoretically, we have MST cost $\leq$ Optimal TSP cost $\leq$ Approximate TSP cost $\leq 2 \times$ MST cost. Therefore, we have:

$$1 \leq \frac{\text{Optimal TSP cost}}{\text{MST cost}} \leq \rho(1.5663852972306513) \leq 2$$

We obtained that the upper bound is 2.

Question 13

In this part, we obtained the route that Santa has to travel and the trajectory results are presented in Fig. 11.

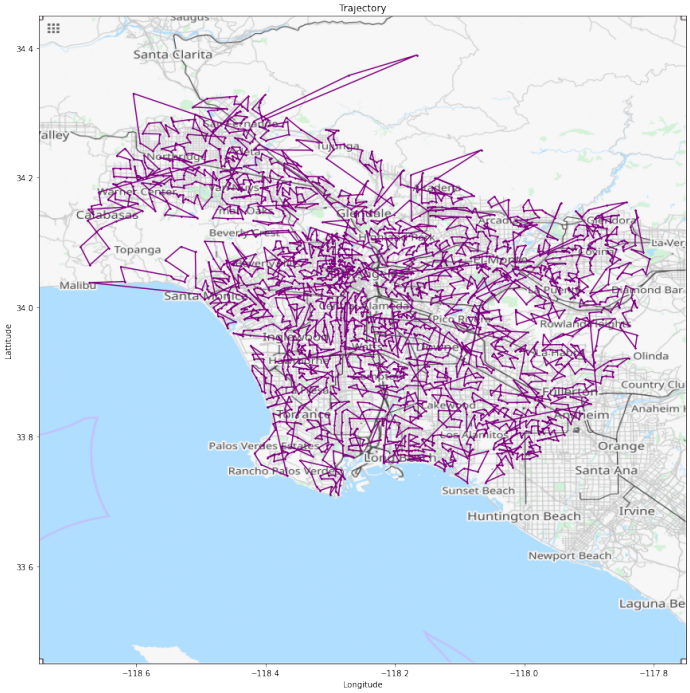


Figure 11: Santa's Travel Trajectory

We could see in Fig. 11 that there are not many overlapping routes which is intuitive because it minimizes the distance Santa has to travel while fulfilling Santa's visit requirements.

Question 14

In this part, we estimated the road map without the actual road dataset using the Delaunay triangulation algorithm. The result is The graph G_Δ, presented in Fig. 12. The graph G_Δ where nodes are different locations and edges are produced by triangulation is shown in Fig. 13.

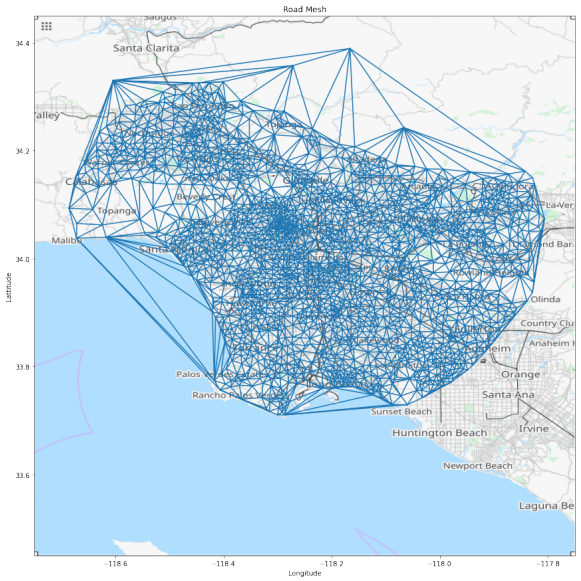


Figure 12: Road Mesh of LA

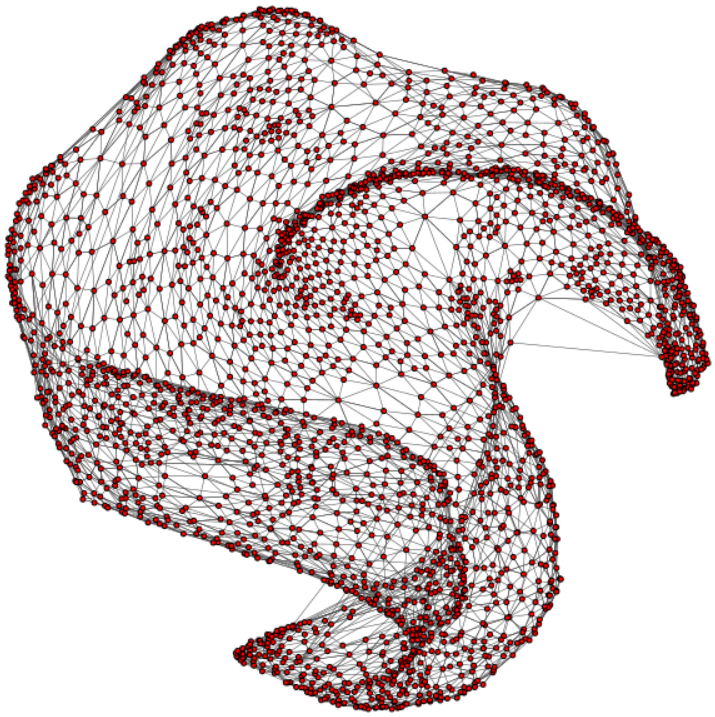


Figure 13: Graph G_Δ

We can observed in Fig. 12 that all the triangles fit in an outer circle and the center is more dense compared with the peripheral area. Nevertheless, there are some impossible routes such as the ones crossed the ocean and those over the mountains. this is explained by the definition of the Delaunay triangulation algorithm, which tries to fit all the triangles into the circle without producing narrow triangles.

Question 15

The velocity v of the car is defined as $v = \frac{d}{t}$ where d is the distance and t is the mean travel time.

The safety distance is $\frac{2\times v}{3600} = \frac{v}{1800}$ mile.

Cars remained in the road equal $\frac{d}{0.003+\frac{v}{1800}}$.

Max Traffic Flow (two lanes) is $\frac{2\times \frac{d}{t}}{0.003+\frac{v}{1800}} = \frac{2}{\frac{0.003}{v}+\frac{1}{1800}}$ cars/hr

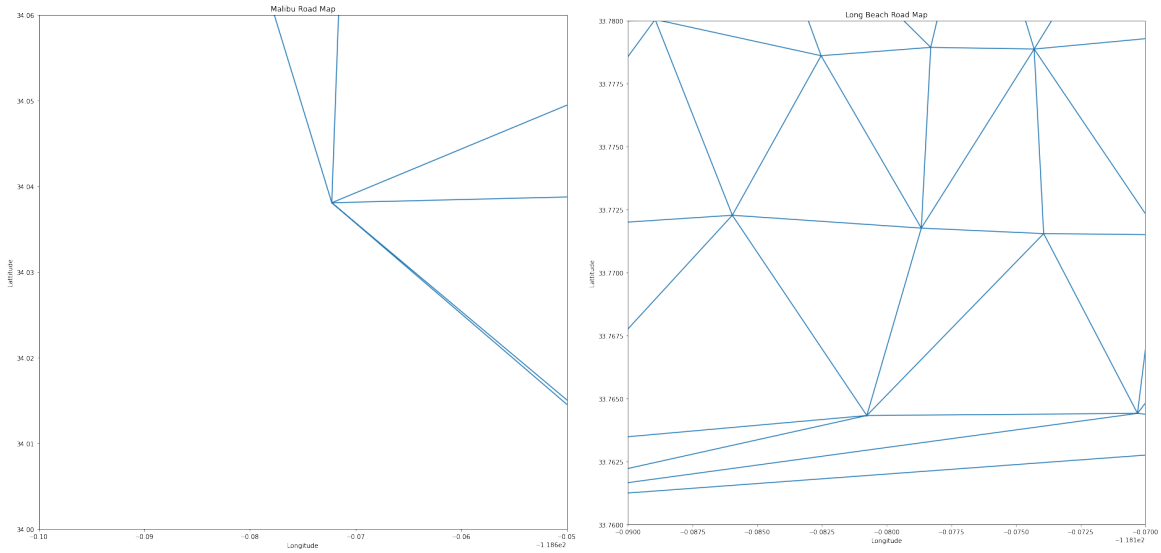
Question 16

We first found the corresponding node for Malibu and Long Beach since the coordinates provided may not exactly match with the actual coordinates in the dataset. The node we found are **Malibu: 1699** and **Long Beach: 660**. The max flow of car from Malibu to Long Beach is reported in Table 2.

Max Flow	**edge-disjoint paths**
14497.24	6

Table 2: Max Flow from Malibu to Long Beach and Corresponding edge-disjoint Paths

We also zoomed in to the road map of Malibu and Long Beach to take a closer look of the paths. The results are recorded in Fig. 14.



(a) Malibu Road Map

(b) Long Beach Road Map

Figure 14: Road Map of Malibu and Long Beach

It is observed that Malibu has 6 outgoing paths and Long Beach has 6 incoming paths, which match the edge dis-joint paths we found.

Question 17

In this part, we pruned the graph G_Δ by applying threshold to the traveling time to remove some impossible paths. In our case, we set the threshold to 11 minutes, which is 660 seconds. We only maintain paths that require less than 660 seconds traveling time and pruned those that exceed the limit. The resulting pruned graph $\tilde{G}_\Delta$ is presented in Fig. 15.

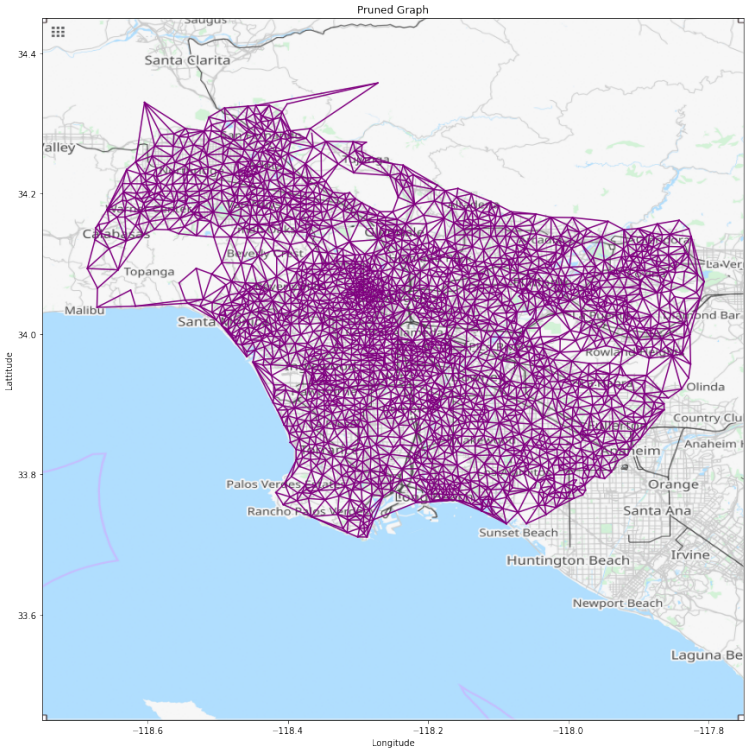


Figure 15: Pruned Road Mesh of LA using Time Threshold

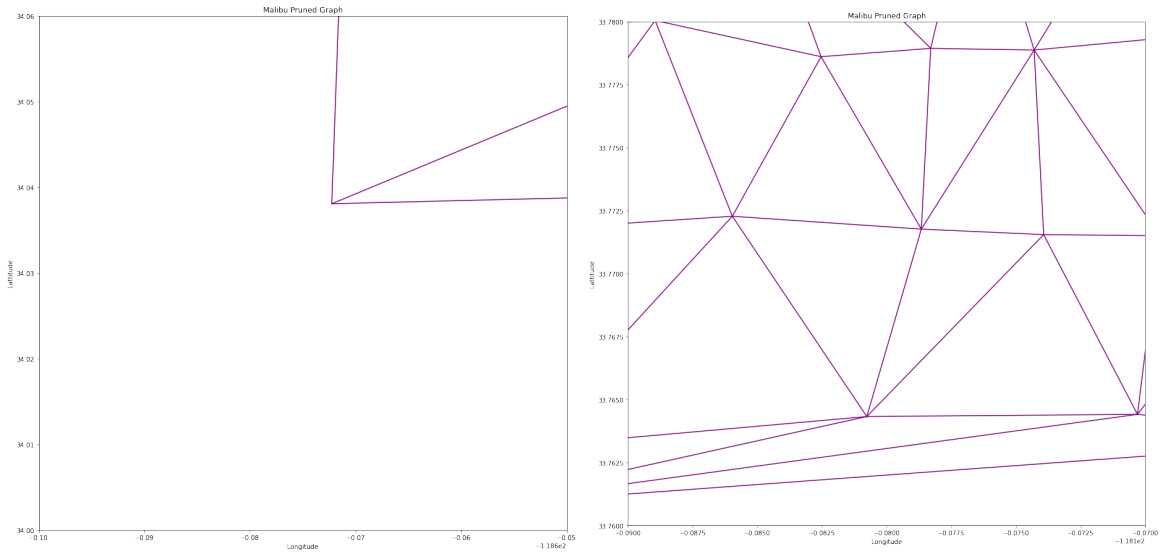
We can obtained from the graph that most of the impossible paths have been removed, especially those crossing the ocean and going over the mountains. The pruned graph makes better sense in that it maintains the good routes while eliminates the bad ones. This result indicates that our threshold method works.

Question 18

After pruning the graph, we obtained a reduced road map result, and we obtained the max car flow for the pruned graph. This calculation is reported in Table 3.

Max Flow	**edge-disjoint paths**
13755.92	3

Table 3: Pruned Max Flow from Malibu to Long Beach and Corresponding edge-disjoint Paths



(a) Malibu Pruned Road Map

(b) Long Beach Pruned Road Map

Figure 16: Pruned Road Map of Malibu and Long Beach

We noticed that the max flow is reduced because some paths are pruned for exceeding the time threshold. Additionally, the edge-disjoint paths is reduced to 3, and this matches the graph we see on the Malibu and Long Beach road.

Constructing New Roads

Using the pruned graph without the unreal links along the concavities of the beach generated in Question 17, let us assume it is the actual road network of LA. The government of LA wants to construct 20 new roads. We need to decide where to construct the roads (edges). We will examine 6 strategies to construct new roads.

Question 19 - Strategy 1

In the first strategy, we reduced the maximum extra traveling distance by constructing 20 new roads all at once. The extra travelling distance between 2 locations is the difference of the shortest traveling distance and the straight line distance, i.e.

extra_distance(v,s) = distance_of_shortest_path(v,s) - euclidean_distance(v,s)

As distance of shortest paths is not a straight line, it is always larger than the euclidean distance. Extra distances are calculated for all pairs of points. The top 20 pairs with highest extra distance are chosen as the new roads to be built. Table 4 shows the list of edges added in strategy 1. New edges are graphed in red in Figure 17.

The 20 new edges are able to connect pairs of points with the longest distances. Some new edges are built to cross the concavity of the beach. These new edges, however, are still concentrated. The advantage of this simple strategy is time-efficient – it only took 4.48 seconds to compute the shortest paths, find the extra distances, and add new edges. Finding the extra distances and adding new edges took 1.26 seconds. We used Dijkstra algorithm to find the shortest paths. Its time complexity is $O(E + Vlog(V))$, with V is the number of vertices and E is the number of edges in the pruned graph. The first strategy also needs to sort the *extra_distance*. With the sorting cost is $O(nlog(n))$ and $n = V^2$, the time complexity for strategy 1 is $O(E + VlogV + V^2log(V^2))$.

Added Edge
[1783, 2416]
[2140, 2419]
[1860, 2416]
[382, 2419]
[383, 2419]
[1699, 1783]
[391, 2419]
[1901, 2419]
[381, 2419]
[1699, 1860]
[2163, 2419]
[386, 2419]
[390, 2419]
[1904, 2419]
[45, 2419]
[1783, 2413]
[1906, 2419]
[2158, 2419]
[1902, 2419]
[392, 2419]

Table 4: List of added edges in strategy 1

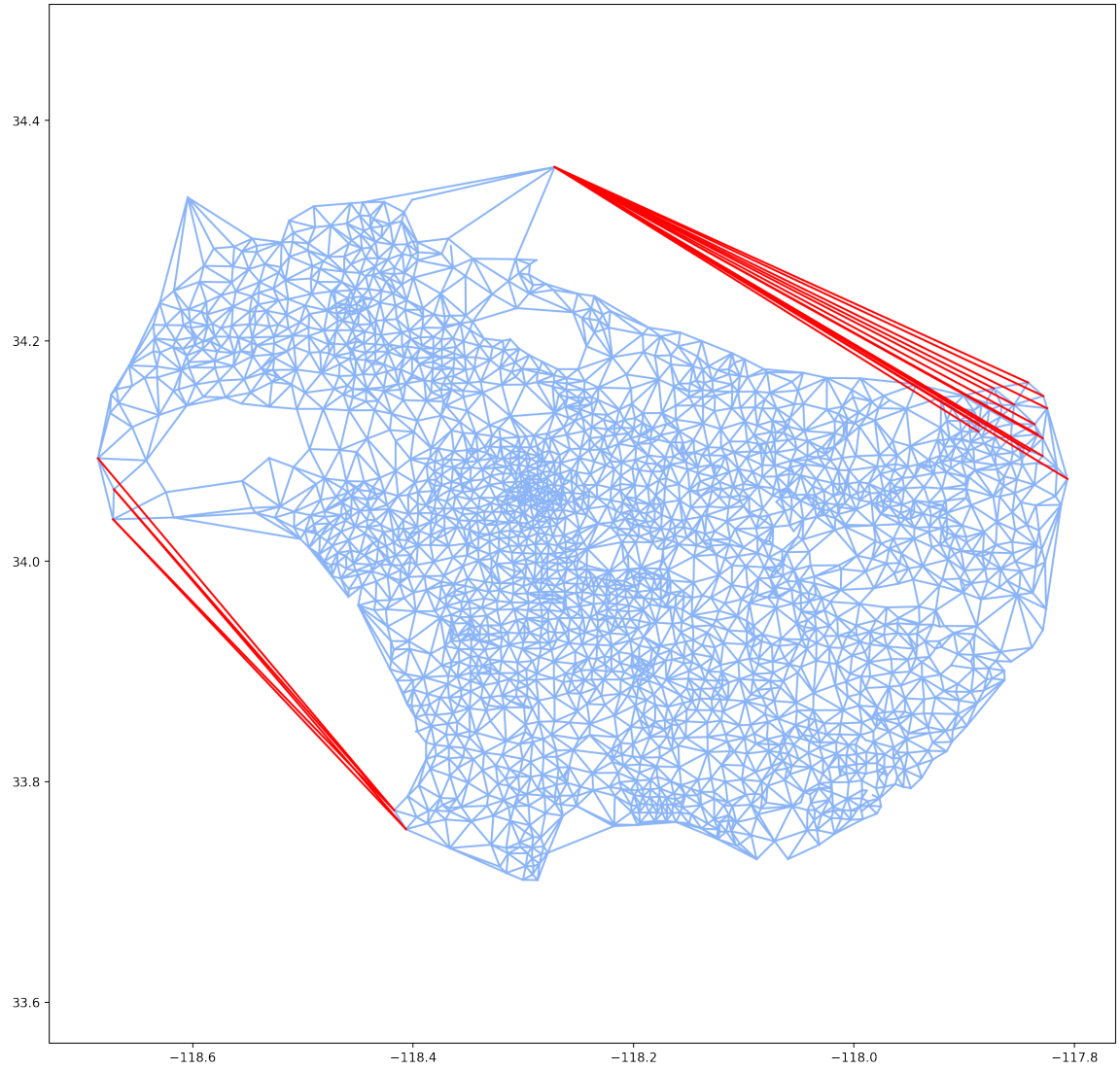


Figure 17: Strategy 1 - Static edges by distance

Question 20 - Strategy 2

The second strategy is similar to the first, with the only difference is the random frequency applied to every pair of points. In real world, some pairs are more in demand than other pairs. Assume that the frequency of travel between between each pair is a random integer between [1,1000], we obtain the revised *extra_distance* as follow:

$$weighted\_extra\_distance(v, s) = extra\_distance(v, s) - frequency(v, s)$$

The top 20 pairs with highest *weighted_extra_distance* are the new edges we suggest (see Table 5). New edges are graphed in red in Figure 18.

In terms of time complexity, there is virtually no change comparing to strategy 1. The time it took for the strategy 2 to compute the *weighted_extra_distance*, sort the distance, and add the 20 edges to the graph was only 1.43 seconds. The only difference is the multiplication of *extra_distance* and $frequency$, which is handled efficiently by Numpy library. The time complexity is still $O(E + VlogV + V^2log(V^2))$.

Added Edge
[1783, 2413]
[389, 2419]
[2163, 2419]
[989, 1510]
[49, 2419]
[383, 2419]
[1906, 2419]
[257, 2419]
[66, 2419]
[2141, 2419]
[388, 2419]
[2068, 2419]
[69, 2419]
[252, 2419]
[2072, 2419]
[1005, 1510]
[2419, 2464]
[2161, 2419]
[430, 1783]
[2419, 2575]

Table 5: List of edges added in strategy 2

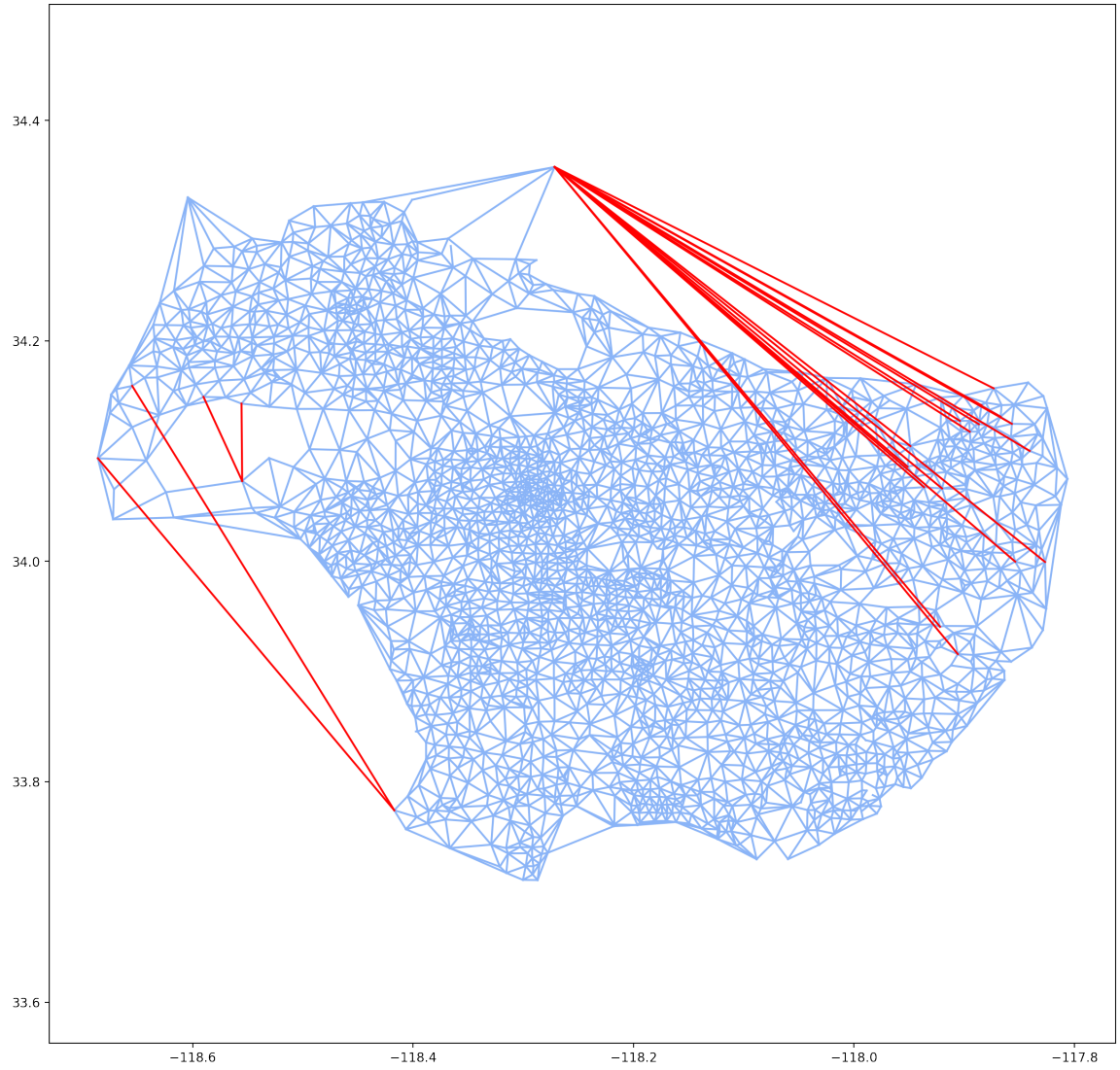


Figure 18: Strategy 2 - Static edges by weighted distance

Question 21 - Strategy 3

In strategy 1 and strategy 2, we created all roads at the same. In strategy 3, we create the roads one by one. The algorithm is as the following:

1. Compute *extra_distance* between all the pairs in the graph.
2. Create a road between the pair with highest *extra_distance* and update the graph.
3. Repeat Steps 1 and 2 19 more times to find 20 new edges in total.

Table 6 shows the list of edges created in Strategy 3. Figure 19 shows the new edges created on actual coordinates.

Intuitively, strategy 3 should be 20 times slower than strategy 1 because the shortest path algorithm is called for each iteration. The time it took to run on Colab for this strategy is 43.45 seconds. The time complexity is $O(20E + 20VlogV + 20V^2log(V^2))$.

Added Edge
[1783, 2416]
[2140, 2419]
[986, 1510]
[49, 2419]
[285, 2419]
[1717, 2419]
[1783, 2417]
[1956, 2419]
[2247, 2419]
[1005, 1678]
[2414, 2619]
[121, 689]
[2242, 2419]
[1699, 1781]
[144, 2619]
[2402, 2416]
[144, 2040]
[1700, 1782]
[356, 1679]
[2414, 2559]

Table 6: List of edges added in strategy 3

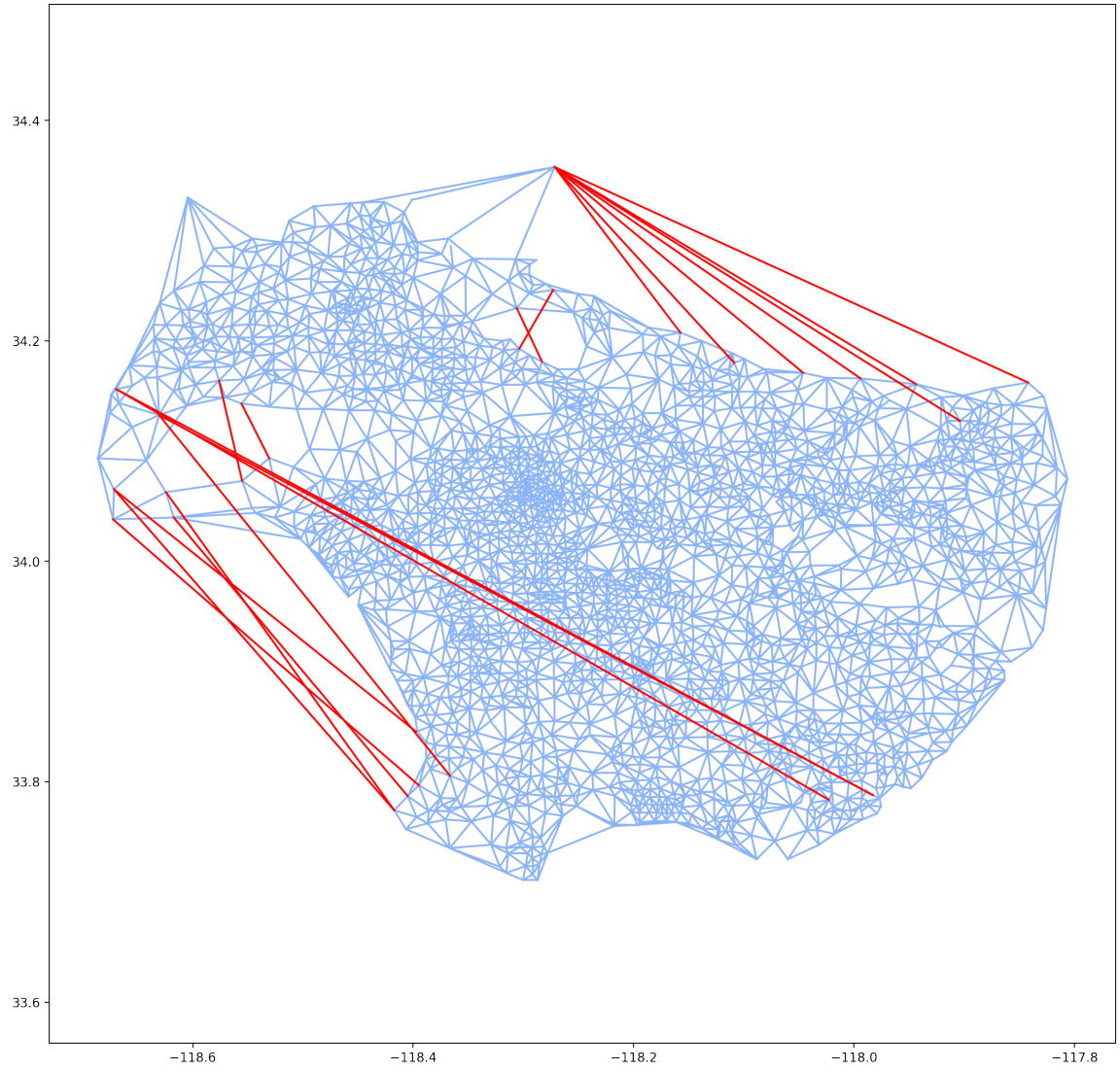


Figure 19: Strategy 3 - Dynamic edges by distance

Strategy 4
Instead of minimizing the travel distance, this strategy builds edges that optimize the traveling time. The extra traveling time between 2 locations is the difference of the shortest traveling time and the straight line travel time, i.e.

$$extra\_time(v,s) = travel\_time\_of\_shortest\_path(v,s) - euclidean\_distance(v,s)/travel\_speed(v,s)$$

$$travel\_speed(v,s) = distance\_of\_shortest\_path(v,s)/travel\_time\_of\_shortest\_path(v,s)$$

Based on the given travel time between edges given by Uber, we are able to calculate the $travel\_time\_of\_shortest\_path$ using Dijkstra shortest path algorithm. By using the $distance\_of\_shortest\_path$ already calculated in strategy 1, we obtain $travel\_speed$. Assume that the $travel\_speed$ of the shortest path is similar to that of the straight path, we can find the travel time of the straight path, which is $euclidean\_distance(v,s)/travel\_speed(v,s)$. Table 7 shows the list of edges created in strategy 4. Figure 20 shows the new edges created on actual coordinates.

Added Edge
[1783, 2416]
[1860, 2416]
[1783, 2413]
[1859, 2416]
[1860, 2413]
[1699, 1783]
[1699, 1860]
[985, 1678]
[988, 1510]
[1782, 2416]
[989, 1510]
[144, 1783]
[144, 1860]
[430, 1783]
[950, 1510]
[989, 1678]
[1882, 2416]
[430, 1860]
[951, 1510]
[984, 1678]

Table 7: List of edges added in Strategy 4

The time complexity of strategy 4 is similar to strategy 1 - we need to find the travel time shortest paths instead of the distance shortest paths. It took 3.66 seconds on Colab to compute shortest travel time path, sort extra time, and create new edges. The time complexity is $O(E+VlogV+V^2log(V^2))$.

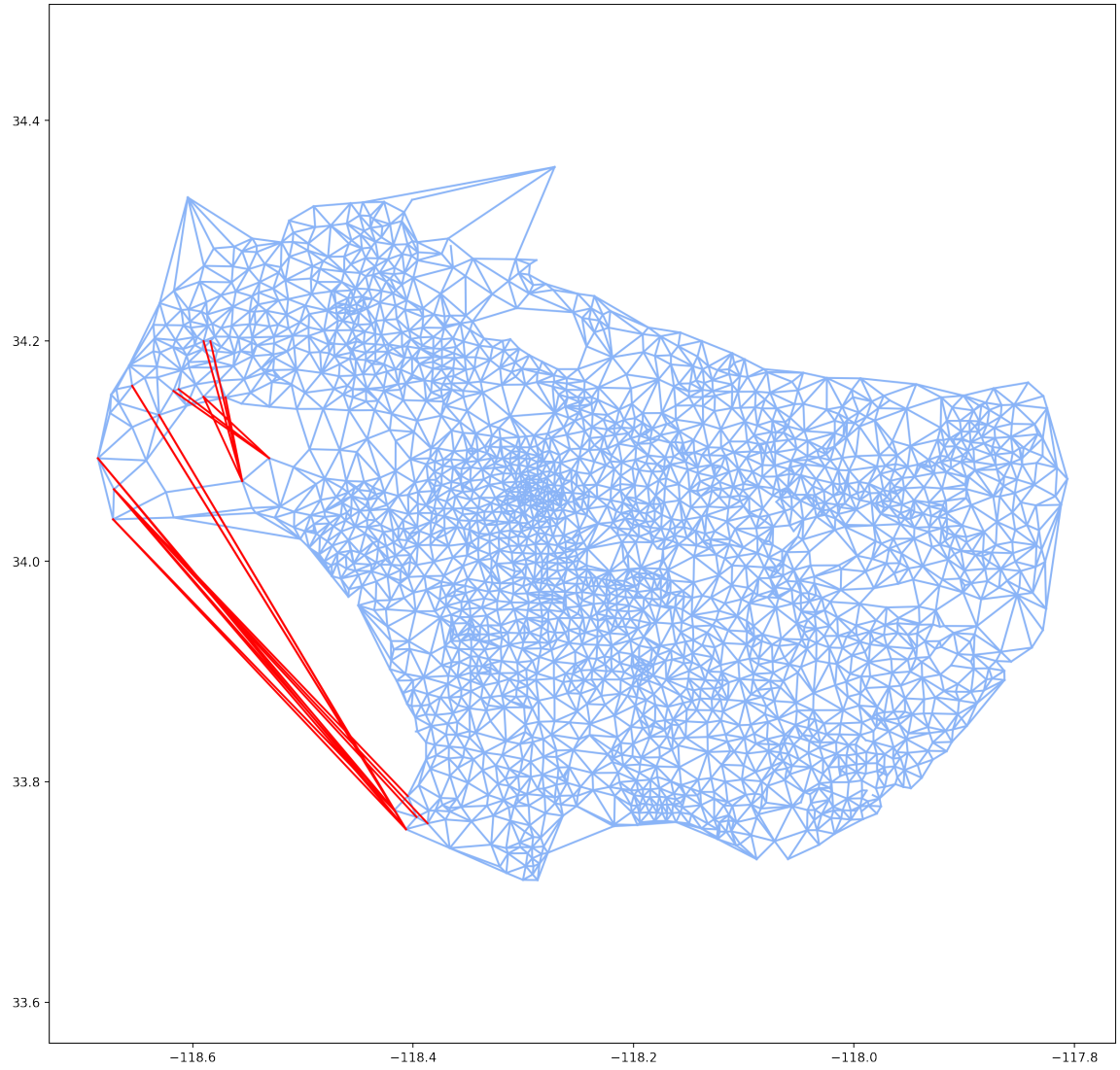


Figure 20: Strategy 4 - Static edges by travel time

Question 22 - Strategy 5
Similar to strategy 3, we create roads one by one while optimizing the travel time. The algorithm is as following:

1. Calculate shortest path travel time using Dijkstra algorithm.
2. Calculate shortest path distance using Dijkstra algorithm.
3. Update *travel_speed*.
4. Update *extra_time*.
5. Create a road between the pair with highest *extra_time* and update the graph.
6. Repeat 19 times for a total of 20 new edges.

Table 8 shows the list of edges created in strategy 5. Lengths of the newly created edges are included for further discussion in strategy 6. Figure 21 shows the new edges created on actual coordinates.

Added Edge	**Length**
[2262, 2419]	35.51
[2081, 2419]	33.59
[2140, 2419]	32.57
[2273, 2419]	30.92
[49, 2419]	29.94
[144, 1875]	27.28
[1783, 2416]	26.68
[285, 2419]	26.41
[1964, 2419]	26.03
[1783, 2417]	24.51
[2402, 2416]	24.23
[1717, 2419]	23.28
[1700, 1782]	22.78
[1956, 2419]	20.20
[2247, 2419]	16.63
[985, 1678]	7.31
[988, 1510]	5.33
[120, 687]	3.69
[1003, 1678]	3.22
[121, 1679]	2.55

Table 8: List of edges and their length in Strategy 5

Edges created by strategy 5 are similar to strategy 3. New roads are created to connect Angeles National Park, the concavity of the coast, and Topanga region. It is intuitive because these areas have high traveling time. Comparing to strategy 4, new edges are spread out.

The time it took to run on Colab was 86.99 seconds. Similar to strategy 3, Dijkstra shortest path algorithm is called in each iteration. One difference is to update the $travel\_speed$, we need to know the current $travel\_time\_of\_shortest\_path$ and $distance\_of\_shortest\_path$. For each iteration, we call Dijkstra algorithm twice, doubling the time from strategy 3. Therefore, the time complexity for strategy 4 is $O(40E + 40VlogV + 40V^2log(V^2))$.

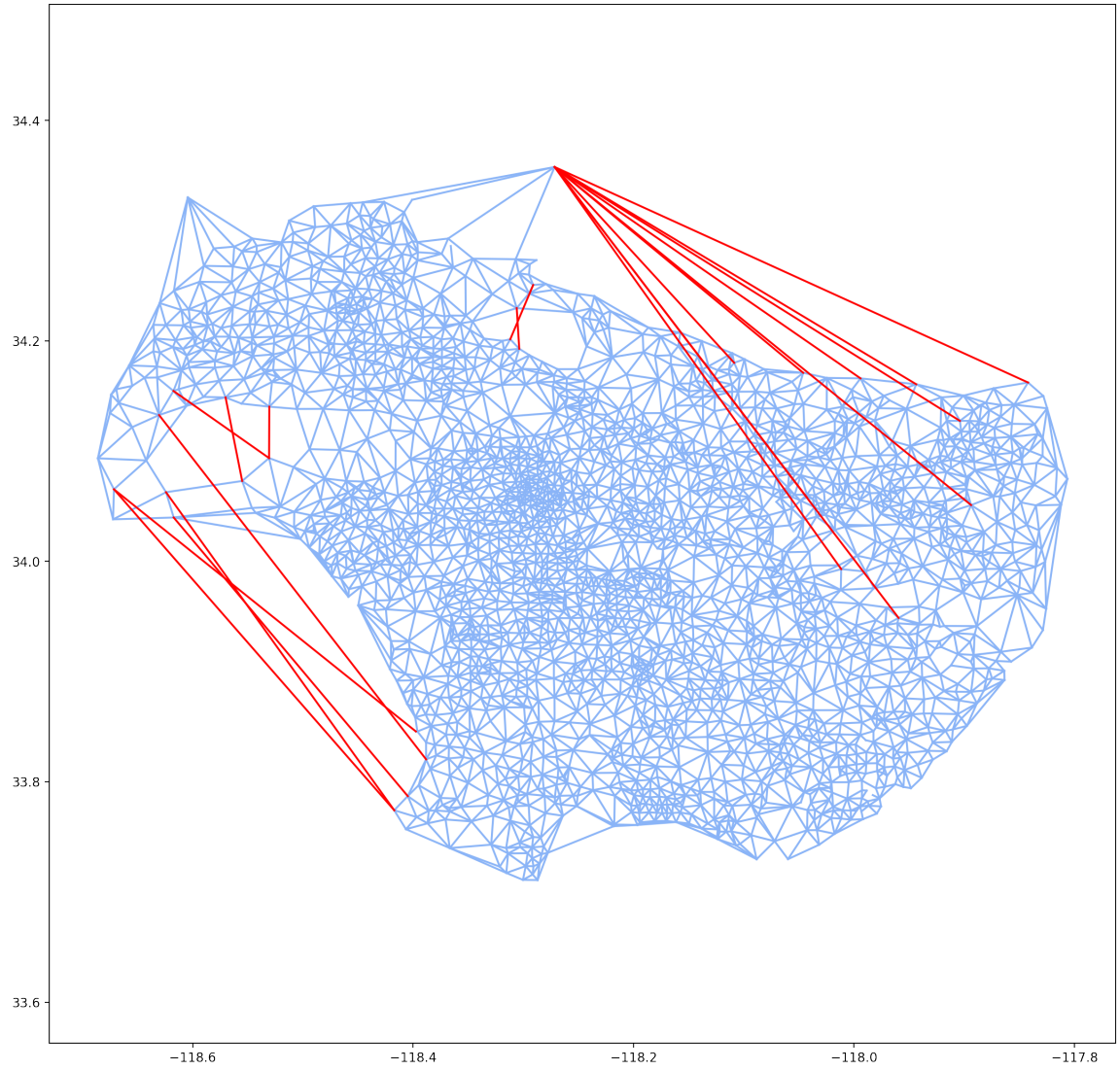


Figure 21: Strategy 5 - Dynamic edges by travel time

Question 23 - Strategy Comparison

Strategy 1 vs. Strategy 2

We can see that new edges in strategy 2 is similar to strategy 1. One vertex at the top, which is in the Angeles National Forest region, is shared with 16 new edges. It is reasonable because for the new edges to connect at the top vertex due to less existing roads in mountain region. In contrast to strategy 1, due to the randomness in frequency, there are 2 new edges connect at the Topanga region.

Strategy 2 seems to be better due to the 2 new edges connecting the Topanga region. However, both strategies are not good at distributing the new edges.

Strategy 1 vs. Strategy 3

There are noticeable changes in strategy 3 comparing to strategies 1 and 2. More new edges in this strategy connect the concavity of the beach. There are 2 new edges just below the Angeles

National Forest area. There are 2 edges that connect across the area, which will significantly reduce the traveling distance.

Strategy 3 seems to do a better job at distributing the new edges. As shortest paths are re-calculated each time a new edge is constructed, subsequent edges are able to built between regions where traveling distances are still considerable. However, due to the recalculation of shortest paths, strategy 3 is much slower to compute comparing to strategy 1(43.45 seconds vs. 4.48 seconds).

Strategy 1 vs. Strategy 4

New edges created by strategy 4 focus on connecting the concavity of the coast and the Topanga region. It is reasonable because the travel time near the coast is high due to low speed and high congestion. Topanga region, due to the mountain terrain, also has high travel time.

It seems that there is no winner between strategy 1 and strategy 4. Both focus on different regions. Strategy 1 is able to connect edges at the Angeles National Forest area, while strategy 4 is able to cover the mountainous Topanga region. Both strategies have edges connect the concavity of the beach, and both connect densely at selected regions. The two strategies take into consideration different aspects. Therefore, it depends on the preference of the decision maker to decide which aspect requires more attention.

Static vs. Dynamic

If we want to improve the overall road network by constructing new roads, dynamic strategies seem to be the better choice. As roads are built gradually, the shortest path algorithm is able to recalculate and find other regions that need roads to be built. New roads will not be as concentrated comparing to the one-time calculation.
If the goal is only to reduce traveling distances, then the optimal solution is the strategy 3. Because in each iteration, the algorithm finds the biggest gap between the distance of shortest path and that of a straight path to build a road.

Strategy 6 - Open Ended

Based on Table 8, we see that some new roads are too long and connect across the concavity of the beach. In all 5 strategies, all roads are concentrated in some regions. Let us assume that the government of LA wants to build high-speed roads that are distributed more evenly. We want to impose some constraints on constructing new roads:

- With limited budget, the lengths of new roads are between 5 miles and 20 miles.
- The vertices of new roads must not be within 5-mile radius of others. This constraint helps distributing new roads evenly.

We want to improve upon strategy 5 and propose 3 ways to satisfy the constraints. To impose these new constraints, the main goal is to manipulate $extra\_time$. In each iteration, a new road is built if the $extra\_time$ is the highest. Therefore, we want to lower $extra\_time$ if either condition is violated.

To satisfy the first condition, we can modify the $extra\_time$ as following:

$$extra\_time = travel\_time\_of\_shortest\_path - modified\_euclidean\_distance/travel\_speed$$

in which $modified\_euclidean\_distance$ is a copy of $euclidean\_distance$. The only difference with the original matrix is for any pair that violates the first condition (i.e. less than 5 or larger than 20), the values are set at an arbitrarily large value. We used a value of 10,000. A large $modified\_euclidean\_distance$ leads to small if not negative values for $extra\_time$. When it comes to the sorting stage, pairs with small $extra\_time$ do not get picked to build edges. Table 9 shows the edges added by strategy 6. We can see that the shortest road is 5.39 miles, and the longest road is 19.77 miles, satisfying the first condition.

Added Edge	**Length**
[321, 2416]	19.77
[777, 1317]	19.13
[1483, 2491]	18.87
[1108, 2029]	18.85
[1799, 2419]	18.49
[1916, 2507]	18.34
[1783, 1941]	17.97
[523, 1995]	17.93
[396, 2084]	17.91
[1502, 2047]	17.77
[2096, 2619]	17.26
[908, 1403]	16.63
[1661, 2563]	15.73
[423, 2321]	15.67
[654, 1770]	12.87
[888, 2244]	10.53
[513, 2060]	9.27
[985, 1678]	7.32
[122, 1684]	5.97
[731, 2125]	5.39

Table 9: List of edges and their length in Strategy 6

For the second condition, we first attempted to use the neighborhood function in the igraph library to locate the neighbors of the newly created edges. However, the calculation was prohibitively slow, taking more than 10 minutes just to create 9 new edges. We decided to abandon the approach and came up with the algorithm as follows:

- Create a copy of the pairwise *euclidean_distance* and convert it to a boolean matrix, returning True if smaller than 5 miles. We named this matrix *is_neighbor*.
- Create a list named *vertices_to_exclude*, which contains vertices of newly created edges.
- During each iteration, before sorting and finding the edge with the highest *extra_time*, we need to obtain a list of vertices that are neighboring to those in *vertices_to_exclude*. It can be achieved by filtering for True in rows (or columns) in *is_neighbor* with indexes matched with the list *vertices_to_exclude*. The filtered positions of True cells are the neighboring vertices. Let us call this list *neighbors*.
- Finally, in the *extra_time* matrix, we set rows and columns whose indexes matched with the *neighbors* the value of 0, ensuring that these neighboring vertices will not get picked at the sorting step.

Figure 22 shows the added edges for strategy 6, and Figure 23 show edges with 5-mile radius. We can see a few vertices that are close to others, but no vertex truly violates other areas.

While the second condition makes sure no roads are built too close to each other, it does not guarantee distribution. We employed another strategy to further encourage even distribution. One way is to make sure the shortest path algorithm utilizes the newly created edges so that the areas surrounding these new roads are less likely to get picked after sorting. In strategy 5, after adding a new edge at the end of an iteration, we need to add the distance and traveling time for the edge for subsequent iterations. The traveling time is calculated by using the straight path distance divide by the speed. In strategy 6, we multiply the traveling time with a 'disperse factor' of 0.12. The resulting traveling time for the newly added edge is now much smaller, incentivizing the shortest

path algorithm. We can see in Figure 22 that new roads cover all areas in the map with different directions and orientations.

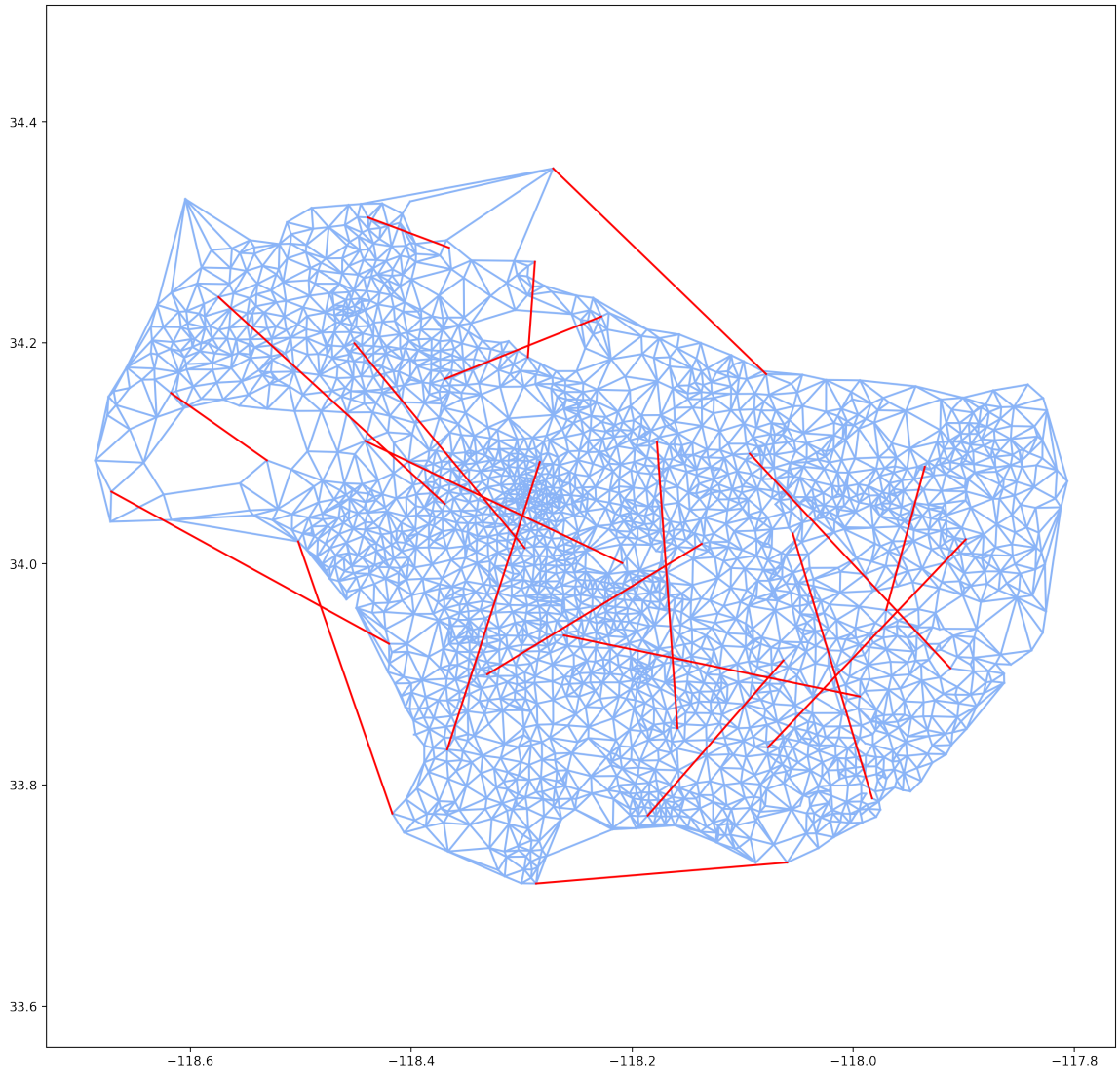


Figure 22: Strategy 6 - Dynamic edges by travel time, with constraints

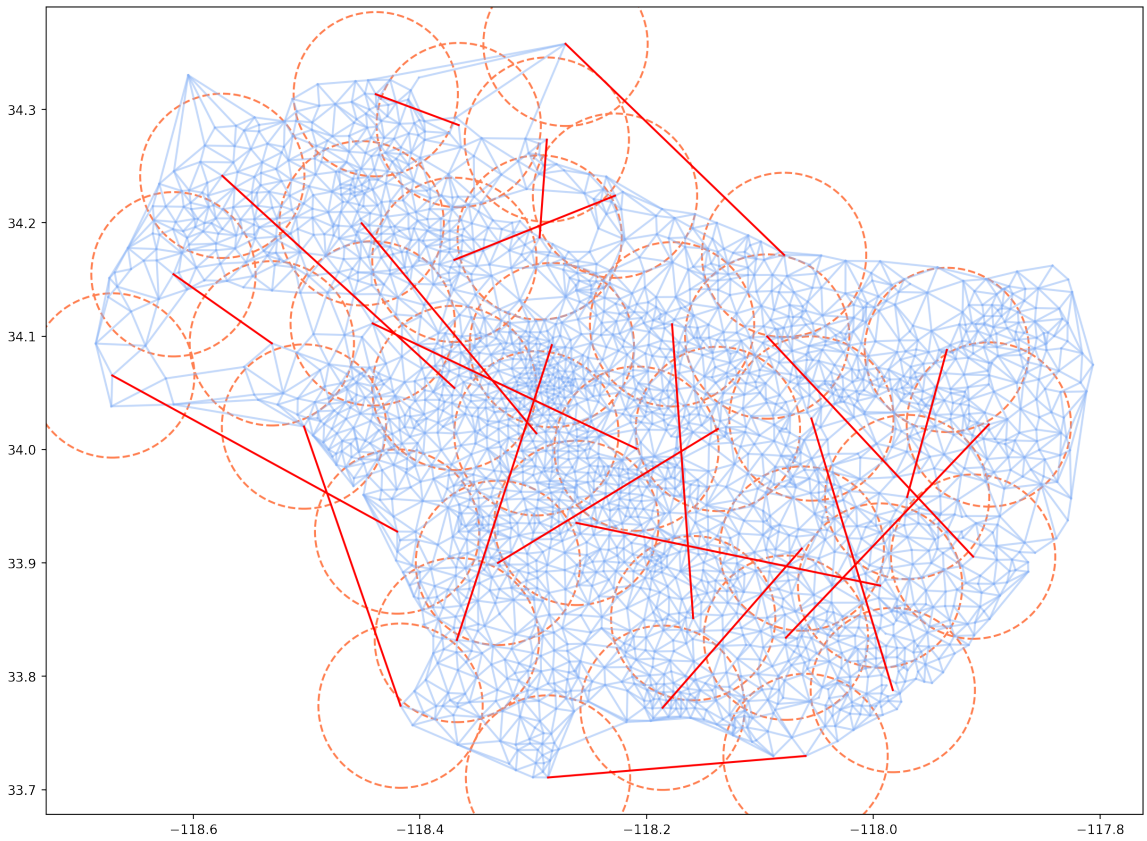


Figure 23: Strategy 6 - Vertices of new roads and their vicinity

All three implementations to enforce the constraints are run with nominal burden. The time it took to run on Colab for strategy 6 was 119.02 seconds vs. 120.69 seconds in strategy 5. All implementations require one-off calculations, and filtering arrays based on criteria utilizes efficiency of Numpy arrays.

In summary, constraint implementations in strategy 6 boil down to the control of the *extra_time* matrix. The first constraint implementation indirectly suppresses *extra_time* via lowering $modified\_euclidean\_distance$ or $travel\_time\_of\_straight\_path$. The second implementation directly zeros *extra_time* if the pairs violate the constraint. The third encourages distribution by locally lowering *extra_time* at newly created edges via $travel\_time\_of\_shortest\_path$.

Part 3 - Multi Depot Capacitated Vehicle Routing Problem

In this task, we look into capacitated vehicle routing problem (CVRP). We use real-world data, obtained from the following link:

https://www.kaggle.com/datasets/hemanthboddapu/sample-data-shipments-vehicle-routing-simulation

The dataset contains weights associated with Indian cities in 2020. For this task, we use weights as demands for each city. To simplify the task, we average demands by cities. There are 92 cities with various demands, ranging from 0 to 15. We assume that there are 5 depots situated at 5 major cities - Delhi, Mumbai, Bangalore, Hyderabad, and Kolkata. We have in total 15 trucks, each of which has the capacity of 50. The task is to identify the optimal truck route that (1) minimizing the distance, (2) carry all goods that satisfy the cities' demands, and (3), without overloading the truck (rolling total carried goods is not larger than truck capacity).

Find the Optimal Route

We utilized ORtools library created by Google to work on this task. We model the problem as the following:

- Find pairwise shortest distances among 92 cities. The optimal route is found by using this metric.
- Each depot has 3 trucks, each with capacity of 50.
- Each truck starts and ends at the same depot.

It should be noted that the ORtools library finds approximate solution and not optimal solution. Fortunately, its approximate solution is close to the optimal solution and takes much less time than an exhaustive search; and we shall call . We used the following heuristic in our solver:

- The strategy to find the first solution: path with the cheapest arc. In other words, a truck starts from its depot (the "start" node) and goes to the nearest city, then extend the route by iterating on the last node added to the route.

The solver was efficient and only took 1 second to find the solution. The route is presented in the Appendix. The total distance of all routes is 21,233 km.

Find the Optimal Vehicle Distribution

After setting up 3-truck-per-depot as our baseline, we want to find the optimal vehicle distribution. That is, we want to find the number of trucks for each depot resulting in the smallest travel distance. We model the problem as following:

- There are 15 available trucks in total.
- Among 5 depots, each should have at least 1 vehicle. It means each depot may have a number of trucks ranging from 1 to 11.

- The objective is to scan the hypothesis space to find the minimum total travel distance.

With the above constraints, the hypothesis space has 1001 plans. Running the ORTools solver for all plans took 16 minutes. We are able to find the optimal vehicle distribution as following:

- Bangalore depot: 1 truck
- Delhi depot: 5 trucks
- Hyderabad depot: 3 trucks
- Kolkata depot: 3 trucks
- Mumbai depot: 3 trucks

The optimal vehicle distribution reflects the demands for areas surrounding each depot. We can see that Bangalore area does not have much demand and only has 1 truck to operate. On the other hand, Delhi surrounding demand's is high and receives 5 trucks. Detailed truck routes are presented in Appendix. By using the above truck distribution, the total distance of all routes is 20,895 km, saving 338 km from the baseline plan.

Appendix

The Optimal Route for 3-truck-per-depot

Route for vehicle 1: Bangalore, Load(0.00) - Salem, Load(5.72) - Chennai, Load(13.14) - Nellore, Load(22.41) - Guntur, Load(31.58) - Kurnool, Load(39.76) - Bangalore, Load(39.76)
Distance of the route: 1383km
Load of the route: 39.76

Route for vehicle 2: Bangalore, Load(0.00) - Tiruppur, Load(7.77) - Coimbatore, Load(12.26) - Kochi, Load(18.45) - Thiruvananthapuram, Load(27.13) - Tinnevelly, Load(32.89) - Madurai, Load(41.05) - Trichinopoly, Load(46.41) - Bangalore, Load(46.41)
Distance of the route: 1174km
Load of the route: 46.41

Route for vehicle 3: Bangalore, Load(0.00) - Mysore, Load(7.03) - Shimoga, Load(13.67) - Malegaon, Load(18.94) - Dhulia, Load(24.72) - Indore, Load(31.30) - Jalgaon, Load(38.48) - Aurangabad, Load(46.32) - Bangalore, Load(46.32)
Distance of the route: 2475km
Load of the route: 46.32

Route for vehicle 4: Delhi, Load(0.00) - Moradabad, Load(5.35) - Bareilly, Load(11.43) - Lucknow, Load(20.24) - Cawnpore, Load(27.81) - Jabalpur, Load(38.77) - Jaipur, Load(45.98) - Delhi, Load(45.98)
Distance of the route: 1719km
Load of the route: 45.98

Route for vehicle 5: Delhi, Load(0.00) - Ludhiana, Load(5.36) - Jalandhar, Load(12.01) - Amritsar, Load(18.09) - Jammu, Load(23.25) - Srinagar, Load(30.22) - Chandigarh, Load(35.27) - Dehra Dun, Load(39.90) - Saharanpur, Load(45.51) - Delhi, Load(45.51)
Distance of the route: 1433km
Load of the route: 45.51

Route for vehicle 6: Delhi, Load(0.00) - Faridabad, Load(10.49) - Gwalior, Load(19.18) - Agra, Load(25.53) - Aligarh, Load(32.94) - Ghaziabad, Load(40.47) - Delhi, Load(40.47)
Distance of the route: 597km
Load of the route: 40.47

Route for vehicle 7: Hyderabad, Load(0.00) - Ujjain, Load(4.67) - Jodhpur, Load(12.96) - Bikaner, Load(24.59) - Ajmer, Load(28.54) - Kota, Load(34.87) - Bhopal, Load(42.01) - Hyderabad, Load(42.01)
Distance of the route: 2666km
Load of the route: 42.01

Route for vehicle 8: Hyderabad, Load(0.00) - Bezwada, Load(5.54) - Vishakhapatnam, Load(15.90)

- Raipur, Load(24.86) - Bhilai, Load(34.10) - Warangal, Load(44.71) - Hyderabad, Load(44.71)
Distance of the route: 1554km
Load of the route: 44.71

Route for vehicle 9: Hyderabad, Load(0.00) - Chanda, Load(7.73) - Nagpur, Load(15.37) - Amravati, Load(21.72) - Nanded, Load(29.40) - Gulbarga, Load(44.37) - Hyderabad, Load(44.37)
Distance of the route: 1156km
Load of the route: 44.37

Route for vehicle 10: Kolkata, Load(0.00) - Dispur, Load(8.29) - Guwahati, Load(15.85) - Patna, Load(21.82) - Gorakhpur, Load(29.07) - Allahabad, Load(37.02) - Mirzapur, Load(41.61) - Ranchi, Load(47.46) - Kolkata, Load(47.46)
Distance of the route: 2369km
Load of the route: 47.46

Route for vehicle 11: Kolkata, Load(0.00) - Cuttack, Load(8.39) - Bhubaneshwar, Load(14.47) - Raurkela, Load(19.35) - Varanasi, Load(29.99) - Gaya, Load(38.00) - Jamshedpur, Load(45.13) - Kolkata, Load(45.13)
Distance of the route: 1674km
Load of the route: 45.13

Route for vehicle 12: Kolkata, Load(0.00) - Haora, Load(4.84) - Durgapur, Load(14.86) - Asansol, Load(23.54) - Dhanbad, Load(32.64) - Kolkata, Load(32.64)
Distance of the route: 484km
Load of the route: 32.64

Route for vehicle 13: Mumbai, Load(0.00) - Surat, Load(8.40) - Vadodara, Load(14.49) - Ahmadabad, Load(22.43) - Jamnagar, Load(35.16) - Rajkot, Load(42.53) - Bhavnagar, Load(46.42) - Mumbai, Load(46.42)
Distance of the route: 1285km
Load of the route: 46.42

Route for vehicle 14: Mumbai, Load(0.00) - Bhayandar, Load(5.63) - Bhiwandi, Load(10.80) - Nasik, Load(17.32) - Chinchvad, Load(24.43) - Pune, Load(32.22) - Hubli, Load(37.94) - Kolhapur, Load(41.30) - Mumbai, Load(41.30)
Distance of the route: 1169km
Load of the route: 41.30

Route for vehicle 15: Mumbai, Load(0.00) - Ulhasnagar, Load(8.48) - Kalyan, Load(17.76) - Thane, Load(26.89) - Mumbai, Load(26.89)
Distance of the route: 95km
Load of the route: 26.89

Total distance of all routes: 21,233 km
Total load of all routes: 635.39

The Optimal Route with Optimized Vehicle Distribution

Route for vehicle 1: Bangalore, Load(0.00) - Salem, Load(5.72) - Trichinopoly, Load(11.07) - Madurai, Load(19.23) - Tinnevelly, Load(24.99) - Thiruvananthapuram, Load(33.67) - Kochi, Load(39.86) - Coimbatore, Load(44.35) - Bangalore, Load(44.35)
Distance of the route: 1153km
Load of the route: 44.35

Route for vehicle 2: Delhi, Load(0.00) - Bikaner, Load(11.62) - Jodhpur, Load(19.91) - Ajmer, Load(23.87) - Jaipur, Load(31.08) - Agra, Load(37.43) - Aligarh, Load(44.84) - Delhi, Load(44.84)
Distance of the route: 1289km
Load of the route: 44.84

Route for vehicle 3: Delhi, Load(0.00) - Gwalior, Load(8.69) - Jabalpur, Load(19.65) - Bhilai, Load(28.89) - Raipur, Load(37.86) - Cawnpore, Load(45.43) - Delhi, Load(45.43)
Distance of the route: 1936km
Load of the route: 45.43

Route for vehicle 4: Delhi, Load(0.00) - Moradabad, Load(5.35) - Bareilly, Load(11.43) - Bhopal, Load(18.58) - Jalgaon, Load(25.76) - Dhulia, Load(31.53) - Indore, Load(38.11) - Kota, Load(44.44) - Delhi, Load(44.44)
Distance of the route: 2137km
Load of the route: 44.44

Route for vehicle 5: Delhi, Load(0.00) - Ludhiana, Load(5.36) - Jalandhar, Load(12.01) - Amritsar, Load(18.09) - Jammu, Load(23.25) - Srinagar, Load(30.22) - Chandigarh, Load(35.27) - Dehra Dun, Load(39.90) - Saharanpur, Load(45.51) - Delhi, Load(45.51)
Distance of the route: 1433km
Load of the route: 45.51

Route for vehicle 6: Delhi, Load(0.00) - Faridabad, Load(10.49) - Ghaziabad, Load(18.02) - Delhi, Load(18.02)
Distance of the route: 71km
Load of the route: 18.02

Route for vehicle 7: Hyderabad, Load(0.00) - Gulbarga, Load(14.97) - Mysore, Load(22.00) - Tiruppur, Load(29.77) - Chennai, Load(37.20) - Kurnool, Load(45.37) - Hyderabad, Load(45.37)
Distance of the route: 1832km
Load of the route: 45.37

Route for vehicle 8: Hyderabad, Load(0.00) - Nanded, Load(7.68) - Aurangabad, Load(15.53) - Malegaon, Load(20.80) - Ujjain, Load(25.47) - Amravati, Load(31.82) - Nagpur, Load(39.46) - Chanda, Load(47.19) - Hyderabad, Load(47.19)
Distance of the route: 1777km
Load of the route: 47.19

Route for vehicle 9: Hyderabad, Load(0.00) - Warangal, Load(10.61) - Vishakhapatnam, Load(20.97) - Bezwada, Load(26.51) - Guntur, Load(35.67) - Nellore, Load(44.95) - Hyderabad, Load(44.95)
Distance of the route: 1451km
Load of the route: 44.95

Route for vehicle 10: Kolkata, Load(0.00) - Dispur, Load(8.29) - Guwahati, Load(15.85) - Patna, Load(21.82) - Gorakhpur, Load(29.07) - Lucknow, Load(37.88) - Allahabad, Load(45.82) - Kolkata, Load(45.82)

Distance of the route: 2568km
Load of the route: 45.82

Route for vehicle 11: Kolkata, Load(0.00) - Cuttack, Load(8.39) - Bhubaneshwar, Load(14.47) - Raurkela, Load(19.35) - Mirzapur, Load(23.94) - Varanasi, Load(34.58) - Gaya, Load(42.58) - Kolkata, Load(42.58)
Distance of the route: 1677km
Load of the route: 42.58

Route for vehicle 12: Kolkata, Load(0.00) - Haora, Load(4.84) - Durgapur, Load(14.86) - Asansol, Load(23.54) - Dhanbad, Load(32.64) - Ranchi, Load(38.50) - Jamshedpur, Load(45.63) - Kolkata, Load(45.63)
Distance of the route: 696km
Load of the route: 45.63

Route for vehicle 13: Mumbai, Load(0.00) - Surat, Load(8.40) - Vadodara, Load(14.49) - Ahmadabad, Load(22.43) - Jamnagar, Load(35.16) - Rajkot, Load(42.53) - Bhavnagar, Load(46.42) - Mumbai, Load(46.42)
Distance of the route: 1285km
Load of the route: 46.42

Route for vehicle 14: Mumbai, Load(0.00) - Bhayandar, Load(5.63) - Nasik, Load(12.15) - Chinchvad, Load(19.26) - Pune, Load(27.05) - Hubli, Load(32.77) - Shimoga, Load(39.41) - Kolhapur, Load(42.77) - Mumbai, Load(42.77)
Distance of the route: 1489km
Load of the route: 42.77

Route for vehicle 15: Mumbai, Load(0.00) - Ulhasnagar, Load(8.48) - Kalyan, Load(17.76) - Bhiwandi, Load(22.93) - Thane, Load(32.06) - Mumbai, Load(32.06)
Distance of the route: 101km
Load of the route: 32.06

Total distance of all routes: 20,895 km

Total load of all routes: 635.39