ECE 232E Spring 2023

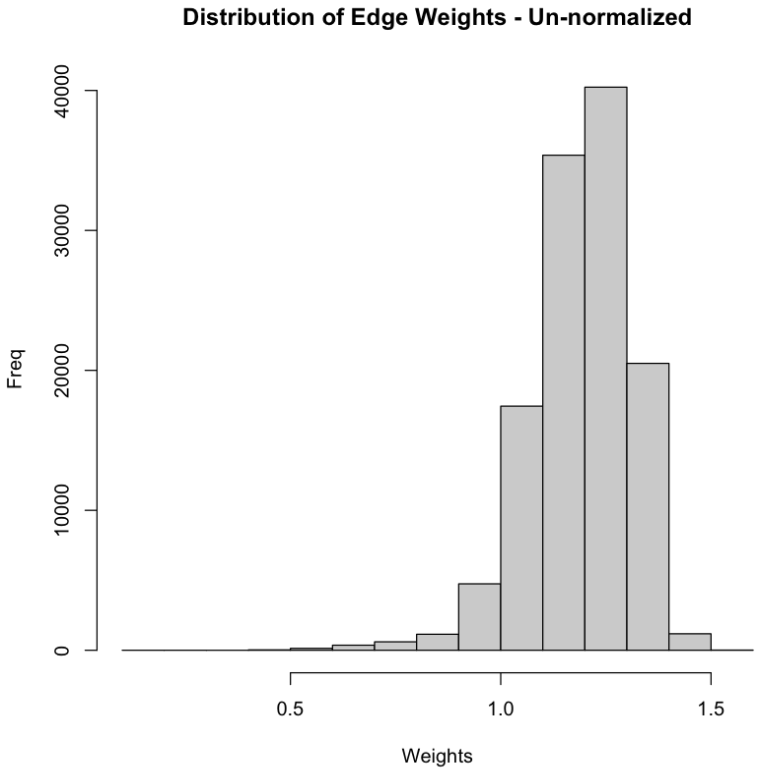
Graph Algorithms

Question 1

The upper bound and lower bound are due to Pearson's coefficient. As per the Cauchy-Shwarz Inequality, the bounds of ρ range from [-1,+1]. The value of -1 indicates that the stocks are inversely correlated and move in opposite directions. The value of +1 indicates that the stocks are positively correlated and move in the same direction.

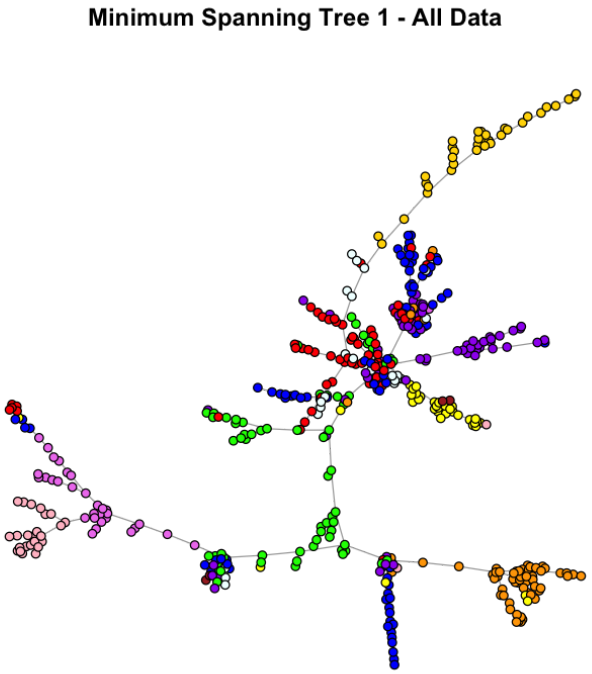
One of the reasons why regular return is not preferred and log-normalized return is preferred is that it eases possible data skewness. Further, it also lessens the negative effect of variance and outliers on the data. Also, it normalizes and balances the data.

Question 2



Question 3

To construct the Minimum Spanning Tree (MST), we make use of Prim's algorithm. The figure below depicts different sectors, with each sector represented by a specific color. It is observed that most stocks that belong to the same sector/color scheme are grouped together. Stocks that have a lower correlation appear to be further apart and thus, have larger edge weights. If stocks belong to the same sector/color scheme, they change in the same direction.



Question 4

The figure below depicts the MST along with the communities calculated using *walktrap*. To evaluate the clustering, two metrics i.e., homogeneity and completeness score are used. Homogeneity refers to how well clusters contain similar datapoints that belong to a particular class i.e., their "purity". Homogeneity is calculated using the below equation:

homogeneity = 1 - (H(C|K) / H(C))

where, H(C|K) is the conditional entropy of the true classes given the cluster assignments and H(C) is the entropy of the true classes.

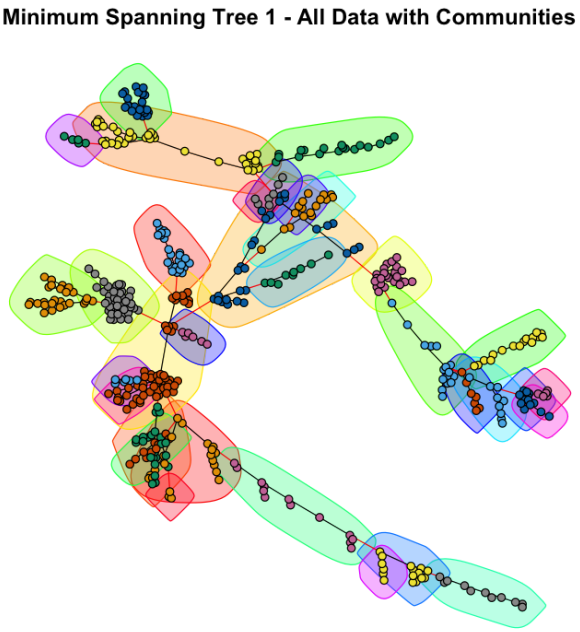
Completeness score indicated how "full" a class' representation is and whether there are datapoints from that class that are represented in other clusters. Completeness score is calculated using the below equation:

completeness = 1 - (H(K|C) / H(K))

where, H(K|C) is the conditional entropy of the cluster assignments given the true classes and H(K) is the entropy of the cluster assignments.

The homogeneity and completeness scores calculated using the *clever* library are:

1. Homogeneity Score: 0.682644648161367
2. Completeness Score: 0.479284479244588



Question 5

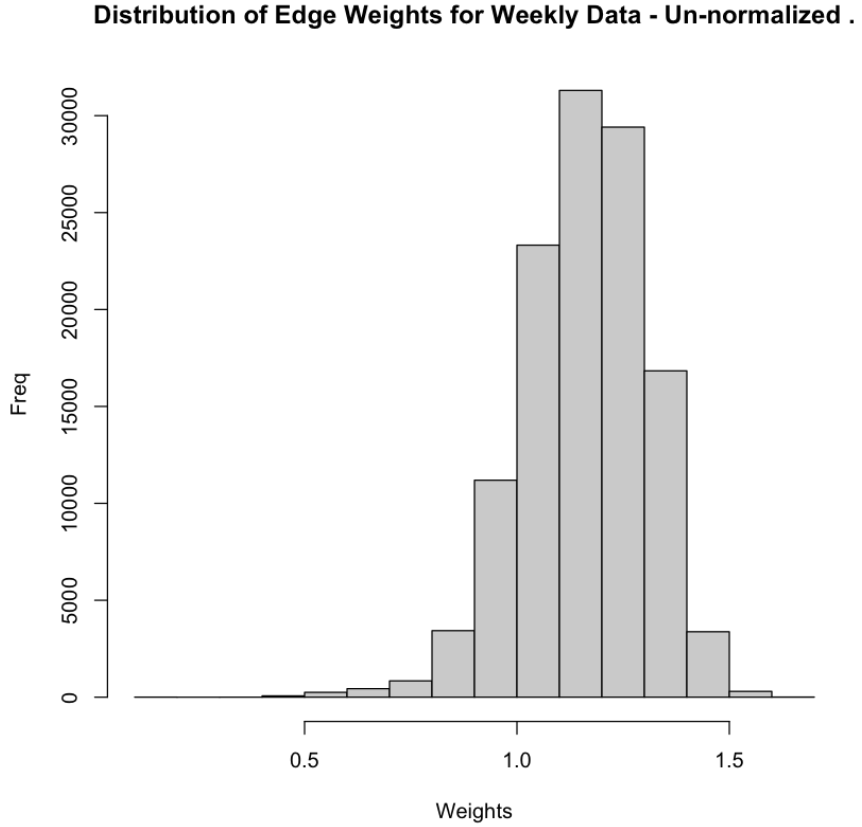
Two variations of alpha calculation using Qi and Si values are used, where Qi refers to node neighbors and Si refers to node sectors. It is observed that the first method (using Qi) has a higher score. This is because the first method takes into account nodes that have high correlation by exploiting the MST clusters formed.

The scores calculated are:

1. $\alpha_1 = 0.828930$
2. $\alpha_2 = 0.114188$

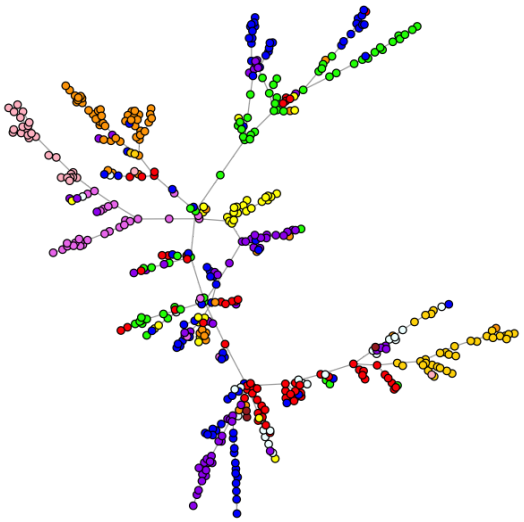
Question 6

The above steps were applied to weekly data for the day *Monday*. The un-normalized distribution of edge weights for weekly data is seen below

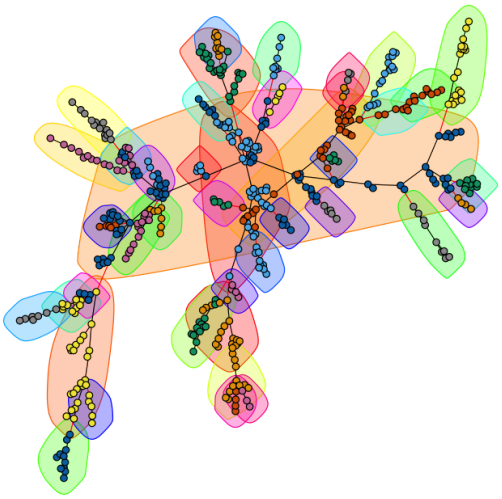


To construct the Minimum Spanning Tree (MST), we make use of Prim's algorithm. The figure below depicts different sectors, with each sector represented by a specific color. It is observed that although similar colors are moderately well-grouped, stocks belonging to the same sector/color scheme are less grouped together than MST 1, indicating a poorer performance with weekly data as opposed to daily.

Minimum Spanning Tree 2 - Weekly Data



Minimum Spanning Tree 2 - Weekly Data with Communities



The homogeneity and completeness scores calculated using only weekly data are:

1. Homogeneity Score: 0.581123705715172
2. Completeness Score: 0.390043529189445

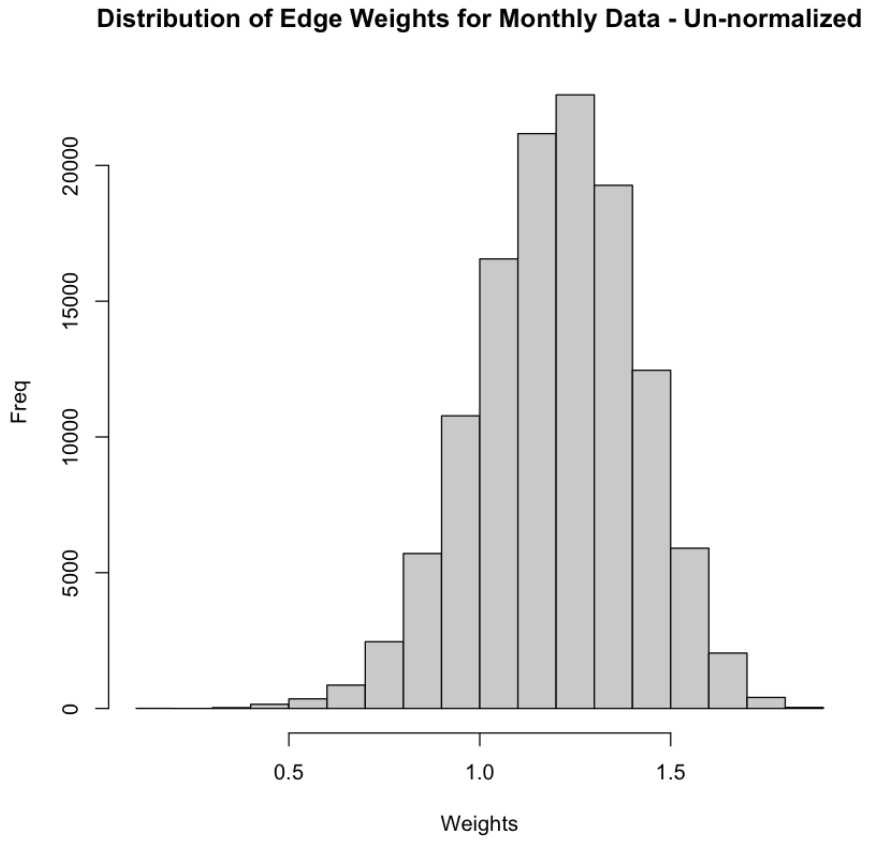
Two variations of alpha calculation using Qi and Si values are used, where Qi refers to node neighbors and Si refers to node sectors. It is observed that the first method (using Qi) has a higher score. This is because the first method takes into account nodes that have high correlation by exploiting the MST clusters formed.

The alpha scores calculated are:

1. $\alpha_1 = 0.743957$
2. $\alpha_2 = 0.114309$

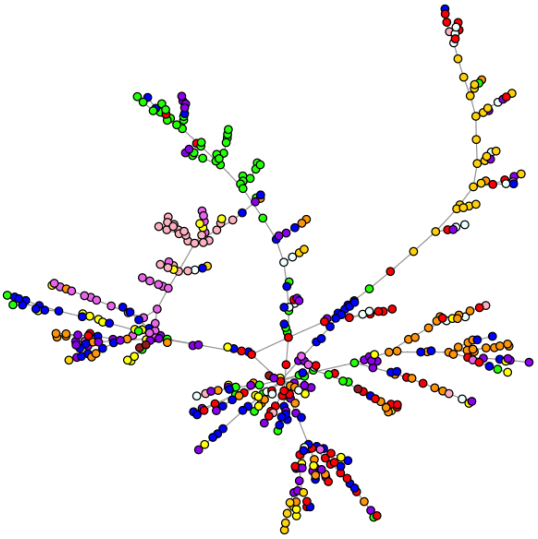
Question 7

The above steps were applied to monthly data for the date *15th*. The un-normalized distribution of edge weights for monthly data is seen below

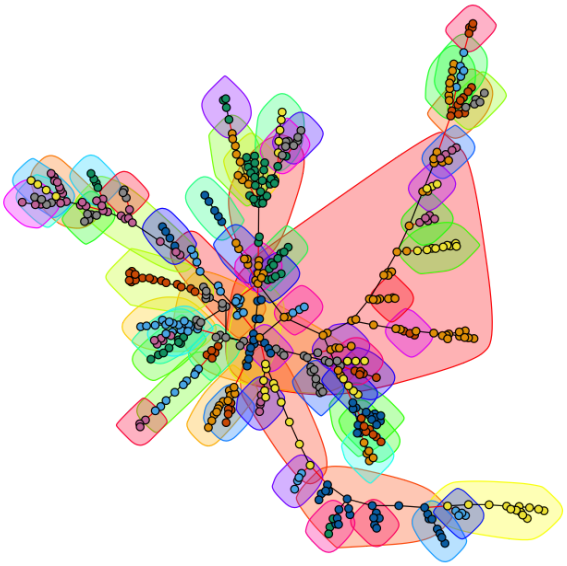


To construct the Minimum Spanning Tree (MST), we make use of Prim's algorithm. The figure below depicts different sectors, with each sector represented by a specific color. It is observed that stocks belonging to the same sector/color scheme are further scattered than MST 2, indicating a poorer performance with monthly data as opposed to weekly and daily data.

Minimum Spanning Tree 3 - Monthly Data



Minimum Spanning Tree 3 - Monthly Data with Communities



The homogeneity and completeness scores calculated using only monthly data are:

1. Homogeneity Score: 0.479447330844777
2. Completeness Score: 0.27755124113306

Two variations of alpha calculation using Qi and Si values are used, where Qi refers to node neighbors and Si refers to node sectors. It is observed that the first method (using Qi) has a higher score. This is because the first method takes into account nodes that have high correlation by exploiting the MST clusters formed.
The alpha scores calculated are:

1. $\alpha_1 = 0.484446$
2. $\alpha_2 = 0.114309$

Question 8

On comparing daily, weekly, and monthly data, few observations can be made.

Data	**Homogeneity Score**	**Completeness Score**	**α_1**	**α_2**
Daily	0.6826446	0.4792845	0.828930	0.114188
Weekly (*Monday*)	0.5811237	0.3900435	0.743957	0.114309
Monthly (*15th*)	0.4794473	0.2775512	0.484446	0.114309

Since higher homogeneity and completeness scores are indicative of how well the clustering has been done, we can see that given the above values, weekly data outperforms monthly, and daily data outperforms weekly. This is also clear in the generated MSTs. The same trend is followed in the α_1 values, whereas, since α_2 does not make use of neighbor data, the values do not see a drastic difference. These results along with the MSTs indicate a higher granularity in daily data when compared to weekly and monthly data.

Let's Help Santa

Question 9

In this question, we are asked to create a graph from the Uber movements data and report the number of nodes and edges present in it. After creating the graph, we realised that it is a disconnected graph and therefore to ease downstream tasks, the giant connected components was taken. The GCC has a total of 2,649 nodes and a total of 100,358 edges connecting it.

Question 10

In this question, we are asked to generate a MST (Minimum Spanning Tree) of the graph created in the previous question and report a few of the street addresses that connect two edges in the MST. A visualisation of the graph can be seen in Fig.1 and Table 1 which reports the street addresses and the distance between the points for 5 such edges.

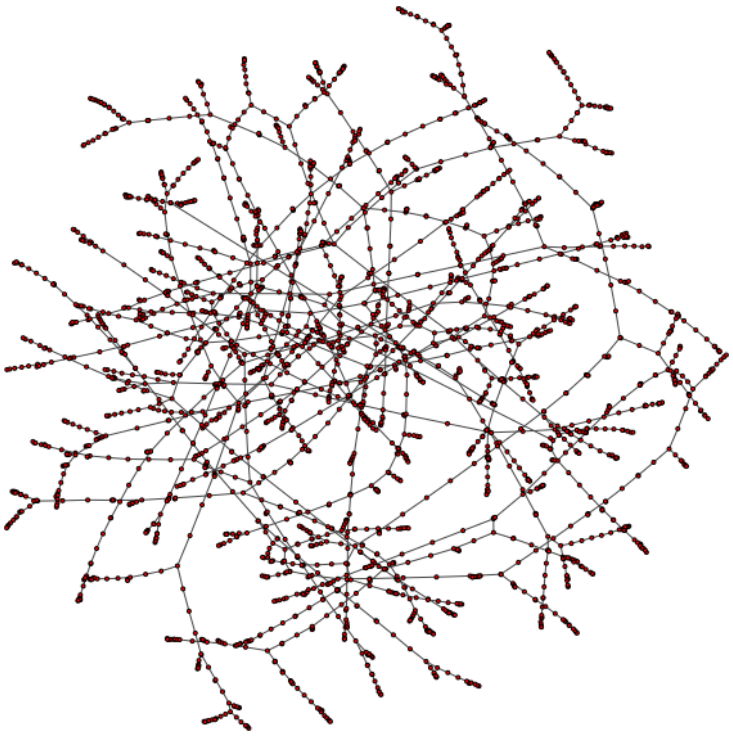


Fig. 1 Visualiastion of MST Graph

Table 1: Edges of a MST Graph with Addresses and Distance

Source Node	Address	Destination Node	Address	Distance
[34.10309557, -118.12053321]	823 E Grand Ave, Alhambra, CA 91801	[34.09626386, -118.13138209]	300 N 3rd St, Alhambra, CA 91801	0.572mi
[34.09645121, -118.13785063]	400 N Marguerita Ave, Alhambra, CA 91801	[34.08538654, - 118.14184446]	500 S Marengo Ave, Alhambra, CA 91803	0.618mi
[33.86132792, -118.08062876]	11830-11850 187th St, Artesia, CA 90701	[33.85443324, -118.07799459]	11932 Bos St, Cerritos, CA 90703	0.357mi
[33.98065181, -118.19460307]	6217-6365 Riverside Ave, Bell, CA 90201	[33.97718283, -118.19844176]	6615 Bear Ave, Bell, CA 90201	0.486mi
[33.88616168, -118.1956265]	1123 S Stoneacre Ave, Compton, CA 90221	[33.87661236, -118.19691603]	301 E Cummings Ln, Long Beach, CA 90805	0.665mi

From the Table, we do see that the distance between edges is small, which is obvious because a Minimum Spanning Tree (MST) attempts to minimise the sum of edge weights and in the process of doing so tries to find a way to touch all vertices in minimum weight.

Question 11

The triangle inequality is a mathematical property that the sum of any two sides of a triangle is greater than the third side and the physical realisation of this is that time it takes to traverse from one point to another will be smaller than if taken through a proxy node which. In this experiment we sampled 1000 such points in our graph and the triangle inequality holds 92.6% of the time.

Question 12

In this question, we were asked to implement a *1-approximation* of the TSP (Travelling Salesman Program) and report the empirical upper bound derived from our implementation.

In the *1-approximate* algorithm the following steps are done:

1. Create a Minimum Spanning Tree (MST) of the graph
2. Create a multi-graph (each edge is replaced by two directed edges) from the MST
3. Find the Euler cycle in the multi-graph
4. Pick the first appearance of a node in the Euler Cycle to get a tour sequence

From the lecture, we were able to theoretically derive that the length of the tour returned by the algorithm and the theoretical derivations are listed below, with the final theoretical limited

$$Weight\ of\ MST\ \leq\ Weight\ of\ Minimum\ Tour\ Length \tag{1}$$

$$Length\ of\ Euler\ Tour\ =\ 2\ x\ Weight\ of\ MST \tag{2}$$

$$Length\ of\ calculated\ tour\ \leq\ Length\ of\ Euler\ Tour \tag{3}$$

$$Length\ of\ calculated\ tour\ \leq\ 2\ x\ Weight\ of\ MST \tag{4}$$

$$Length\ of\ calculated\ tour\ \leq\ 2\ *\ Weight\ of\ Minimum\ Tour\ Length \tag{5}$$

$$\rho = \frac{Approximate\ TSP\ Cost}{Optimal\ TSP\ Cost} \leq 2 \quad \text{(Theoretical Upper Bound)} \tag{6}$$

Where equation (6) is just rearranging equation (5). In the practical implementation of it, the weight of the MST was *269084* and the weight of the calculated tour was *421489* which put t ogether gives us a practical upper bound of ~1.57, thus empirically we get

$$\rho = \frac{Approximate\ TSP\ Cost}{Optimal\ TSP\ Cost} \leq 1.57 \quad \text{(Empirical Upper Bound)} \tag{7}$$

Question 13

In this problem we were asked to plot Santa's trajectory around Los Angeles using the Travelling Salesman Problem algorithm. The nodes representing the points in the Uber data can be seen in Fig.2 and the actual trajectory can be seen in Fig. 3

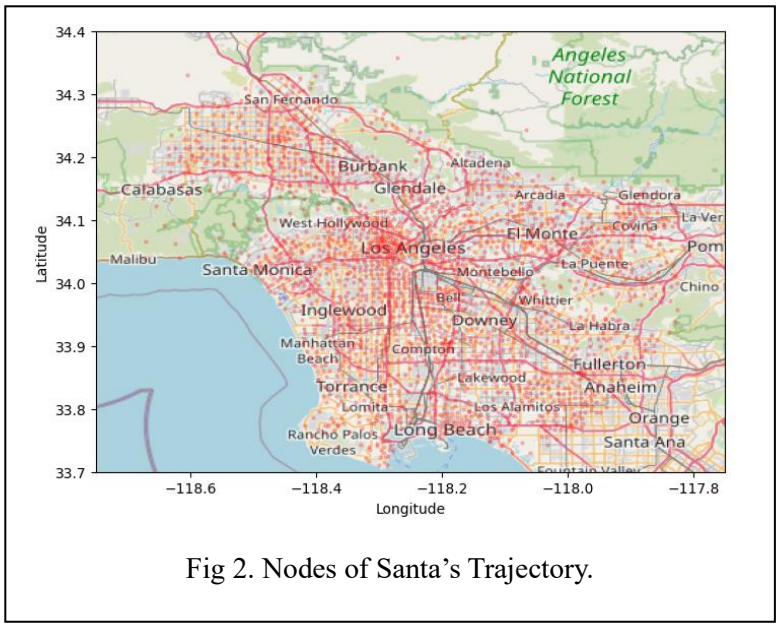


Fig 2. Nodes of Santa's Trajectory.

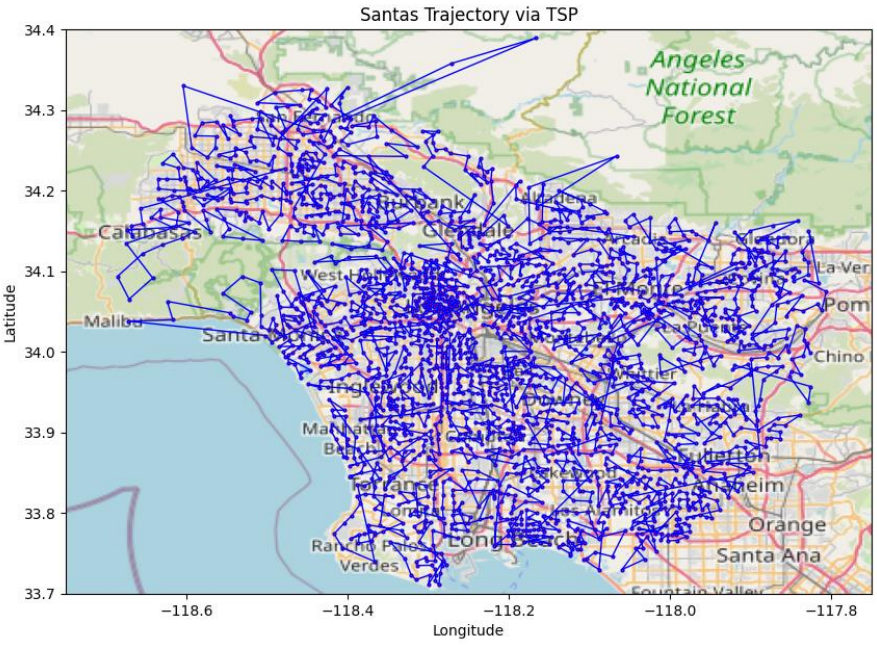


Fig.3 Santas's Trajectory using TSP

To further check if our results were intuitive, we plotted the first 5 nodes of the trajectory and can be seen in Fig. 4

We see that Santa starts near Alhambra and then proceeds to make his way towards to the Topanga mountains from which he then proceeds to visit all the nodes along the cost. While the actual route might involve revisiting certain edges, since during the calculation there are edges that are not present in the MST which are present in the original graph from which the MST, however the results do make intuitive sense since in Fig, we do not see loops.

Fig.4 First 5 Coordinates of Santa's TSP

Question 14

In this experiment, we were asked to generate a road mesh network from the coordinates of the nodes present in the graph using Delaunay's Triangulation algorithm. Delaunay Triangulation Algorithm operates on a discrete set of points and returns a set of triangles such that no point in the set of points lies within the circumference of any triangle, or in other words. All nodes becomes the vertices of a triangle and the edges between are the proposed road maps, which is a practical way of realising the road maps since most road maps or meshes follow a geometric method which the triangulation algorithm can emulate. The output of the triangulation algorithm can be seen in Fig.5 and the actual graph that was then generated from the proposed network can be seen in Fig. 6

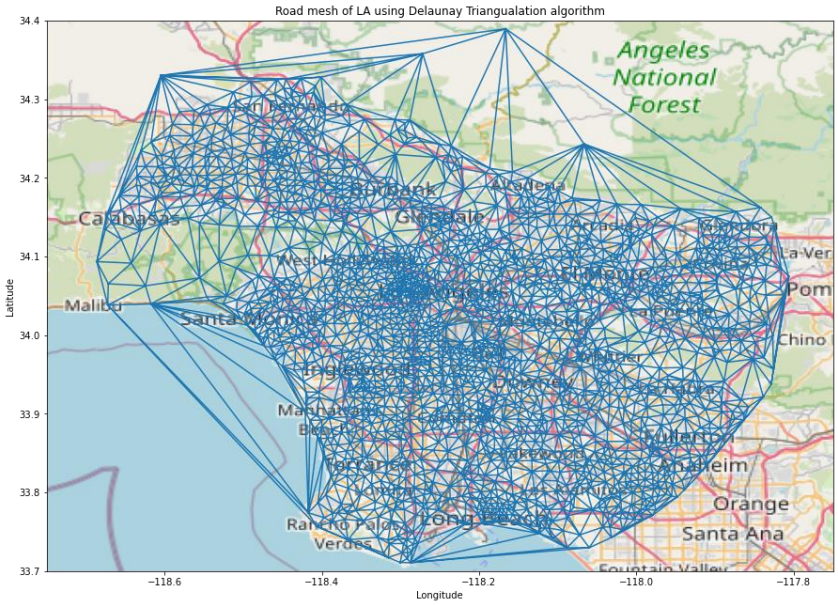


Fig.5 Proposed Road Mesh using Triangulation Algorithm

When comparing the two, the first few differences you realise are that the graph GΔ has fewer edges, albeit fewer fake edges such as roads directly connecting Malibu to Rancho Verdes over the ocean and roads connecting the Topanga mountains. This is because of how the triangulation algorithm tries to minimise narrow triangles and makes sure that each point does not lie in the circumference of a triangle. While this a good algorithm for visualising the road network, care had to be exercised to remove unrealistic edges.

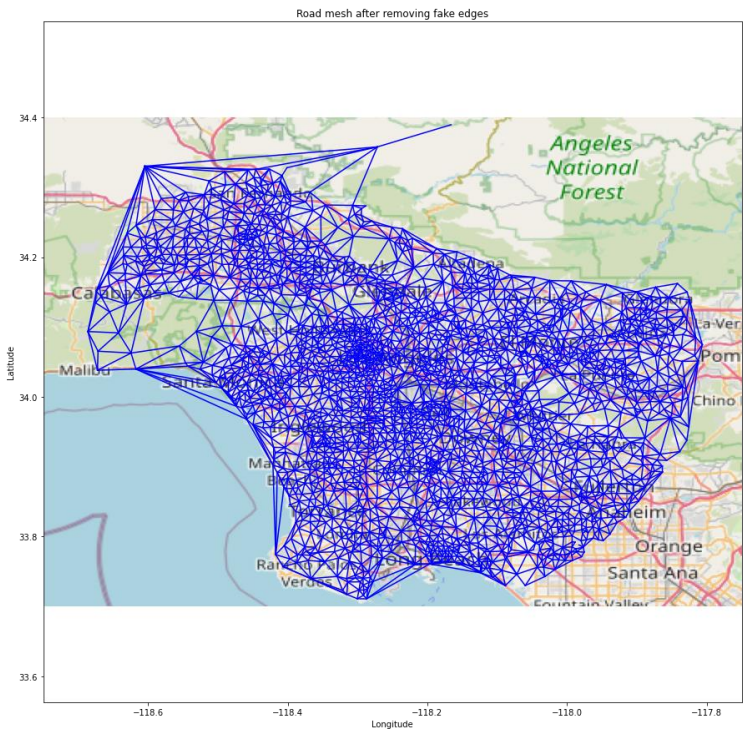


Fig.6 Visualisation of Graph after removing fake edges

Question 15

The derivation for this goes as follows are mentioned in equations (8-14).

- $Total\ distance = \frac{(Speed\ of\ Car)\ x\ (Mean\ Travel\ Time)}{60\ x\ 60}$
- $Safe\ Distance\ = 0.003\ +\ (Speed\ of\ Car\ )x\ \frac{2}{60\ x\ 60}$
- $Number\ of\ Cars\ in\ a\ lane\ = \frac{Total\ distance}{Safe\ Distance}$
- $Number\ of\ Cars\ in\ both\ lanes\ =\ 2\ x\ (Number\ of\ Cars\ in\ a\ lane)$
- $TrafficFlow(cars/hour) = \frac{60x\ 60\ x\ (Number\ of\ Cars\ in\ both\ lanes)}{Mean\ Travel\ Time}$
- $Simplified\ Derivation\ = \frac{3600\ x\ (Speed\ of\ Car)}{5.4\ +\ (Speed\ of\ Car)}$
- $Speed\ of\ Car\ = \frac{69}{(Mean\ Travel\ Time)} x\ \|Coordinates_1 - Coordinates_2\|_2$
- $Coordinates_{1\ or\ 2}\ = \begin{vmatrix} Latitude\ of\ Point \\ Longitude\ of\ Point \end{vmatrix}$

The *Coordinates* variable stores the latitude and the longitude and the variable *mean travel time* has to be divided by 3,600 since it is originally in seconds and we need to convert it to hours. We also need to multiply the norm 2 distance between the points by 69 to convert it from degrees used in latitude and longitude to miles.

Question 16

In this question, we were asked to calculate the maximum number of cars that can commute per hour from Malibu to Long Beach and calculate the number of edge-disjoint paths between the two spots. The first thing we had to do was figure out the node associated with the coordinates given in the project. By iteratively going through the graph and finding the coordinates which had the least Euclidean distance to the points. The inbuilt maxflow function as used in the graph where the source was set as the vertex which was closest to Malibu and the destination was set to the vertex closest to Long Beach.

Using the derivation from the previous question and using the inbuilt maxflow question, the number of cars that can travel from Malibu to Long Beach came to be 7,579 cars. The number of disjoint paths between the two nodes is 6. The number of disjoint paths between the two can be visualised from the graphs which are shown in Fig.7 and Fig.8 respectively.

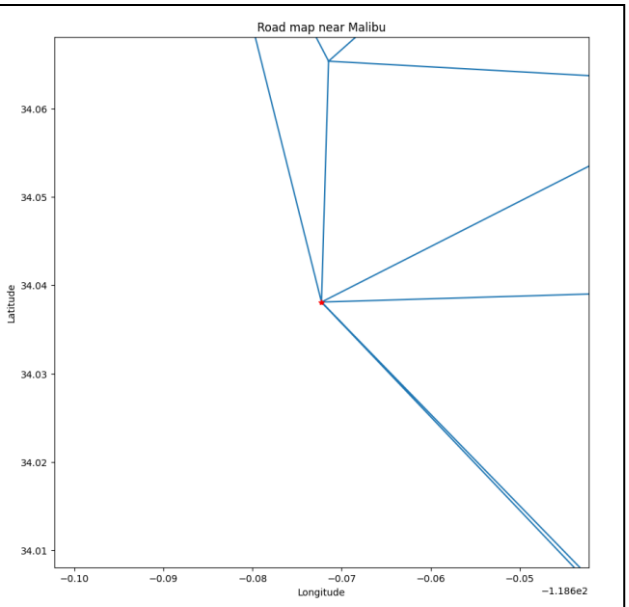


Fig.7 Roadmap near Malibu

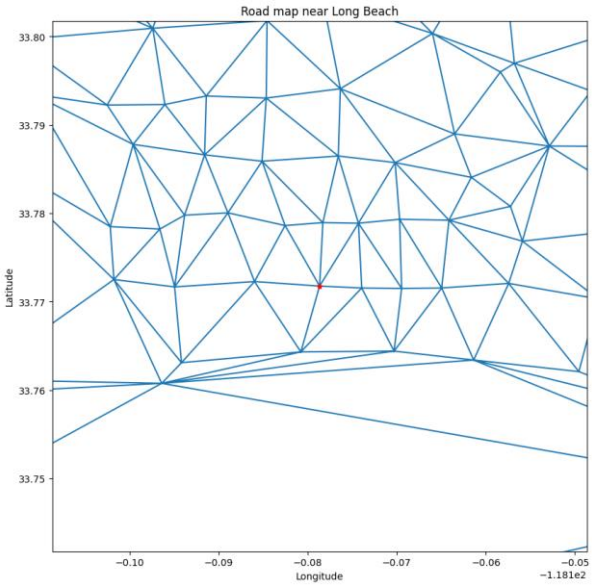


Fig.8 Roadmap near Long Beach

Upon close inspection, you can see that the degree of the Malibu node is 6 and the degree of the Long Beach vertex is also 6, therefore by taking the minimum of the two, the number of edge-disjoint paths from Malibu to Long beach came out to be 6.

Question 17

In this question we were asked to apply a thresholding method on the graph, wherein edges with a mean travel time greater than *800s*, this was done to remove edges which were traversing across the ocean and over the Topanga mountains, while these transit times between vertices are true, the road map suggested by Delaunay's triangulation makes these road maps unrealistic, therefore in this question such edges were trimmed and the results for this can be seen in Fig. 9 and Fig.10 .

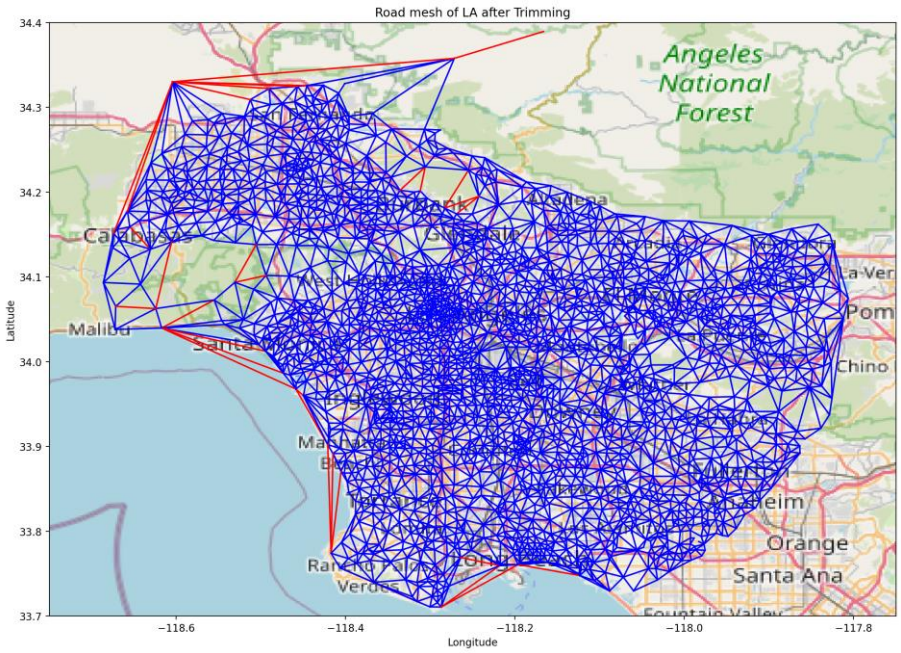


Fig.9 Roadmap after trimming

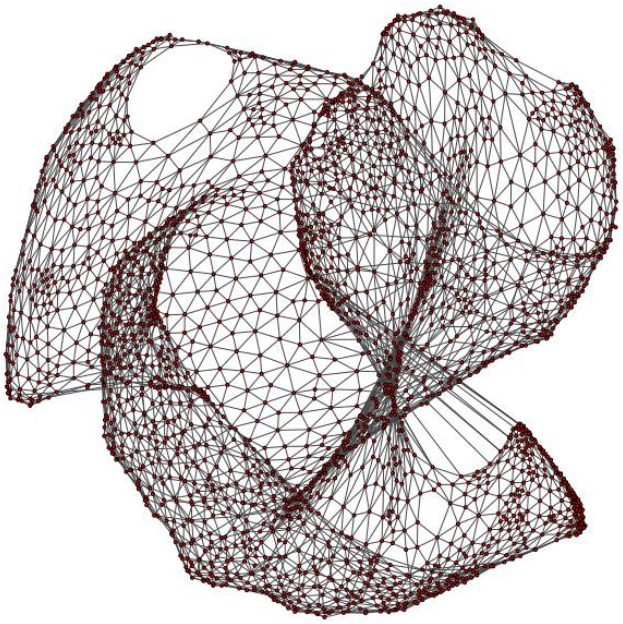


Fig.10 Roadmap after trimming

Upon close inspection of the trimmed graph superimposed on the LA map, we can see that the road mesh is now more realistic with no edges/roads crossing over the ocean or roads traversing over the mountains, and there are fewer long roads which makes intuitive sense as well, since there are hardly any roads that traverse from one part of the city to another by skipping over the others, road are local and connect to other roads which can be seen from the figure as well. Therefore, the thresholding did work.

Question 18

In this question, the goal was to recalculate the number of edge-disjoint paths between Malibu and Long Beach and calculate the max flow between Malibu and Long Beach. The number of edge-disjoint paths between Malibu and Long Beach came to 4 after pruning the graph which can be visualised as shown in Fig.11 and Fig. 12

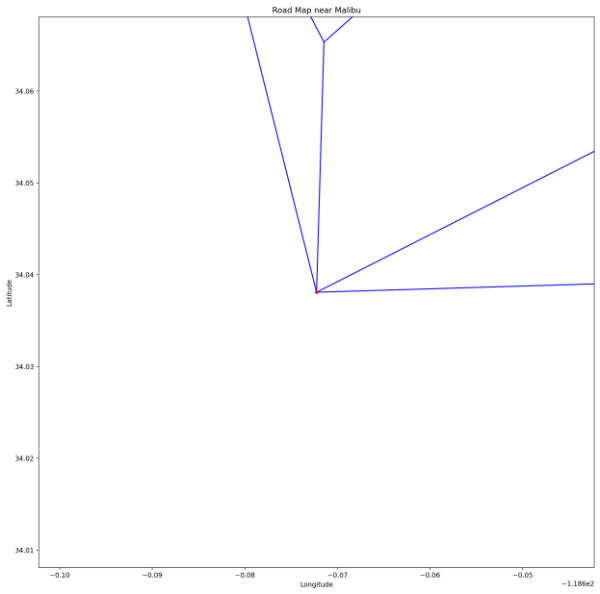


Fig.11 Roadmap near Malibu

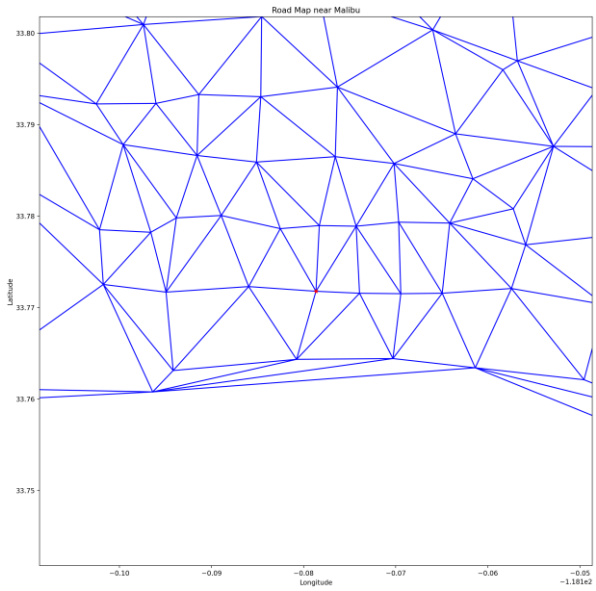


Fig.12 Roadmap near Long Beach

From Fig.11 we can see that two edges have been trimmed from the Malibu Vertex, while none were trimmed near Long Beach. The degree of the Malibu Vertex is 4 and the degree of the Long Beach vertex is 6, thus the number of edge disjoint paths between the two is 4 because the minimum degree between the two is 4.

When recalculating the maxflow, the maximum number of cars that can traverse from Malibu to Long Beach came out to 7579, which is the same as the previous max flow value, which highlights that the pruned edges were not the bottleneck in graph, and because max-flow is calculated by trying to find the min-cut between Malibu and Long Beach and the trimmed edges probably didn't have the least weights therefore were never part of the min-cut therefore even after trimming it, the max-flow stayed the same.

Question 19

In this question, the goal was to reduce the extra travelling distance, by building 20 new roads where the extra travelling distance is measured as the difference between the shortest distance actually traversed and the straight-line Euclidean distance between two points. To speed up computations, the Euclidean distance between all points and the shortest path distance were pre-computed instead of being computed *on-the-fly*.

The complexity of calculating the Euclidean Distance is $O(|V|^2)$ and the complexity of calculating the shortest distance traversed is $O(s\times |E| \log|V|+|V|)$ since the shortest path are found by taking a vertex and finding its shortest path distance with every other node, this is efficient since this allows for caching values and reusing them since in the process of finding the path to a longer node, the values for smaller nodes can be found. The value *s* represents the number of source vertex that are queried, which in our case is also $|V|$. Therefore, the final complexity is $O(|V|^2+|V| \times |E| \log|V|+|V|)$ and in our Big O notation, we only need to account for the worst case our complexity can be simplified to $O(|V|^2+|V| \times |E| \log|V|)$, while this can simplified further, it has been left at this case since $|E|>>|V|$, both terms have been left since at different graph sizes, either $|E|$ or $|V|$ will control the complexity.

The Top 20 pairs with the highest extra distance are reported in Table 2 whose format is (source, destination) and can be visualised in Fig.13

(2419,386)	(2413,1783)	(2419,383)	(2419,392)	(2419,1902)
(2419,2140)	(2419,2158)	(2416,1860)	(2419,382)	(2419,391)
(2419,1901)	(2419,1904)	(2419,2163)	(2416,1783)	1860,1699)
(2419,381)	(2419,390)	(2419,45)	(2419,1906)	(1783,1699(

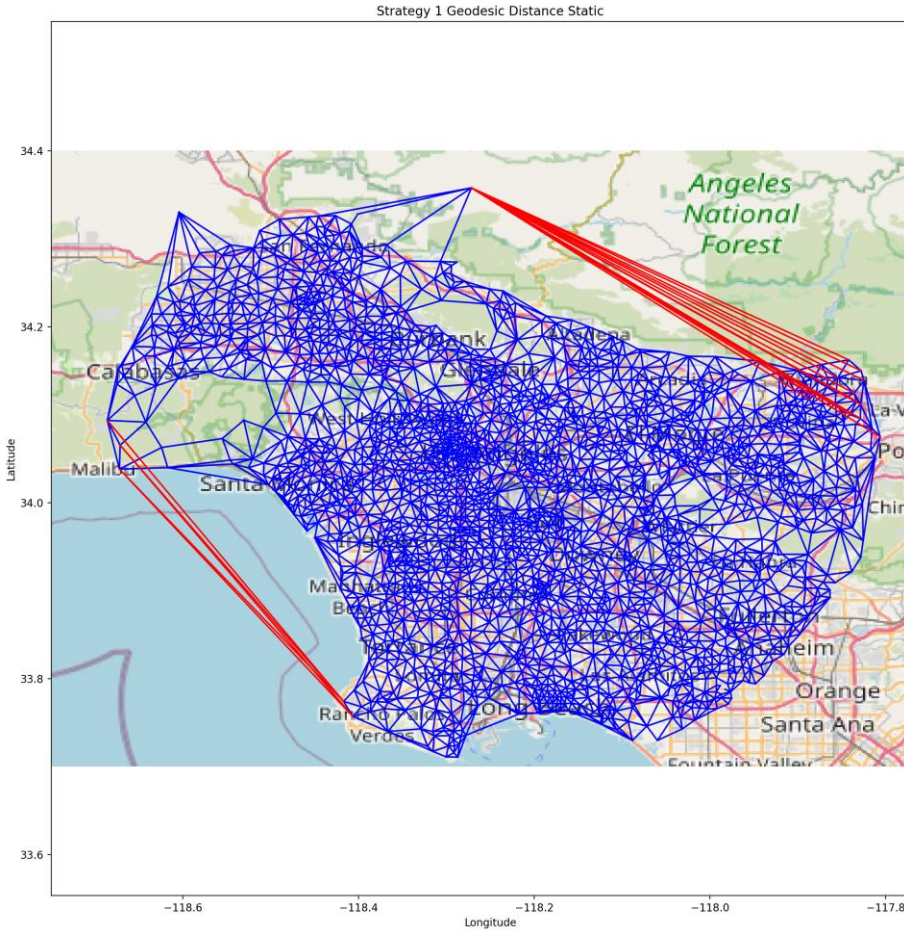


Fig.13 Suggested edges to add with Strategy 1

Question 20

In this question, the goal was to reduce the extra travelling distance , by building 20 new roads and weighting them to account for the fact that some roads are more used than others, the priority of the roads are done by using a random number generator. To speed up computations, the Euclidean distance between all points and the shortest path distance were pre-computed instead of being computed *on-the-fly*.

The complexity of calculating the Euclidean Distance is $O(|V|^2)$ and the complexity of calculating the shortest distance traversed is $O(s\times |E| \log|V|+|V|)$ since the shortest path are found by taking a vertex and finding its shortest path distance with every other node, this is efficient since this allows for caching values and reusing them since in the process of finding the path to a longer node, the values for smaller nodes can be found. The value *s* represents the number of source vertex that are queried, which in our case is also $|V|$. Therefore, the final complexity is $O(|V|^2+|V| \times |E| \log|V|+|V|)$ and in our Big O notation, we only need to account for the worst case our complexity can be simplified to $O(|V|^2+|V| \times |E| \log|V|)$, while this can simplified further, it has been left at this case since $|E|>>|V|$, both terms have been left since at different graph sizes, either $|E|$ or $|V|$ will control the complexity.

The Top 20 pairs with the highest weighted extra distance are reported in Table 3 The format of Table 3 is (source, destination) . and can be visualised in Fig.14

(2413,1783)	(2419,383)	(2419,2161)	(2419,2164)	(2419,2057)
(1511,979)	(1510,989)	(2419,49)	(1783,430)	(2413,1860)
(2472,2419)	(2419,2193)	(2463,2419)	(2419,2141)	(2419,1906)
(2419,231)	(2419,2156)	(2419,69)	(2419,66)	(2419,2052)

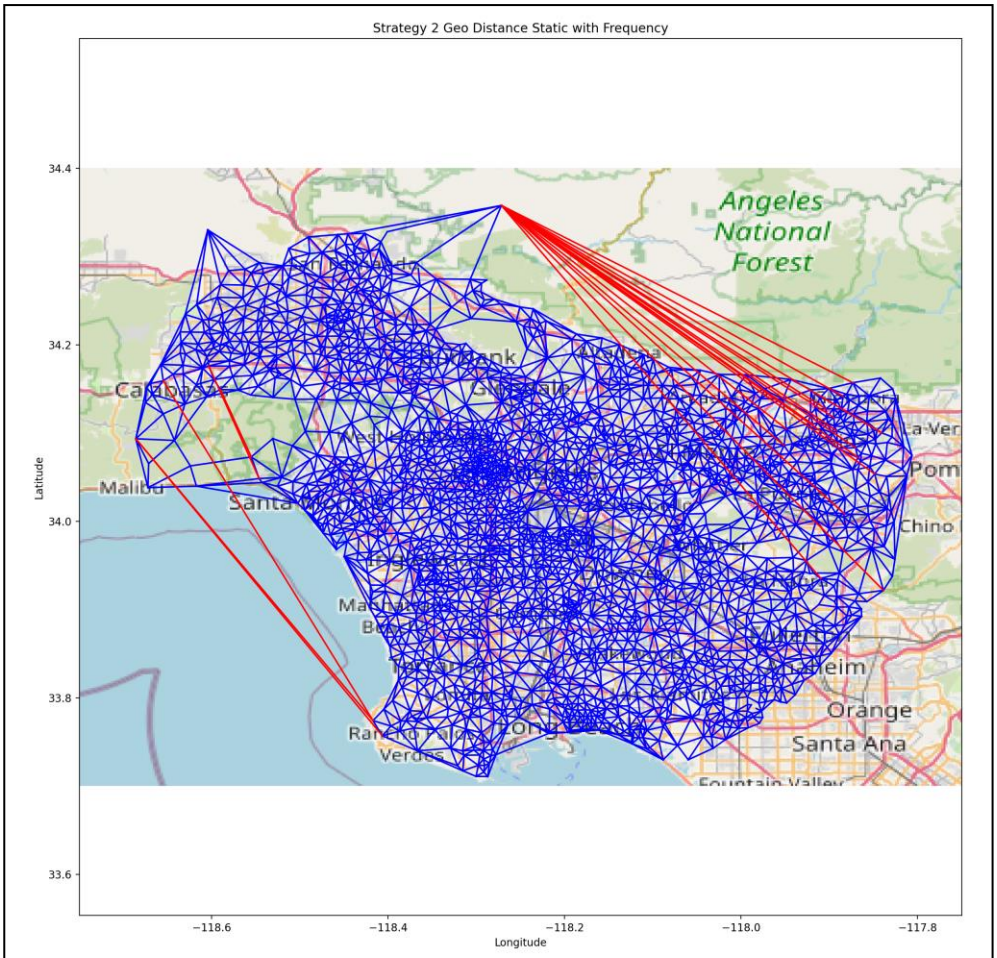


Fig.14 Suggested edges to add with Strategy 2

Question 21

In this question, the goal was to reduce the extra travelling distance, by building 20 new roads one at a time, and recalculating the Euclidean distances and the extra distance after building each new road, this is an iterative strategy.

O ($|V|^2+|V| \times |E| \log|V|+|V|$) is the complexity of running a static strategy and because we are calling this strategy k times, our complexity then becomes O ($k \times (|V|^2+|V| \times |E| \log|V|+|V|)$) and similar to the logic we've applied last time, since in Big O notation we want to only account for the worst case, or the terms that grow the most, we can simplify this to O ($k \times (|V|^2+|V| \times |E| \log|V|)$). Since based on whether we have a large graph or small graph, the either $|V|$ or $|E|$ will dominate the run-time.

The Top 20 pairs when each road is built one at a time is show in Table 4 The format of Table 4 is (source, destination). and the results can be visualised in Fig.15 .

(2419,2247)	(2417,1783)	(689,121)	(2419,285)	(1679,356)
(1510,986)	(2419,1956)	(2416,1783)	(2040,144)	(2637,2416)
(2619,2414)	(1781,1699)	(2416,2402)	(2619,144)	(2419,2242)
(1700,1003)	(2559,2414)	(2419,1717)	(1782,1700)	(1678,1005)

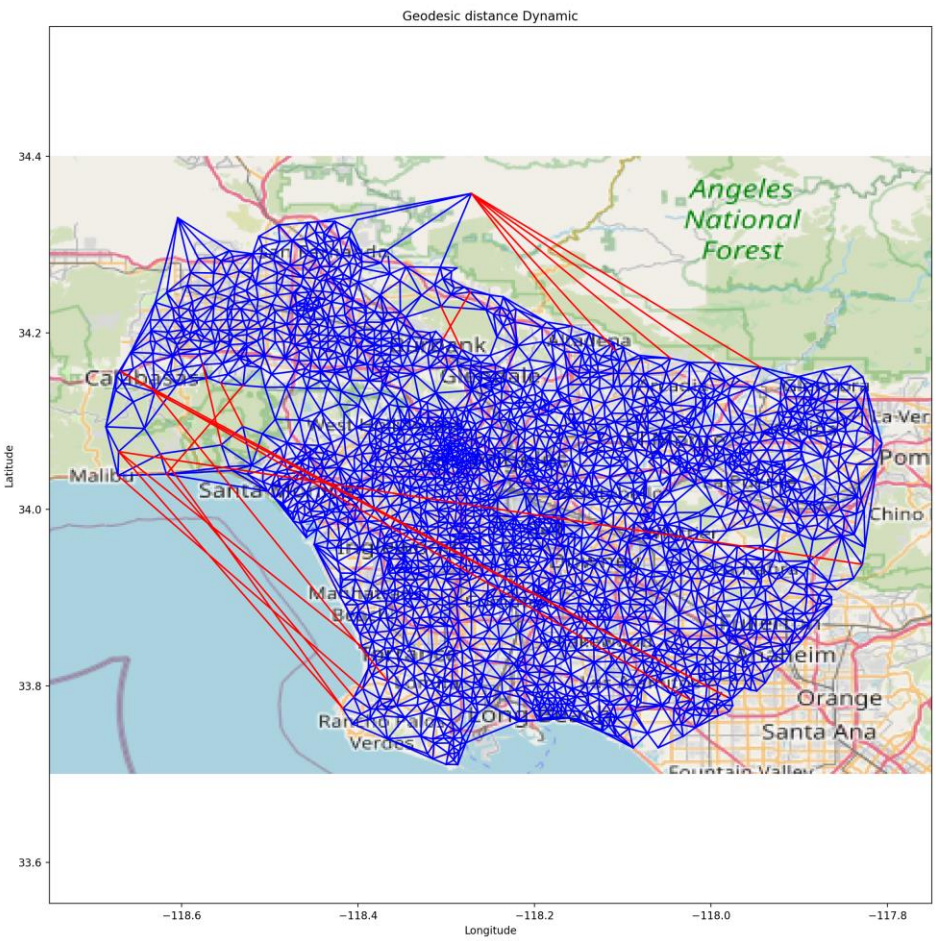


Fig.15 Suggested edges to add with Strategy 3

Question 22

In this question, the goal was to reduce the extra travelling time between two points. The extra travelling time is found by taking the difference between the travel time of the shortest path and the supposed travel time if there was a direct path. The supposed travel time is taken by calculating the speed which is the distance of the shortest path divided by the travel time of the shortest path. The Euclidean distance is then divided by the speed to give what we call the speed on the Euclidean track. Similar to Strategy 1, we precompute all the distances, instead of *on-the-fly,* moreover, for this strategy we need to calculate the Euclidean Distance, Shortest Path Distance and Shortest Travel Time separately and then operated on all three.

Since in this strategy, we need to calculate the Euclidean distance, Shortest Path Distance and Shortest time. The computational complexity is O ($|V|^2$+2× ($|V| \times |E| \log|V|$+$|V|$)) which can be simplified since in Big O notation, constants such as 2 really don't matter, since we are just trying to see how our algorithm grows with an addition of 1 node or edge. Therefore, the final complexity is O ($|V|^2$+$|V| \times |E| \log|V|$), because based on the graph size either $|V|$ or $|E|$ will dominate the run-time.

The Top 20 pairs when each road is built one at a time is show in Table 5 and can be visualised in Fig.16. The format of the Table is (source, destination).

(2413,1783)	(2416,1882)	(1678,989)	(1510,950)	(2416,1860)
(860,144)	(1510,989)	(1678,985)	(1783,144)	(2416,1783)
(1860,430)	(1860,1699)	(2416,1859)	(1783,430)	(2413,1860)
(1510,988)	(1678,984)	(1783,1699)	(1510,951)	(2416,1782)

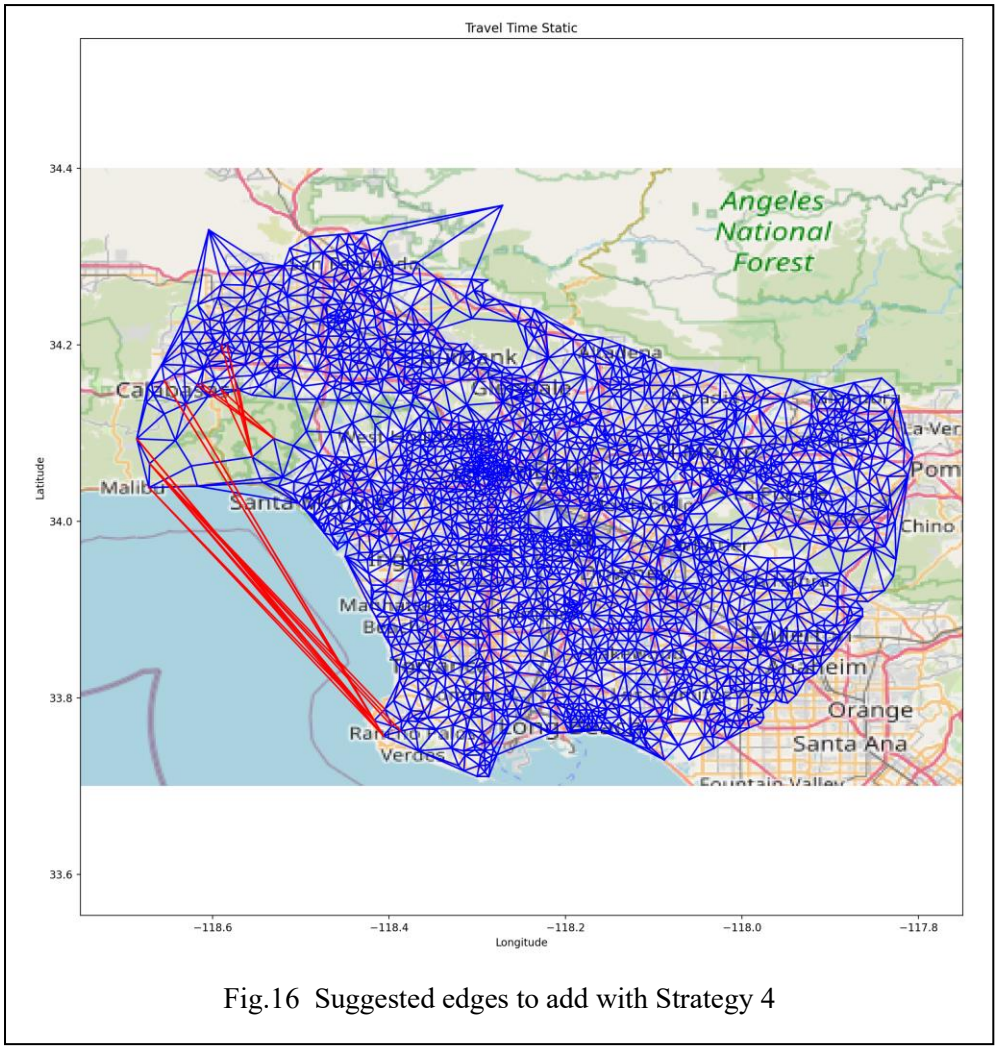


Fig.16 Suggested edges to add with Strategy 4

Question 23

In this experiment, the goal was to reduce the extra travel time by building 20 new roads iteratively instead of suggesting 20 new roads at once. At each step, the Euclidean distance, the shortest distance between two points and the shortest travel time between two points had to be recalculated and like previous implementations we pre-compute the values instead of calculating *on-the-fly* to save time and to exploit caching and reusing values.

Since in this strategy we need to recalculate the Euclidean Distance, Shortest Distance and Shortest travel time k times, the complexity of a single run is multiplied by k times. is O ($|V|^2$+2× ($|V| \times |E| \log|V|+|V|$)) is the complexity of a single run, and the final complexity after k runs is O ($k \times (|V|^2+2\times (|V| \times |E| \log|V|+|V|))$), and applying similar logics of discarding lower order terms and constants. We finally get O($k \times (|V|^2+|V| \times |E| \log|V|)$). Because based on the graph size, either the $|V|$ or $|E|$ will be the main term.

The Top 20 pairs when each road is built one at a time is show in Table 6, this can be visualised in Fig.17. The format of the table is (source, destination).

(2419,2247)	(2419,2262)	(2417,1783)	(2419,2140)	(2419,285)
(687,120)	(1679,121)	(1678,985)	(2419,1956)	(2419,2081)
(2419,49)	(2419,2273)	(1678,1003)	(2416,1783)	(2416,2402)
(1875,144)	(1510,988)	(2419,1717)	(12419,1964)	(1782,1700)

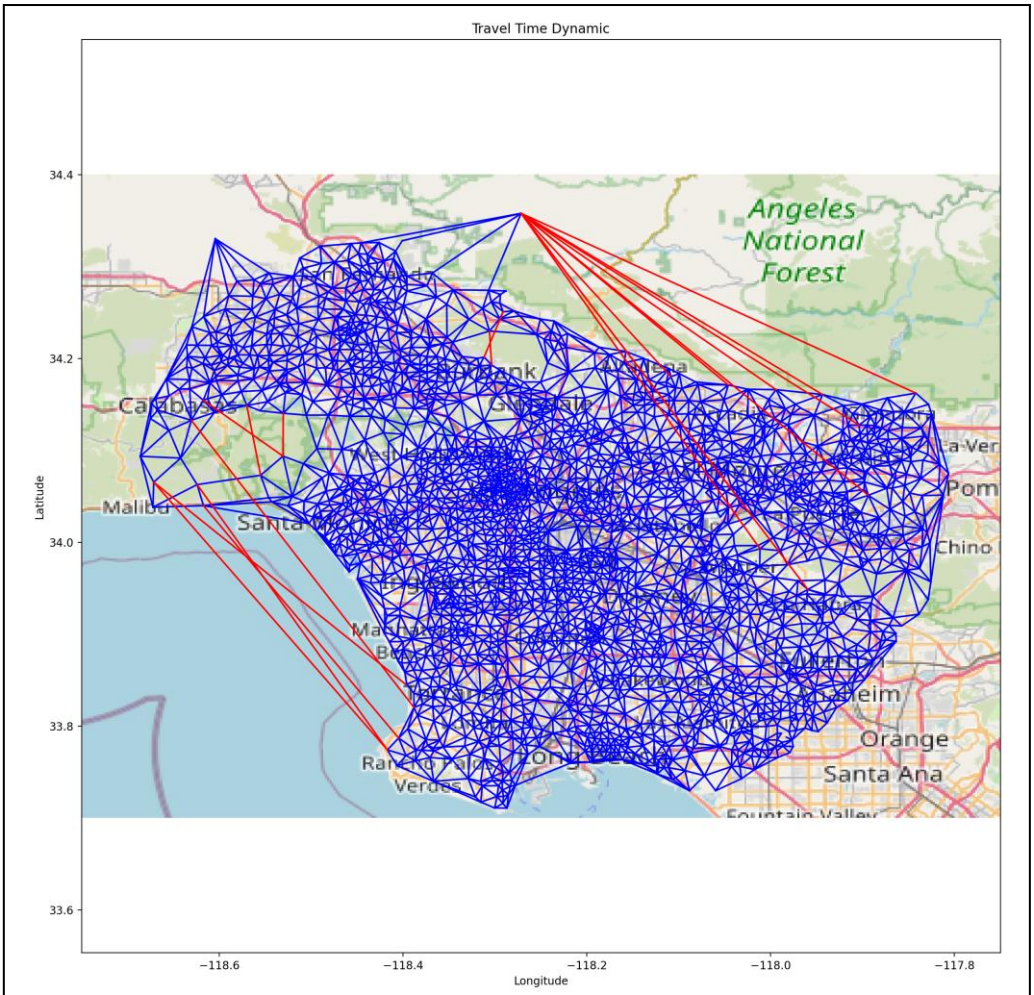


Fig.17 Suggested edges to add with Strategy 4

Question 24

In this question, we are asked to compare the strategies and analyse the results.

(a) 1 vs 2

In both strategy 1 and strategy 2, the goal is to reduce the minimum extra distance between places, by building twenty new roads and in strategy 2, the extra distance is weighted to account for the fact that some roads are more popular than others. Strategy 2 is practically a better strategy than Strategy 1 because there are some transits which are more popular than others and by reducing the travelling distance between the two points, it can alleviate the problems of many LA drivers.

Interestingly the results for both are similar with node (2419) being an import node in both, which highlights that there could be some correlation between routes which have the longest travelling distance and its importance. In Strategy 1, most of the suggested roads are between a vertex in the "Magic Mountain Wilderness" spot and places near "Pomona" and between places near "Malibu" and "Rancho Verdes". In Strategy 2, most of the suggested roads are between "Magic Mountain Wilderness" and places near "Pomona" and places between "Malibu" and "Rancho Verdes", with an extra edge between "Santa Monica" and "Calabasas" in Strategy 2. Some of the proposed edges shown in both strategies were present in the original graph before it was trimmed, which is interesting because the algorithm predicts that an edge should be there , while we trimmed because the transit was greater than 800, in reality this is probably a congested street which is popularly used to transit between points and after removing it, the transit between two points is even more, we probably trimmed a congested freeway or a highway which tends to cut across the city. In both strategies we see that the roads suggested make intuitive sense since the road would cut across the landscape since the current distance travelled is around the circumference of LA such as from "Rancho Verdes" to "Malibu" and "Pomona" to "Magic Mountain Wilderness". Such circuitous distances are always longer than straight-line distances. A simple intuition is that the algorithm is proposing a straight route to a circumference making a chord.

While both strategies optimise different criterion, Strategy 2 is a much better strategy for the real world since it accounts for popularity, otherwise in terms of edges generated both have the same performance since different constraints give different edges.

(b) 1 vs 3

In both Strategy 1 and Strategy 3, the goal is to reduce the extra travelling distance between points. While Strategy 1 is a batch processing sort of approach while the latter is iterative in nature, which takes into account the fact that after adding a road to the network, the shortest distances can be modified. While both are different forms of optimality, again from a practical point of view Strategy 3 is a better strategy since it accounts for the fact that after adding a road the distance to and from a node change and by iteratively adding one road at a time, the top 20 roads are a more realistic version of how the roads will actually be used in real life.

Interestingly, there is some similarity between Strategy 1 and 3 in terms of results, with the "Magic Mountain Wilderness" vertex, "Pomona" vertex, "Malibu" vertex and "Rancho Verdes" vertex still maintaining some relevance. However as seen in the results from Strategy

3, there are more roads that are spread out which does show that after adding a couple of roads to these vertices, the issue of transit to and from these places is solved since Strategy 3 also suggests building roads to and from Anaheim to Calabasas and two edges around the Burbank area.

Intuitively the roads suggested in Strategy 1 as explained before attempt to minimise the distances of people who travel on the circumference to reach a place. In Strategy 3 the goal of this is the same, but at different scales. After roads are added between "Magic Mountain Wilderness" and "Pomona" and "Rancho Verdes" and "Malibu". The longest extra travelling distances are still around the circumference such as from "Malibu" to "Calabasas" since travellers would need to get around the mountains to get to "Calabasas". The pattern we see across the suggested roads are that roads which cause people to travel around things tend to be the ones which the algorithm tends to address because these are the roads where the straight-line distance is quite small and the actual distance is huge.

Strategy 3 is a better strategy because it accounts for the fact that after a road is built, the dynamics of the graph change and by iteratively adding one road at a time, albeit at a higher time complexity, the mean extra travel distance is greatly reduced when compared to the mean reduction in extra travel distance by Strategy 1.

(c) 1 vs 4

Strategy 1 attempts to reduce the extra travelling distance between two points, while Strategy 4 attempts to reduce the extra travelling time between two points. In Strategy 1, a large number of suggested roads are in between "Magic Mountain Wilderness" and "Pomona" followed by "Rancho Verdes" and "Malibu". In Strategy 4, most of the suggested roads are in between "Rancho Verdes" and "Malibu", "Calabasas" and a small subset of roads suggested between "Santa Monica" and "Calabasas". Strategy 1 suggests roads which attempt to cut across the landscape to minimise the travel spent on the circumference. Strategy 4 suggests roads which attempts to decongest busy roads since it accounts for the travel time, and assumes the speed of the car on the new road will be the same as the speed it takes on the path yielding the shortest time. In terms of computational complexity, both yield similar upper bounds, while the results are different because of different optimisation criterion. While there seems to be some correlation between the suggested roads, possibly because more time spent commuting between destinations is also because of the distance needed to commute between the two. Upon closer inspection, we did also see that on these routes suggested, in google maps, a driver has to change between roads a lot which can hint at the presence of a bottleneck since the speed on each road is different.

In terms of which strategy is better, we believe that Strategy 4 is better because within a city, distances may not be as important as transit time since transit times hint at roads which need to be decongested.

(d) Static vs Dynamic

In Static strategies, the time complexity is O ($|V|^2+|V| \times |E| \log|V|$), and in Dynamic Strategies the time complexity is O ($k \times (|V|^2+|V| \times |E| \log|V|)$), where k represents the number of times a road is built iteratively. From a time, complexity point of view static strategies are better, but from a practical point of view Dynamic strategies are better. In term of which one is optimal, the Dynamic strategies are optimal because of the following reasons:

1. It accounts for the fact that after a road is added to the network, the shortest path distance/time changes
2. What static strategies suggest is general heuristics on which roads can decrease the total transit distance/time, because as we have seen in the results prior, there is no need to add 10 roads between “Magic Mountain Wilderness” and “Pomona” when 2 or 3 roads dynamically added do the job. Therefore, static strategies offer some heuristics, the actual relationship between adding a road and the metrics such as extra travel time or distance are better modelled by dynamic strategies.
3. The mean extra distance/time after applying a dynamic strategy is much lower than the same when a static strategy is applied, which highlights that the dynamic strategy can better tackle the problem of reducing the mean transit time/distance.

Question 24 (e)

After implementing Strategy 1 through 4, the Strategy we propose is an variation of Strategy 2, and 5 where the goal is to reduce the extra travel time between places while accounting for the fact that some roads are more popular than others and doing so in an iterative manner to account for the fact that after adding a road to the network, the transit times change because of the addition of a new road.

To help with the weighting process, we also propose that the type of road should also be accounted for such as freeways, highways or even multi-lane roads be given higher priority since these roads are more frequently used than smaller roads which might be more local and have lesser impact on easing congestion in traffic. Additional weighting can be used such as accounting for cost or even if it passes through restricted territory, these are mathematically difficult to model however. The travel time was taken as the optimisation variable since transit times are usually more important than transit distances especially when we are transiting within a city. If we were planning between cities or states, then distances matter since at that point it is a given that the transit will take time. Within a city, since the LA county represents a pseudo-circle, the distances of travelling from one end to another is similar therefore for near constant distances, our goal is to reduce the travelling time. After setting the optimisation variable, we've decided to weight it since some roads are more popular than others and by focussing our efforts on such popular roads the impact of a road is experienced by more drivers and the last decision of making it iterative in nature is because this accounts for the fact that after a road is added the transit times change for the network. Preliminary results after only weighting the transit times can be seen in Fig.

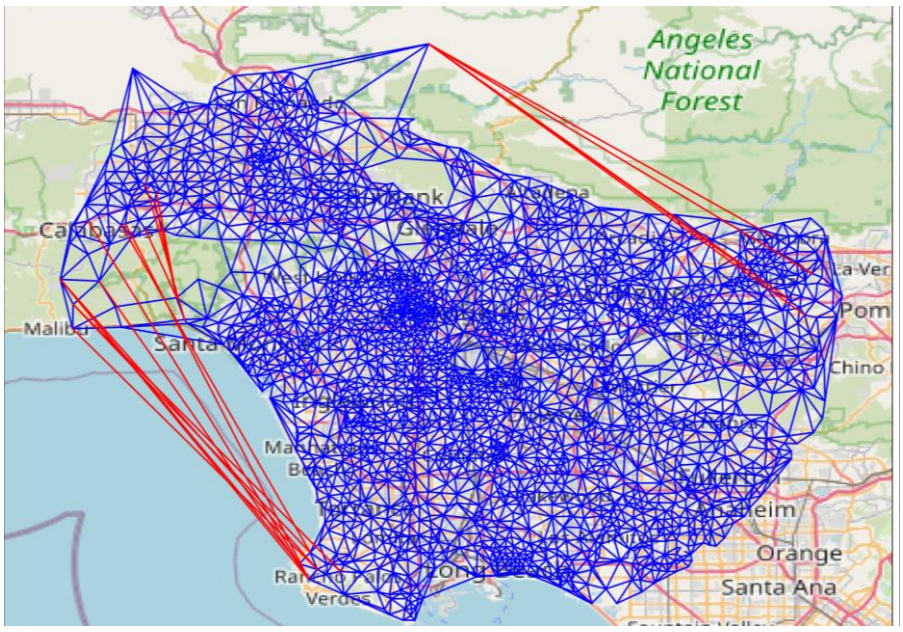


Fig.18 Preliminary edges to add with our custom strategy

Problem Motivation:

The starship (food delivery bots) team at UCLA wants to expand its business to areas beyond the main campus so effectively servicing Westwood as well. For achieving this they need to optimize their cost of operation ensuring that the food delivering bots use the most optimizing configuration of paths amongst all the bots in the fleet. They turn to us for solving and creating optimal strategies.

Detailed Definition:

Following are the key things to consider when solving this problem:

1. Total number of vehicles in the fleet?
2. What is the capacity of each vehicle to carry?
3. How many locations are present for delivery/pickup?
4. What is the quantity that has to be delivered (to a specific location) or picked up.

Based on these constraints we formalize this problem as a **Capacity Vehicle Routing Problem (CVRP)**. In CVRP, each vehicle has a fixed capacity and has to drop-off or pick-up a set amount of objects between different locations. This way of defining the problem is essentially an extension to the classical traveling salesman model (TSP). We add more specific constraints to the TSP to adhere to the new strict restrictions. CVRP falls in the category of the algorithms called **Vehicle Routing Problems (VRP).**

Formal method formulation:

To solve CVRP efficiently this problem definition is treated like a linear integer programming formulation such that the optimal path (route) decided has the least cost.

i : Location-1
j : Location-2
k : vehicle-ID

$x_{i,j,k}$: path between any i,j followed by vehicle-k.
$x_{i,j,k}$ = 1 if this is the most optimal path that should be followed by vehicle-k. Otherwise, 0.

d_{ij} : distance between the two set of coordinates.

CVRP objective can be defined as minimizing distance traveled by each vehicle on the optimal path while ensuring that the capacity restrictions for each vehicle are not exceeded as the demand is satisfied. This is denoted by the minimization equation below, q is the demand at each location.

$$Min \sum_{k=1}^{p} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ijk}$$

(Note: Only one vehicle can enter one node. The same vehicle should enter and leave the same node)

As CVRP is modeled like a linear programming project, it has the following constraints defined that can be used to automatically constrain the model.

Constrain 1: The same vehicle will always enter and leave the same node

$$\sum_{i=1}^{n} x_{ijk} = \sum_{i=1}^{n} x_{jik}$$

Constrain 2: All node should be serviced

$$\sum_{k=1}^{p} \sum_{i=1}^{n} x_{ijk} = 1 \qquad \forall j \in \{2, \ldots, n\}$$

Constrain 3: All vehicles are used

$$\sum_{j=2}^{n} x_{1jk} = 1$$

Constrain 4: The maximum ability of each vehicle is not exceeded

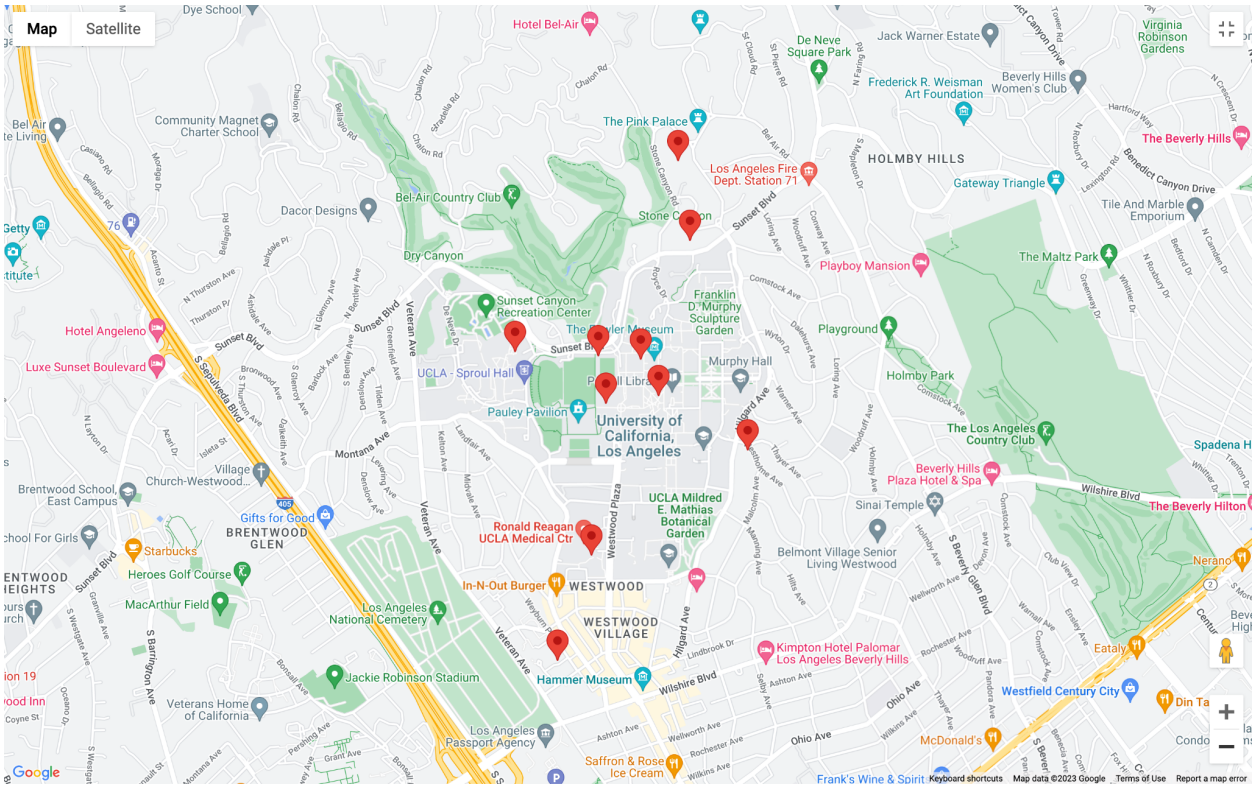
$$\sum_{i=1}^{n} \sum_{j=2}^{n} q_j x_{ijk} \leq Q$$

Methodology:

To extract data points we start with the center of UCLA longitude and latitude values to report the depot [Which in specific are: LAT = 34.071164LONG = -118.445546]. For simulation we look at 10 randomly sampled point which are nearby the starting depot (nearby cause we want to still stay in Westwood). The 10 large pins in Figure 1 are the 10 sampled points.

After extraction of the longitude and latitudes we randomly generate the demand at each of the stops.

In our sample problem the max number of available starships are 5 each with the capacity of 30 units of storage.



(Figure 1. 10 randomly sampled longitude and latitude points denoted with large red marked)

To find the optimal route of starship vehicles we now can solve using following two options:

1. Google-OR Tool for CVRP
2. PuLP library

We make use of the PuLP. This is a modeler for linear programming. In this we define a “solver” object to which different constraints can be added. Based on these constraints, this will automatically pick the most optimal strategy for the solution.

The optimal solution by the module says only two vehicles are needed. Please see Figure-2 and its description for better understanding of the paths taken.

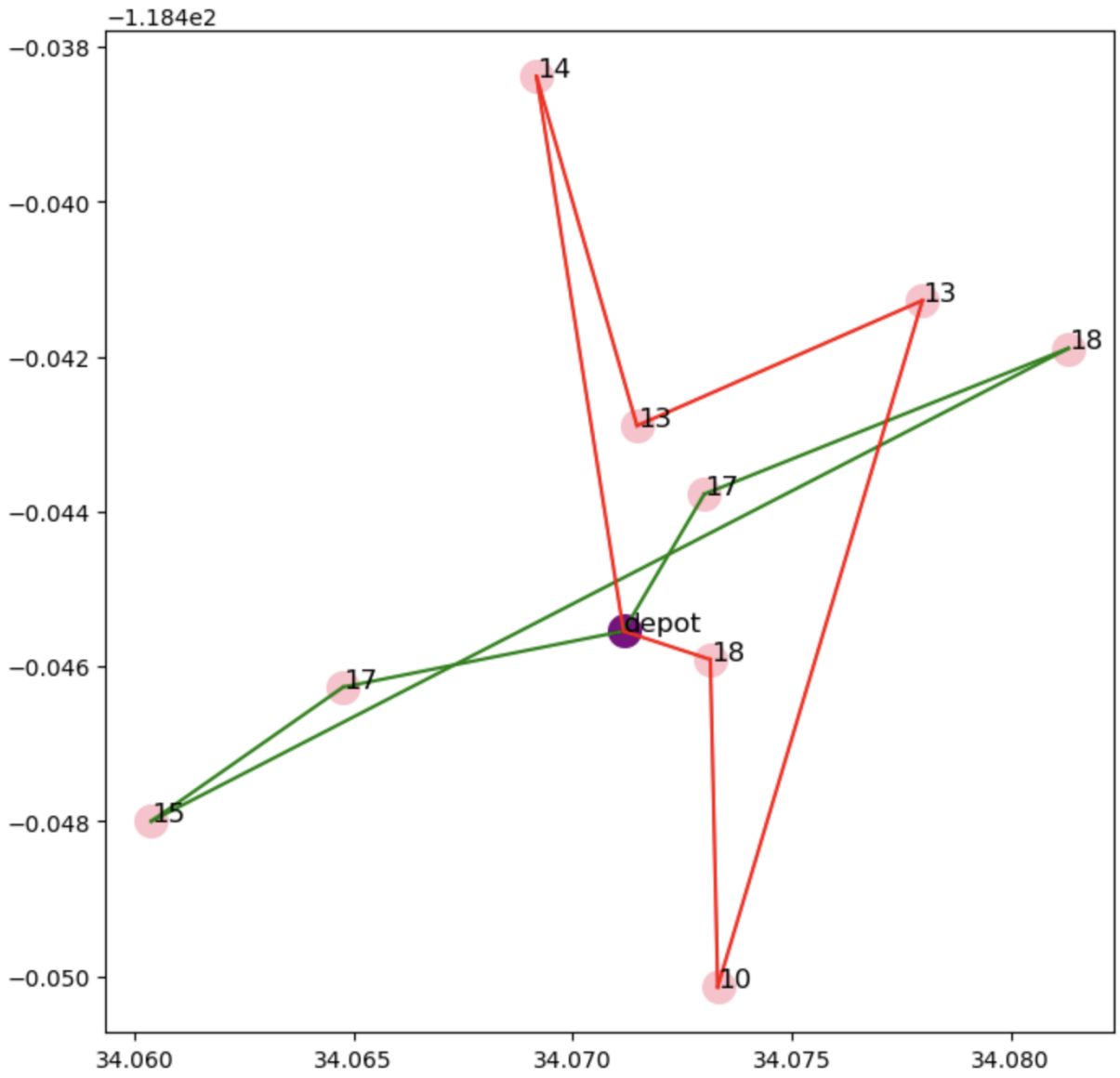


Figure 2. Optimal output path. Purple spot denotes the depot, the pink spots denote the delivery location / pick-up points. Here there are two vehicles used in the optimal solution. The path of vehicle-1 is denoted by green and the parth for vehicle-2 is denoted by red.

Sources:
https://how-to.aimms.com/Articles/332/332-Formulation-CVRP.html
https://medium.com/jdsc-tech-blog/capacitated-vehicle-routing-problem-cvrp-with-python-pulp-and-google-maps-api-5a42dbb594c0