Optimizing Route Navigation In London

Aditya Shankar Sarma Peri, Aakash Vaithyanathan, Nazanin Ghazitabatabai, Praket Kanaujia

I Introduction

A booming population and an increase in the number of automobiles on the road is causing an increase in road traffic worldwide, which is most notably seen in large/major cities and is severely worsened during rush hours. The lack of factoring in of this traffic data is a limitation of many apps that are designed to find the optimal route in order to save the time of the user. This issue is important as increasing traffic jams all over the world is not only leading to delayed times while travelling from one location to another but it is also leading to increasing petrol/diesel consumption which is causing an increase in air pollution and thus, worsening air conditions in many cities.

The two most-used applications are Waze and Google Maps which are able to accurately direct their users using crowd sourced information, which is the use of the internet to divide work between participants to reach a cumulative result.[**1**] The users share their information about map data, traffic jams, time, road accidents, speed, and police traps. The data is collected by the server and is used to alert other users to the conditions awaiting them through updating the routes and traffic jams.

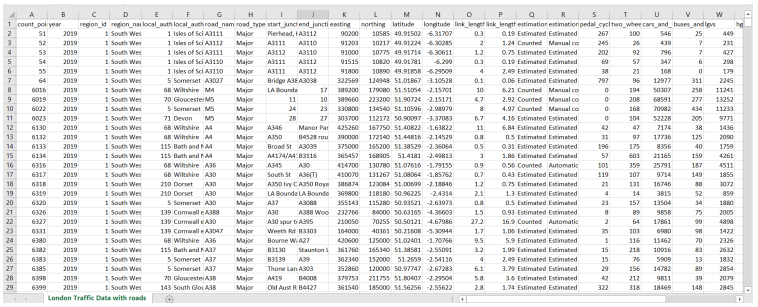
While we ourselves cannot use crowd sourced information and update the data in real time, we can however incorporate the aspect of traffic into our program in order to find the most optimal route based on whether the user wants an optimal route in terms of distance, time and most direct. Thus, our goal is to have an application that can assist the client by showing them the best route to take, factoring in whether the client wants the fastest route, the shortest in terms of length or the most direct. We will then visualize the best possible paths to take, for different means of transportation along with their travel time.

The question we want to analyze is :

How can we identify the different possible routes a user can take to reach a specific destination? Furthermore, how can this path chosen be optimized based on factors such as mode of transport and traffic volume?

II Dataset Description

The dataset used for this purpose is a CSV file extracted from data.gov.uk (The department of Transport). The CSV file contains the data pertaining to all the roads in the United Kingdom, from the year 2000 to 2019. A sample of the original dataset is attached below :



count_point_id	year	region_id	region_name	local_authority	local_authority_name	road_name	road_type	start_junction_ref	end_junction_ref	easting	northing	latitude	longitude	link_length_km	link_length_miles	estimation_method	estimation_method_detailed	pedal_cycles	two_wheeled_motor_vehicles	cars_and_taxis	buses_and_coaches	lgvs	hgvs
51	2019	1	South West	1	Isles of Scilly	A3111	Major	Pierhead, Hugh Town	A3112	90200	10585	49.91502	-6.31707	0.3	0.19	Estimated	Estimated	267	100	546	25	449	231
52	2019	1	South West	1	Isles of Scilly	A3112	Major	A3111	A3110	91203	10217	49.91224	-6.30285	2	1.24	Counted	Manual count	245	26	439	7	231	127
53	2019	1	South West	1	Isles of Scilly	A3111	Major	A3112	A3110	91000	10775	49.91714	-6.30611	1.2	0.75	Estimated	Estimated	202	92	796	7	427	231
54	2019	1	South West	1	Isles of Scilly	A3110	Major	A3111	A3112	91515	10820	49.91781	-6.299	0.3	0.19	Estimated	Estimated	69	57	347	6	298	179
55	2019	1	South West	1	Isles of Scilly	A3110	Major	A3111	A3112	91800	10890	49.91858	-6.29509	4	2.49	Estimated	Estimated	38	21	168	0	179	127
64	2019	1	South West	5	Somerset	A3027	Major	Bridge A38	A3038	322569	124948	51.01867	-3.10528	0.1	0.06	Estimated	Estimated	797	96	12977	311	2245	1127
6016	2019	1	South West	68	Wiltshire	M4	Major	LA Boundary	17	389200	179080	51.51054	-2.15701	10	6.21	Counted	Manual count	0	194	50307	258	11241	3127
6019	2019	1	South West	70	Gloucestershire	M5	Major	11	10	389660	223200	51.90724	-2.15171	4.7	2.92	Counted	Manual count	0	208	68591	277	13252	3127
6022	2019	1	South West	5	Somerset	M5	Major	24	23	330800	134540	51.10596	-2.98979	8	4.97	Counted	Manual count	0	168	70982	434	11233	3127
6023	2019	1	South West	71	Devon	M5	Major	28	27	303700	112172	50.90097	-3.37083	6.7	4.16	Estimated	Estimated	0	104	52228	205	9771	3127
6130	2019	1	South West	68	Wiltshire	A4	Major	A346	Manor Park	425260	167750	51.40822	-1.63822	11	6.84	Estimated	Estimated	42	47	7174	38	1436	1127
6132	2019	1	South West	68	Wiltshire	A4	Major	A350	B4528 roundabout	390000	172140	51.44816	-2.14529	0.8	0.5	Estimated	Estimated	31	97	17736	125	2090	1127
6133	2019	1	South West	115	Bath and North East Somerset	A4	Major	Broad St	A3039	375000	165200	51.38529	-2.36064	0.5	0.31	Estimated	Estimated	196	175	8356	40	1759	1127
6134	2019	1	South West	115	Bath and North East Somerset	A4	Major	A4174/A4	B3116	365457	168905	51.4181	-2.49813	3	1.86	Estimated	Estimated	57	603	21165	159	4261	1127
6316	2019	1	South West	68	Wiltshire	A36	Major	A345	A30	414700	130780	51.07616	-1.79155	0.9	0.56	Counted	Automatic	101	359	25791	187	4511	1127
6317	2019	1	South West	68	Wiltshire	A30	Major	South St	A36(T)	410070	131267	51.08064	-1.85762	0.7	0.43	Estimated	Estimated	119	107	9714	149	1855	1127
6318	2019	1	South West	210	Dorset	A30	Major	A350 by C	A350 Roye	386874	123084	51.00699	-2.18846	1.2	0.75	Estimated	Estimated	21	131	16746	88	3072	1127
6319	2019	1	South West	210	Dorset	A30	Major	LA Boundary	LA Boundary	369800	118180	50.96225	-2.4314	2.1	1.3	Estimated	Estimated	4	14	3815	52	859	1127
6320	2019	1	South West	5	Somerset	A30	Major	A37	A3088	355143	115280	50.93521	-2.63973	0.8	0.5	Estimated	Estimated	23	157	13504	34	1880	1127
6326	2019	1	South West	139	Cornwall	A388	Major	A30	A388 Woo	232766	84000	50.63165	-4.36603	1.5	0.93	Estimated	Estimated	8	89	9858	75	2005	1127
6327	2019	1	South West	139	Cornwall	A30	Major	A30 spur r	A395	210050	70255	50.50121	-4.67986	27.2	16.9	Counted	Automatic	2	64	17861	99	4898	1127
6331	2019	1	South West	139	Cornwall	A3047	Major	Weeth Rd	B3303	164000	40361	50.21608	-5.30944	1.7	1.06	Estimated	Estimated	35	103	6980	98	1422	1127
6380	2019	1	South West	68	Wiltshire	A36	Major	Bourne Wo	A27	420600	125000	51.02401	-1.70766	9.5	5.9	Estimated	Estimated	1	116	11462	70	2326	1127
6382	2019	1	South West	115	Bath and North East Somerset	A37	Major	B3130	Staunton L	361760	165340	51.38581	-2.55091	3.2	1.99	Estimated	Estimated	15	218	10016	83	2632	1127
6383	2019	1	South West	5	Somerset	A37	Major	B3139	A39	362340	152000	51.2659	-2.54116	4	2.49	Estimated	Estimated	15	76	5909	13	1832	1127
6385	2019	1	South West	5	Somerset	A37	Major	Thone Lane	A303	352860	120000	50.97747	-2.67283	6.1	3.79	Estimated	Estimated	29	156	14782	89	2854	1127
6398	2019	1	South West	70	Gloucestershire	A38	Major	A419	B4008	379753	211755	51.80407	-2.29504	5.8	3.6	Estimated	Estimated	42	212	9811	39	2079	1127
6399	2019	1	South West	143	South Gloucestershire	A38	Major	Old Aust R	B4427	361540	185000	51.56256	-2.55622	2.8	1.74	Estimated	Estimated	322	318	18469	148	2845	1127

London Traffic Data with roads

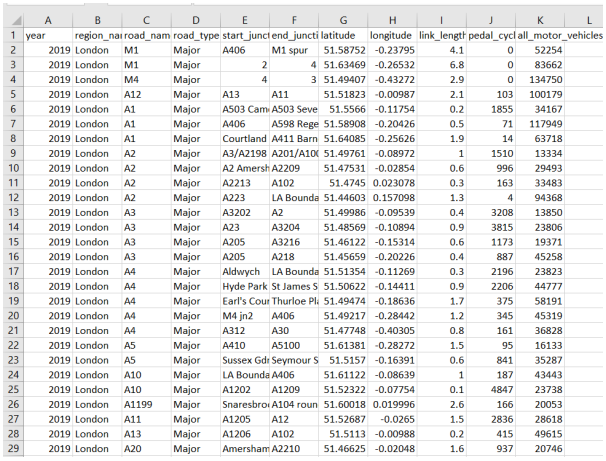
a. Some of the important data columns in the dataset are, start junction , end junction , latitude and longitude, pedal cycles and all motor vehicles.

b. The start junction column essentially refers to the name of the junction where the road begins and the end junction column refers to the corresponding end junction of the given road. Furthermore, link length refers to the distance between the start and end junctions, i.e the length of the road

c. The latitude and longitude together constitute the location of the start junction. Furthermore, pedal cycles and all motor vehicles respectively represent the average number of pedal cycles and motor vehicles passing through the road in 1 day.

   One thing to note here is that the data in this dataset is repetitive as it contains information pertaining to the road system for every single year , starting from 2000 (through to 2019). Furthermore, only the data columns mentioned above are relevant to our computations. Hence, we have filtered our dataset accordingly.

This is what the filtered dataset looks like :



	A	B	C	D	E	F	G	H	I	J	K	L
1	year	region_nai	road_nam	road_type	start_junct	end_juncti	latitude	longitude	link_length	pedal_cycl	all_motor_vehicles	
2	2019	London	M1	Major	A406	M1 spur	51.58752	-0.23795	4.1	0	52254	
3	2019	London	M1	Major	2	4	51.63469	-0.26532	6.8	0	83662	
4	2019	London	M4	Major	4	3	51.49407	-0.43272	2.9	0	134750	
5	2019	London	A12	Major	A13	A11	51.51823	-0.00987	2.1	103	100179	
6	2019	London	A1	Major	A503 Cam	A503 Seve	51.5566	-0.11754	0.2	1855	34167	
7	2019	London	A1	Major	A406	A598 Rege	51.58908	-0.20426	0.5	71	117949	
8	2019	London	A1	Major	Courtland	A411 Barn	51.64085	-0.25626	1.9	14	63718	
9	2019	London	A2	Major	A3/A2198	A201/A10(	51.49761	-0.08972	1	1510	13334	
10	2019	London	A2	Major	A2 Amersh	A2209	51.47531	-0.02854	0.6	996	29493	
11	2019	London	A2	Major	A2213	A102	51.4745	0.023078	0.3	163	33483	
12	2019	London	A2	Major	A223	LA Bounda	51.44603	0.157098	1.3	4	94368	
13	2019	London	A3	Major	A3202	A2	51.49986	-0.09539	0.4	3208	13850	
14	2019	London	A3	Major	A23	A3204	51.48569	-0.10894	0.9	3815	23806	
15	2019	London	A3	Major	A205	A3216	51.46122	-0.15314	0.6	1173	19371	
16	2019	London	A3	Major	A205	A218	51.45659	-0.20226	0.4	887	45258	
17	2019	London	A4	Major	Aldwych	LA Bounda	51.51354	-0.11269	0.3	2196	23823	
18	2019	London	A4	Major	Hyde Park	St James S	51.50622	-0.14411	0.9	2206	44777	
19	2019	London	A4	Major	Earl's Cour	Thurloe Pl	51.49474	-0.18636	1.7	375	58191	
20	2019	London	A4	Major	M4 jn2	A406	51.49217	-0.28442	1.2	345	45319	
21	2019	London	A4	Major	A312	A30	51.47748	-0.40305	0.8	161	36828	
22	2019	London	A5	Major	A410	A5100	51.61381	-0.28272	1.5	95	16133	
23	2019	London	A5	Major	Sussex Gdr	Seymour S	51.5157	-0.16391	0.6	841	35287	
24	2019	London	A10	Major	LA Bounda	A406	51.61122	-0.08639	1	187	43443	
25	2019	London	A10	Major	A1202	A1209	51.52322	-0.07754	0.1	4847	23738	
26	2019	London	A1199	Major	Snaresbro	A104 roun	51.60018	0.019996	2.6	166	20053	
27	2019	London	A11	Major	A1205	A12	51.52687	-0.0265	1.5	2836	28618	
28	2019	London	A13	Major	A1206	A102	51.5113	-0.00988	0.2	415	49615	
29	2019	London	A20	Major	Amersham	A2210	51.46625	-0.02048	1.6	937	20746	

III Computational Overview

In this project we have used the graph data structure to represent the entire system of roads, which has been implemented as its own data class. Each vertex represents a junction. The vertex object is named as *'Junction'* and the graph object is named as *'RoadSystem'*. If two junctions are adjacent to one another, then it implies that there is a road between them.

Each junction item has a *'name'* and a *'neighbours'* attribute. The *'name'* attribute is a string which represents the starting point of the junction. The *'neighbours'* attribute essentially is a dictionary which maps the neighbouring junctions to a tuple. This tuple consists of two elements, the first one refers distance between the two junctions (in km), and the second element is a list of integers, where the first integer represents the number of bicycles and the second integer represents the number of motor vehicles.

Note: Some of the methods of the *'RoadSystem'* data class are analogous to what we have learned in the class. The method *'add_junction'* adds a junction onto the RoadSystem and this is similar to the method *'add_vertex'* from the lectures. The method *'add_road'* adds a road between two neighbouring junctions and this is conceptually the same as the *'add_edge'* method from lectures. [**2**]

In order to load data from the CSV file into the graph object we have implemented the top level function *'load_graph()'*. Furthermore, we have implemented another top level function named *'load_dict()'*, this function Reads file_name and returns a dictionary where the keys are the start junctions and the values are a list of dictionaries. Each dictionary contains the keys road name, end junction, latitude, longitude, pedal cycle, distance, and all motor vehicles with their corresponding value. This function helps us retrieve data whenever need be without having to read the csv file over and over again.

Furthermore, for this project we have performed 5 different computations.

a. ***Multiple Paths***
The multiple_path method essentially asks the user to input the names of the start and end junctions. This method has another parameter 'n' which essentially refers to the number of distinct paths the user seeks from the program. Further, the multiple_path method generates n random paths using a helper method named find_path. The find_path method essentially helps in finding a random path between the start and end junctions and is conceptually very similar to the methods we have learned in lectures.

Note: In the event where the total number of distinct path between the two junctions is less than n, the functions outputs the maximum number of distinct paths possible.

b. ***Shortest Path***
It is always good to be aware of the different paths we could take while travelling from one place to another. However, in most cases we look for the path with the shortest distance. This is exactly what this method does. It helps the user find the shortest path in terms of distance between the two junctions of the user's choice. Our first way of approaching this problem was to find all the possible paths between the start and end junctions and then further computing the path which had the shortest distance. However, this was computationally very inefficient. Hence, in order to solve this problem we implemented our coded version of *Dikshitra's algorithm*. [**3**]

First it sets the distance of all other nodes from junction1_name to infinity. All the nodes are added to unvisited. Until junction2_name is in unvisited, the for loop will go through the vertices in unvisited and will find the vertex whose distance from junction1_name is less than the distance of between the first vertex in unvisited and junction1_name. Looping through the vertex's neighbor, if the distance between the vertex and the neighbor + distance vertex from junction1_name is less than the distance of

neighbor and junction1_name, then the distance of the neighbor will be set to that value. In the end we will append the vertices in the correct order to path and return it with its distance.

c. ***Direct Route***
Sometimes the shortest route may not be the most direct route between two junctions. For instance, there are three start junctions 'A', 'B', 'C'. Let's assume that 'A' is adjacent to both 'B' and 'C'. Let the distance between 'A' and 'B' be 4 km , and 'A' and 'C' be 1 km. Furthermore, let the distance between 'B' and 'C' be 2 km. Therefore, in principle, the shortest path between 'A' to 'B' would be 'A -> B -> C'. However, in most cases the user would find it more convenient to travel through the direct edge between 'A' and 'B' despite the fact that it isn't necessarily the shortest route . This is exactly what this algorithm computes. It computes the most direct path between the start and end junctions. In order to carry out this computation we have used our own coded version of the *Breadth First Search Algorithm*.[**4**] [**5**]

Using the BFS algorithm, start will be added to the queue list and while queue is not empty, path is set to the first element of queue, and node will be set to the last element of the path. Then looping over the neighbors of node, we will append them to the path and add the new path to queue. In each iteration if neighbor.name = end, then we will return the new path right away along with its distance.

d. ***Time Taken Between Junctions***
This computation calculates the time it takes to travel between two junctions , based on the mode of transport and based on the amount of traffic that is on the road. It uses a mathematical formula that relates traffic flow and traffic speed to traffic density [**6**]:

$$\text{traffic flow} = \frac{\text{number of daily vehicles}}{24}$$

$$\text{traffic density} = \frac{\frac{\text{traffic flow}}{60}}{\text{distance of the road}}$$

$$\text{speed} = \frac{\text{traffic flow}}{\text{traffic density}}$$

$$\text{time} = \frac{\text{distance of the road}}{\text{speed}}$$

e. ***Paths With Shortest Time***
The path_with_shortest_time calculates the shortest time it takes to reach from one location to another, taking traffic conditions into account, by calculating the time taken for multiple paths, that are given by the output of the multiple_path function. This is achieved through calling the time_taken_between_junctions method for each start and end junction for each path. The time taken and the paths are stored in lists and by taking the minimum of the list, we can find the path that is the fastest in terms of time. the result is outputted as a tuple which gives the path, the time taken and the distance of the path, in that order.

IV Visualizations

Based on the above described computations, there are 5 possible visualizations that the user can view representing the various relationships between the different road junctions in a interactive manner.

A. Visualizing Multiple Paths

For representing our data, we use the **folium library** [**7**].

The *visualize_multiple_path()* plots a map of the London region with the specified London coordinates for this map. The function extracts all the possible paths that the user has requested for with the help of the *multiple_path()* under the **computations.py** file.

With the help of the following helpers :

a. *helper_plot_markers() – for plotting the markers for each junction*

b. *helper_road_road_link() – for plotting a road link between every 2 connected junctions in a path*

the function updates the state of the **plot_map** variable to make a complete map.

Based on the inputs given by the user, if there exists a path between the specified junctions, a map will successfully be generated. If no such path exists based on the requirements specified by the user, a message will be displayed indicating to them that the plot could not be made.
To view the generated map, the user must open the *multiple_routes.html* file found under the **Plots_Generated** folder of the main project file.

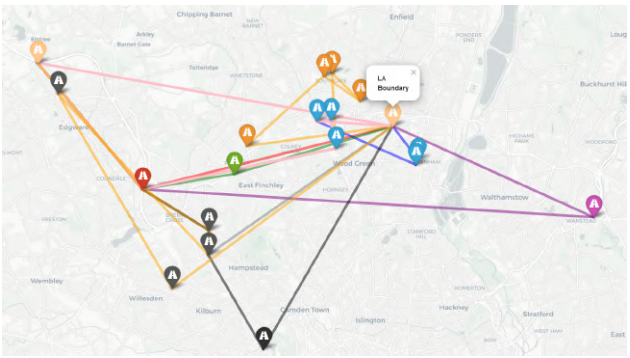
The function makes use of two main methods from the folium library:

1. ***folium.Marker()*** *– for plotting the junctions as markers.*

2. ***folium.Polyline()****– to plot a link between the 2 junctions in a path.*

Brief Discussion of Visualizing Multiple Paths

We want to visualize *based on the specified inputs, how many unique paths are drawn between the start and end junctions and how do these paths visually represent this connection.*



In the graph displayed above, the different colour markers with *road icons* represent the different individual junctions for all possible paths. The colour coded road links represent the path between the 2 roads. Each unique path is traced with a single colour to represent a complete path. Different unique coloured paths represent the unique paths generated.

This specific output is achieved by running the following method under **visualizations.py** :

visualize_multiple_path('LA Boundary', 'M1 spur', 10)

B. Visualizing Shortest Path

For representing our data, we use the **folium library**. The *visualize_shortest_path()* plots a map of the London region with the specified London coordinates for this map. The function extracts the shortest path that the user has requested for with the help of the *shortest_path()* under the **computations.py** file.

With the help of the following helpers:

a. *helper_plot_markers() – for plotting the markers for each junction*

b. *helper_plot_shortest_marker() – for plotting the markers specific for the shortest path.*

c. *helper_plot_shortest_road_links()–for plotting the road link between junctions in the shortest path*

d. *helper_road_road_link() – for plotting a road link between every 2 connected junctions in a path other than the shortest path.*

the function updates the state of the **plot_map** variable to make a complete map.

Based on the inputs given by the user for the start and end junctions, if there exists a shortest path between the specified junctions, a map will successfully be generated. If no such path exists based on the requirements specified by the user, a message will be displayed indicating to them that the plot could not be made.
To view the generated map, the user must open the *shortest_route.html* file found under the **Plots_Generated** folder of the main project file.

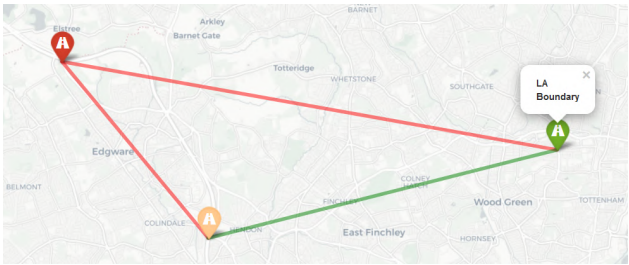
The function makes use of two main methods from the folium library :

1. ***folium.Marker()*** *– for plotting the junctions as markers.*

2. ***folium.Polyline()****– to plot a link between the 2 junctions in a path.*

Brief Discussion of Visualizing Shortest path

We want to visualize *based on the specified inputs, what is the shortest path that the user can obtain between the two junctions. Further, how can this path be visually represented.*



In the graph displayed above, the different colour markers with *road icons* represent the different individual junctions for all possible paths. The green colour coded road links represent the shortest path between the 2 inputted junctions whereas the red colour coded road link represents an alternate random route between the two junctions. The 2 paths are visually represented to show comparatively how the shortest path compares to any other path generated by the function between the 2 junctions. Each unique path is traced with a single colour to represent a complete path. Different unique coloured paths represent the unique paths generated as described above

This specific output is achieved by running the following method under **visualizations.py** :

visualize_shortest_path('A406', 'LA Boundary', 10)

C. Visualizing Direct Route

For representing our data, we use the **folium library**.

The *visualize_direct_path()* plots a map of the London region with the specified London coordinates for this map. The function extracts the most direct path that the user has requested for with the help of the *direct_route()* under the **computations.py** file.

With the help of the following helpers :

a. *helper_plot_markers() – for plotting the markers for each junction in the paths obtained other than the direct path*

b. *helper_plot_shortest_marker() – for plotting the markers specific for the junctions in direct path*

c. *helper_plot_road_link() – for plotting a road link between every 2 connected junctions in a path other than the direct path*

d. *helper_plot_shortest_road_links() – function for plotting the road link between the two junctions in the direct path*

the function updates the state of the **plot_map** variable to make a complete map.

Based on the inputs given by the user for the start and end junctions, if there exists a direct path between the specified junctions, a map will successfully be generated. If no such path exists based on the requirements specified by the user, a message will be displayed indicating to them that the plot could not be made.
To view the generated map, the user must open the *most_direct_route.html* file found under the **Plots_Generated** folder of the main project file.

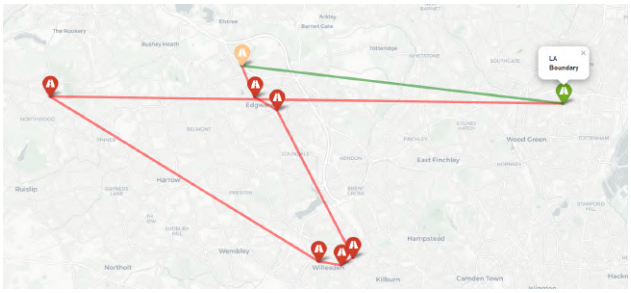
The function makes use of two main methods from the folium library :

1. ***folium.Marker()*** *– for plotting the junctions as markers.*

2. ***folium.Polyline()****– to plot a link between the 2 junctions in a path.*

Brief Discussion of Visualizing Direct Route

We want to visualize *based on the specified inputs, what is the most direct path that the user can obtain between the two junctions. Further, how can this path be visually represented.*



In the graph displayed above, the different colour markers with *road icons* represent the different individual junctions for all possible paths. The green colour coded road links represent the most direct path between the 2 inputted junctions whereas the red colour coded road link represents an alternate random route between the two junctions. The 2 paths are visually represented to show comparatively how the most direct path compares to any other path generated by the function between the 2 junctions. Each unique path is traced with a single colour to represent a complete path. Different unique coloured paths represent the unique paths generated as described above

This specific output is achieved by running the following method under **visualizations.py** :

visualize_direct_path('M1 spur', 'LA Boundary')

D. Visualizing Shortest Time Based Route

For representing our data, we use the **folium library**.

The *visualize_shortest_time_path()* plots a map of the London region with the specified London coordinates for this map. The function extracts the shortest time based path that the user has requested for with the help of the *multiple_path()* and *path_with_shortest_time()* under the **computations.py** file.

With the help of the following helpers :

a. *helper_plot_shortest_marker() – for plotting the markers for each junction in shortest time based path obtained*

b. *helper_plot_time_road_links() – for plotting a road link between every 2 connected junctions in the shortest time based path*

the function updates the state of the **plot_map** variable to make a complete map.

Based on the inputs given by the user for the start junction end junction, and the preferred mode of transportation (*either **cycle** or **vehicle***), if there exists a shortest time path between the specified junctions, a map will successfully be generated. If no such path exists based on the requirements specified by the user, a message will be displayed indicating to them that the plot could not be made.
To view the generated map, the user must open the *shortest_time_route.html* file found under the **Plots_Generated** folder of the main project file.

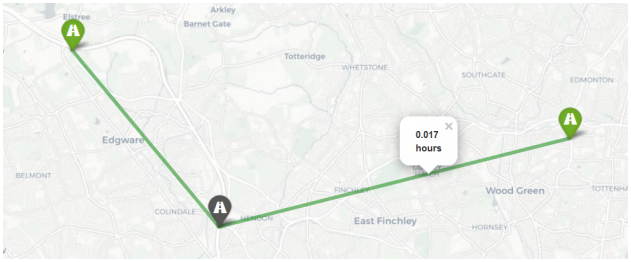
The function makes use of two main methods from the folium library :

1. ***folium.Marker()** – for plotting the junctions as markers*

2. ***folium.Polyline()**– to plot a link between the 2 junctions in a path.*

Brief Discussion of Visualizing Shortest Time Based Route

We want to visualize *based on the specified inputs, what is the shortest time based path that the user can obtain between the two junctions. Further, how can this path be visually represented.*



In the graph displayed above, the different colour markers with *road icons* represent the different individual junctions for the shortest time based route as requested by the user. The green colour coded road links represent this path between the 2 inputted junctions where the black marker represents the intermediate junction between the path. Further, upon clicking each junction, the user can obtain the name of the junction and by clicking the name of the junction, the user can obtain the time taken to travel between the two junctions as shown in the graph above.

This specific output is achieved by running the following method under **visualizations.py** :

visualize_shortest_time_path('LA Boundary', 'M1 spur', 'vehicle')

E. Visualizing Paths With Specific Stops

For representing our data, we use the **folium library**.

The *visualize_path_specific_stops()* plots a map of the London region with the specified London coordinates for this map. The function extracts all the paths between the given junctions and then extracts the path with specific path as requested by the user. The function performs this with the help of *multiple_path()* under the **computations.py** file.

With the help of the following helpers :

a. *helper_plot_specific_marker() – for plotting the markers for each junction in the path with specific requested stops.*
b. *helper_plot_shortest_road_links() – for plotting a road link between every 2 connected junctions in the specific stops path.*

the function updates the state of the **plot_map** variable to make a complete map.

Based on the inputs given by the user for the start junction end junction, and the preferred number of stops, if there exists a such a path between the specified junctions, a map will successfully be generated. If no such path exists based on the requirements specified by the user, a message will be displayed indicating to them that the plot could not be made.
To view the generated map, the user must open the *path_with_specific_stops.html* file found under the **Plots_Generated** folder of the main project file.

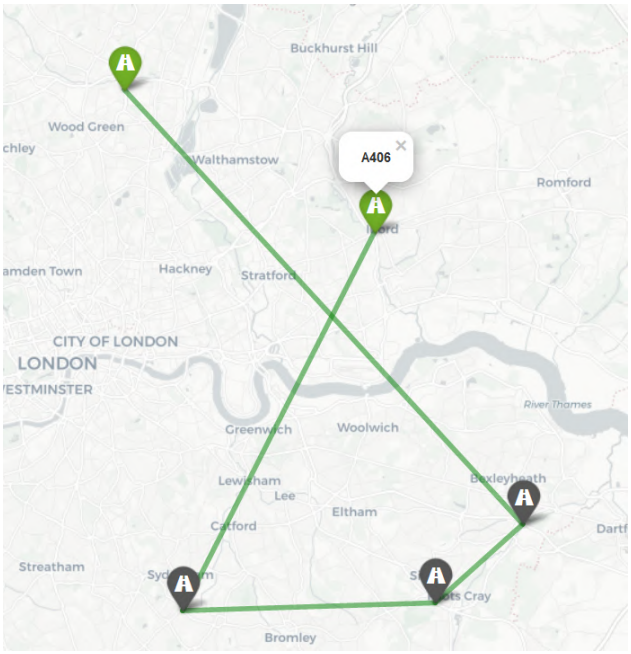
The function makes use of two main methods from the folium library :

1. ***folium.Marker()*** *– for plotting the junctions as markers*

2. ***folium.Polyline()****– to plot a link between the 2 junctions in a path.*

Brief Discussion of Visualizing Path With Specific Stops

We want to visualize *based on the specified inputs and the number of stops the user would like to make, can we find such a path?. If yes, how can this path be visually represented.*



In the graph displayed above, the different colour markers with *road icons* represent the different individual junctions for the path with specific stops as requested by the user. The green colour coded road links represent this path between the 2 inputted junctions where the black marker represents the intermediate junctions between the path. Further, upon clicking each junction, the user can obtain the name of the junction and by hovering over the road link, they can view the path between the two roads.

This specific output is achieved by running the following method under **visualizations.py** :

visualize_path_specific_stops('A406', 'LA Boundary', 3)

V Instructions

1. **Roads Dataset (Raw and unfiltered)** : Average Annual Daily Flow (AADF) - Major and Minor roads. [**8**] Unfiltered Roads Dataset
   Claim ID: eofPfeu4UQH3P7BZ
   Claim Passcode: H9DE65BFQ6yaN7CD

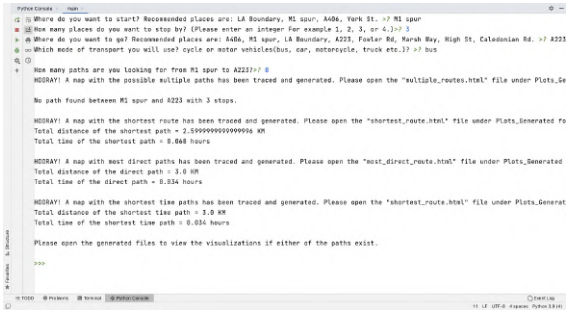
   Note: Our Program does not use the original dataset, only the filtered one. It is not required for this dataset to be downloaded for the program to run and is only given as a reference.

2. **Final Dataset (Filtered)** : Sent through https://send.utoronto.ca
   Claim ID: y5ZcSRddvbj72ZRJ
   Claim Passcode: MXVA796qiJnhnCxF

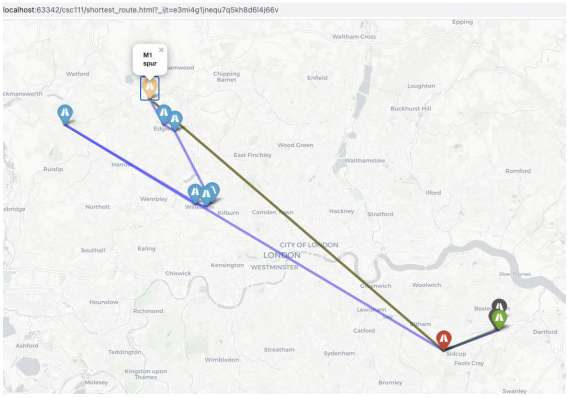
3. Download all the files and folder that have been uploaded to MarkUs. There will be one zip file, **Plots_Generatred** that has to be unzipped.

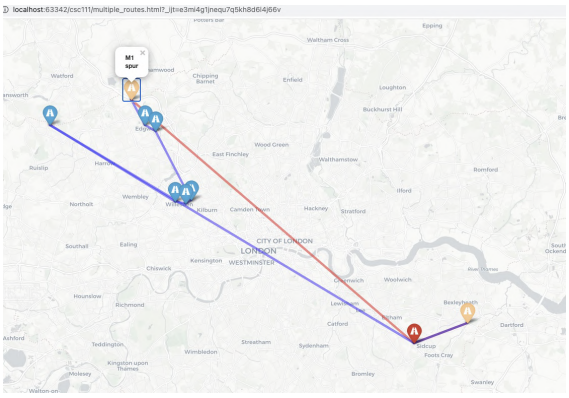
4. Open the **requirements.txt** file that was downloaded and install the libraries mentioned.

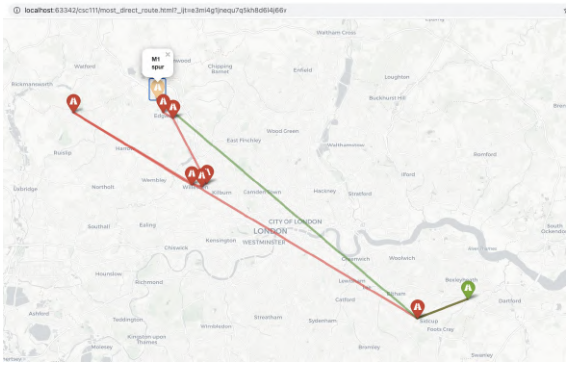
5. Open and check the **road.csv** dataset that was downloaded from MarkUs and cross reference it's contents to the filtered dataset image above. If there are discrepancies, delete the road.csv file and download it again from https://send.utoronto.ca using the ID and Passcode above. Save the dataset in the same folder as the python files for the project and ensure that the name of the dataset file is 'road.csv'.

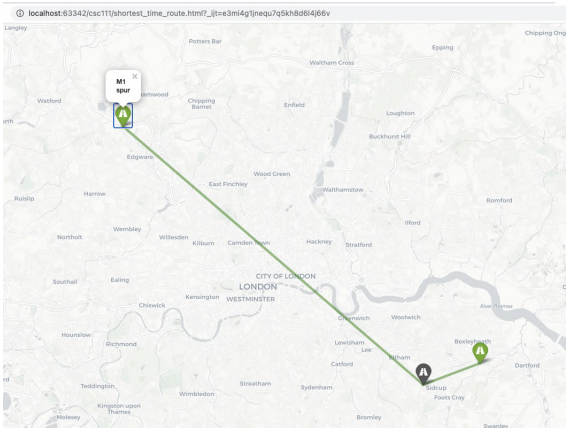


When the main file is run, the python console will pop up and ask you to fill in certain prompts. After the prompts are filled in, you will get messages similar to the ones shown in the image above. Once the plots have been generated, they can be accessed via the **Plots_Generated** folder that is present in the project folder. Running the same inputs as the one shown in the diagram above, the following outputs are expected:









The outputs above are all generated as separate HTML files in the Plots_Generated folder and can be distinguished using the address bar. For example, for the first output image above, the address bar has a section which says "shortest_route.html" which means that the outputted graph is for the shortest route for the start point and end point.

VI Changes To Plan

From the planning phase, we have made some changes to our computational plan as well as to the visualisation section. One feedback we received regarding our project proposal was that the computations we had initially planned and listed were not complex enough. Thus, we added some more computations which include finding multiple paths, the most direct path, and a path that goes through different junctions that are inputted by the user, from a start point to an end point which is inputted by the user. Moreover, we also changed our visualisation library and plan as initially we had thought of using NetworkX to visualise the graph however, we realised later on that this visualisation would not allow the visualisation to be shown on a map as we would want. Thus, after searching online, we decided to use Folium for the visualisation which allowed us to show the graph with exact coordinates on the map as well as increase interactivity with the user.

VII Discussion Section

Through the program that we have implemented, we are able to calculate and output various forms of routes from one location to another based on the inputs from the user. These can be in the form of the shortest, various possible routes or routes through different junctions and all this has been achieved through the use of graphs. We have also been able to factor in road traffic into our calculations and thus find a more accurate value for time any route will take. However, we want to take the data obtained from these visualizations one step further and discuss

How do these visualizations compare to real world situations and what problems do we encounter? Furthermore, based on the presented visualizations, can we provide any solution to the above mentioned problems.

When it comes to navigation of different possible routes, there are 2 main problems that we have identified as we analyze our research question: ***finding optimal path*** and ***accounting for the traffic in different paths***. With over *19,000* vehicles travelling each day per mile on the major urban 'A' roads and roughly *2,600* vehicles travelling each day per mile on the rural 'A' roads in the heart of London, the traffic volumes across the streets has only been increasing in volume [**9**].
According to a statistical traffic volume report published by *Department of Transport*, compared to 2018, the car and road traffic in Great Britan has increased by 2.2% and these traffic numbers have shown and exponential increase of 29.8% in comparison to the data collected in 1994. Other forms of transportation including pedal cycle, vans, lorry and motorways have also shown an increasing number of traffic volumes over the years [**10**].

From the data presented above, it is clear how ***route optimization*** is essential both in terms of helping the user reach their desired destination faster and at the same time account for the increasing traffic over many roads spread across different cities.

Our **visualize_multiple_paths()** and **visualize_shortest_path()** gives a perspective into this problem and provides a solution to this by showing us how as a user, we can accurately visualize the possible options we have to reach between the specified locations and how they can chose the best path in terms of distance requirements. The **Visualization** section under **Computational Overview** talks briefly about the visualization functions mentioned above.
The **visualize_shortest_time_path()** answers the traffic issue and displays the most efficient path between the specified locations keeping in mind of the traffic volumes on different roads. It accounts for this traffic mainly for **pedal cycles** and **motorways** transport. The **Visualization** section under **Computational Overview** talks briefly about the visualization functions mentioned above along with the formula used for calculating this the time based on traffic volume.

However, it is essential to note that these solutions are based on the limited dataset presented to us and are a close approximation to real world situation. It is imperative to note that though the route optimization has certainly improved over the years, even with these solutions, the road traffic has been increasingly high. The question we must then ask ourselves is : *Are there other efficient solutions that can address the above mentioned problems?*

One solution to this problem is the use of *Intelligent Transport System (ITS)*. "ITS aims to achieve traffic efficiency by minimizing traffic problems. It enriches users with prior information about traffic, local convenience real-time running information, seat availability etc. which reduces travel time of commuters as well as enhances their safety and comfort" [**11**].

A research paper published by Elena Alina Stanciu, Et Al. talks about *Optimization of urban road traffic in Intelligent Transport Systems*. They highlight some of the benefits of using ITS which relate to the problems we described earlier. Some of the benefits include : *decongestion of traffic on roads, increased operational efficiency, productivity increase and reduced accidents and safety increase [**12**]* .

Despite the above presented solutions with the help of our algorithms and visualizations, we did encounter and discover certain limitations to the dataset which in turn affected in us answering the problem at hand to a certain degree. Firstly, The dataset we had initially was too big to use. Since it would have taken too much time for the computations to run, we decided to restrict the function to focus on London. However, we had to restrict it further by only focusing on the major roads in London and not the minor roads. Moreover, the traffic data provided in the dataset is only a daily average and thus, the time calculated may not be an accurate representation of the actual time taken as the traffic may vary based on several factors like mode of transport and hour of the day or even the day itself!

Another limitation to this dataset was the locations of the different junctions and the roads connecting them. Due to the lack of a more descriptive data, we did not have access to the actual coordinates of several different smaller roads that connect any two junctions. Consequently, we had to limit our visualizations to show only the direct link connecting the two junctions and not the actual geographic road path traced as done in many different modern GPS sofwares like *Google Maps and Waze*.

For further evaluations, we could modify the visual output of the graph so that perhaps there is a form of a *live runner/marker* that follows the path that the user is suggested to take as is done in the case of modern GPS softwares. This could be achieved using other libraries such as pygame. Another exploration that can be done to improve this program we could look at taking gas stations and other landmarks as additional junctions which can allow for extra stops in a user's journey as this would give a much more accurate connection to a real world situation. To make this program more feasible for public use, if we can find a way to update the data in real time, we could update and optimize to give more accurate, real time updates for the estimated time taken based on current traffic volumes.

VIII References

1. Furchgott, Roy. "Filling in Map Gaps With Waze Games." Wheels Blog, 13 Sept. 2017, wheels.blogs.nytimes.com/2010/05/06/filling-in-the-map-gaps-with-waze-games.

2. David Liu And Mario Badr. "CSC110/111 Course Notes." CSC110/111 Course Notes, www.teach.cs.toronto.edu/

3. Computer Science. "Graph Data Structure 4. Dijkstra's Shortest Path Algorithm." YouTube, 7 May 2016, www.youtube.com/watch?v=pVfj6mxhdMw.

4. Coursera, and University of California San Diego. "Breadth-First Search (Continued)." Coursera, © 2021 Coursera Inc., www.coursera.org/lecture/algorithms-on-graphs/breadth-first-search-continued-WSyTa. Accessed 12 Apr. 2021.

5. Coursera, and University of California San Diego. "Implementation and Analysis." Coursera, © 2021 Coursera Inc., www.coursera.org/lecture/algorithms-on-graphs/implementation-and-analysis-gAXPb. Accessed 15

6. "What's Up with All This Traffic?" TeachEngineering.Org, 20 Jan. 2021, www.teachengineering.org/lessons/view/usf_traffic_lesson01.

7. Folium,https://python-visualization.github.io/folium/

8. "GB Road Traffic Counts - Data.Gov.Uk." GB Road Traffic Counts, 20 Oct. 2020, data.gov.uk/dataset/208c0e7b-353f-4e2d-8b7a-1a7118467acc/gb-road-traffic-counts.

9. "General Facts and Figures about Roads and Road Use." RAC Foundation RSS2, www.racfoundation.org/motoring-faqs/mobilitya40.

10. Havaei-Ahary, Behnom. Road Traffic Estimates: Great Britain 2019. Department of Transport, 10 Sept. 2020, assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/916749/road-traffic-estimates-in-great-britain-2019.pdf.

11. Choudhary, Mahashreveta. “What Is Intelligent Transport System and How It Works?” Geospatial World, www.geospatialworld.net/blogs/what-is-intelligent-transport-system-and-how-it-works.

12. Stanciu, Elena Alina, et al. “Optimization of Urban Road Traffic in Intelligent Transport Systems.” 2012 International Conference on Applied and Theoretical Electricity (ICATE), 2012, doi:10.1109/icate.2012.6403458.