Minerva University

Modeling Urban Traffic Congestion with Road-Network Metrics

Course: CS166 LBA Project

Instructor: Prof. Tambasco

December 7th, 2025

Contents

1 Model Explanation and Improvement

1.1 Code Explanation

The model starts from the road network. The class `NetworkLoader` connects to OpenStreetMap and downloads all the streets around an address in Berlin. It stores this map in the graph `G`, in which each node represents an intersection or end of a road and each edge represents one directed street segment from one intersection to the next. After loading the graph, the class goes through every edge and computes a simple estimate of travel time. It reads the length of the street in meters and the speed limit, if it exists, and if there is no speed limit in the data, it assumes thirty kilometers per hour. With these pieces it computes a travel time in minutes and saves this number as the edge attribute `travel_time`. This is the cost that the simulation uses when it needs a shortest path.

The class `Car` represents one driver moving on this network. A car only knows the node where it is now, the node it wants to reach, and a path which is a list of nodes that it still needs to visit. When a car object is created it has a starting node and a destination but the path list is empty.

The class `TrafficSimulation` controls the movement of all cars over time. When we call `simulate_traffic`, it first collects all nodes of the graph, then it creates `num_cars` many cars. For each car it chooses a random starting node and a random destination node from this list of nodes. At this step it does not guarantee that the start and destination are different, so some cars start already at their final node. After creating this list of `Car` objects, the simulation runs for a fixed number of time steps, stored in `num_steps`.

Each time step, the method `move_cars` updates the state of every car. For each car, it first checks if the car has a planned route. If the path list is empty, the simulation asks `NetworkX` for the shortest path from the current node to the destination, using `travel_time` as the edge weight. This gives a list of nodes from start to finish in order. The code removes the first node from this list, because that node is the current location of the car, and the remaining list becomes the car path. If `NetworkX` cannot find any path between the two nodes, it raises an exception that is caught and the model simply sets the current location equal to the destination and stops moving this car.

After the car has a planned path, the simulation checks if it already reached its destination. If the current node equals the destination, the car is ignored for the rest of this step. Otherwise the car tries to move to the next node in its path, which is taken from the front of the path list. Before the car actually moves, the model checks how crowded that edge is, looking at all other cars in the simulation and counting how many are at the same current node and also wanting to move to the same next node. If this count is larger than five, the model treats this as a traffic jam. In that case, the car does not move. The code pushes the next node back to the start of the path list so the car will try again at the

next time step. If the number of cars is at most five, the car moves to the next node and its `current_location` is updated. The model repeats this logic for every car in each time step. After moving all cars, the simulation prints how many cars are at their destination. At the end of all steps, it returns the list of cars, with their final positions and remaining paths.

The `NetworkVisualizer` class uses this final state to visualize congestion on the map. It does not show the cars as points. Instead it counts, for each directed edge from node `u` to node `v`, how many cars are currently at node `u` and want to move next to node `v`. This is stored in a dictionary where the key is the pair `(u, v)` and the value is the number of cars. Then it draws the road network with `osmnx`. For each edge it looks up this count, making streets with zero cars with a very light red color and thin line, and streets with more cars with darker red and thicker line. This means that on the final map, the darkest and thickest streets are the ones with the largest number of cars that want to use that edge at the same time in the next step.

1.2 What aspects of real roads and traffic the model captures

The simulation does capture some important aspects of real traffic. First, it uses the real network topology from OpenStreetMap. The nodes in the graph represent actual intersections and the edges represent real streets, including one way streets and the real pattern of which roads connect where. This means that features like side streets, main avenues and small loops appear naturally in the model, instead of being invented.

The model also captures route choice in a simple way. Each car picks a shortest path based on travel time, not just on the number of hops. The car looks for the path that minimizes the sum of the `travel_time` values on the edges, which is similar to how a GPS app works, where the driver tries to follow the route that is expected to be faster, not just the physically shortest one.

Speed limits enter the model through the `travel_time` attribute. Different roads have different `maxspeed` values, so the time to travel a given length of road changes across the network. A longer road with a high speed limit can still be attractive compared to a shorter but slower street. This is a basic way to say that fast roads and slow roads are not equal for a driver.

It includes a simple form of congestion feedback. When many cars want to move along the same edge at the same time, some of them are blocked and stay in place. In the code this happens when more than five cars are queued for the same move. This is not a detailed description of queues, but it still creates the effect that heavy demand on a road slows down the flow of cars.

Because of these, the simulation can show bottleneck formation. When many random

trips share parts of their shortest path, certain edges carry much more traffic than others. These edges become darker and thicker in the visualization, which highlights them as bottlenecks. In that sense, the model can reveal which streets in the local network tend to attract a lot of flow and are more likely to become congested, even with all the simplifications.

1.3 Simplifications and limitations

Although the model has some realistic elements, it still makes many simplifications when compared to real traffic. One important is how cars move along streets. In the simulation, a car jumps from one intersection node to the next in a single step, independent of the actual length of that edge. The travel time attribute is used for planning the path, not for the actual motion, which means a car can traverse a very long highway segment in one step, the same as a short side street. The model does not track partial positions along an edge, only the nodes at the ends.

The rule for congestion is also quite strict. The threshold of five cars is the same for every edge in the network, which means there is also almost no physical realism in the way cars behave. Cars have no length or physical size, so they never block space on the road except through the simple counter of how many cars want a given move. A small side street and a multi lane avenue both have the same capacity in terms of this threshold. In addition, the cars only check congestion at the level of the next move, so they do not adapt their route depending on current congestion.

Another simplification is how origins and destinations are chosen. At the beginning of the simulation, every car gets a random start node and a random destination node. In a real city, origins and destinations follow patterns. Morning peaks have cars going from residential areas to work areas, and the opposite in the evening. The random choice here ignores land use, time of day, and demand patterns, so the flows may not represent typical commuting traffic.

The travel time formula itself has limits. It uses a simple division of length by speed limit, not include acceleration, deceleration, traffic control, or the effect of other cars on speed. Once the edge weights are set at the start, they stay constant through the simulation. However, in real traffic, effective speed on a road segment decreases when density increases. The current model cannot adapt travel time based on congestion levels, so jams only appear when many cars want to use the same edge and are prevented from moving. There is no gradual slow down.

1.4 Improved road congestion

1.4.1 How does the model determine whether or not a road is congested?

In the original version a road was considered congested as soon as more than five cars tried to use the same directed edge in the same time step. That threshold did not depend on the type of road or on its length. A small residential street and a multi lane highway both became congested once they had more than five cars queued for the next move. Cars either moved at full speed when the count was below the threshold, or did not move at all when it was above, so there was no gradual slow down as traffic increased.

1.4.2 Improved model

For each edge in the network we estimate a jam capacity that depends on the road type and its length. We use the OSM highway tag to distinguish highways, primary roads and residential streets, and we give each type a different capacity multiplier. Longer edges get higher capacity than tiny ones. This gives a `jam_density` value that says roughly how many cars that specific edge can hold before it is packed.

During the simulation we then compute the current density on each edge by counting how many cars are on it. Using the idea from Session 9, we link density and speed. When the edge is empty cars move at free flow speed, and as more cars enter, the speed factor goes down. When the number of cars approaches the `jam_density` the speed factor gets close to a small minimum value, so cars move but much more slowly. This matches the fundamental diagram from the Nagel and Schreckenberg model, where average speed decreases as density increases.

The last step is to track where cars are along each road, not just at the intersections. We gave each car a `position_on_edge` between 0 and 1 and a `next_node`. When a car starts a new edge its position is 0. At each time step it increases this position by an amount that depends on the speed factor, and when the road is empty the position grows fast and the car crosses the edge in a few steps. When the road is crowded the position grows slowly and the car needs more steps to reach the end. Only when the position reaches 1 does the car move to the next node. With this change we still keep the model simple, but congestion is now modeled as slower movement along each road instead of a hard stop at an arbitrary threshold. The final outcome can be seen in Figure 1.

Figure 1: Traffic snapshot after 10 steps with the improved congestion model. Darker red roads are the ones that currently carry more cars and therefore have lower speeds, while pale roads have little or no traffic. This shows how the new model slows cars down gradually on busy segments instead of stopping them at a fixed threshold.

2 Improving Visualization

For the visualization part we first looked at what the original ChatGPT code was actually showing. The old plot only showed traffic at the final time step of the simulation. At that point most cars have already reached their destinations, so many roads look empty even if they were busy earlier. The map was basically a snapshot of the last moment, not a picture of where traffic spent most of its time. That makes it hard to spot the most recurring congestion.

To fix this we extended the simulation so it keeps a congestion history while it runs. At every step, for each car that is currently traveling along an edge, we add one count to a dictionary that stores how many "car steps" passed through each edge during the

whole run. This means that a road that has ten cars on it for five steps will accumulate fifty counts. When plotting the map, we use these cumulative counts instead of the last snapshot. Edges with low totals are drawn in green, edges with moderate totals in yellow and edges with high totals in red, like a traffic light scale. The most used edge is highlighted in bright cyan (or more than one in case there are multiple roads in first), and line width also grows with the total count, so very busy roads look thick and red while quiet streets are thin and green.

Below the map, the code prints a short report with the top congested roads, using the OpenStreetMap street names whenever they exist (See Table 1). Taken together, the cumulative color map and the text report show not just where cars happened to be at the end, but which specific streets carried the most traffic over the entire simulation (See Figure 2). This makes it much easier to identify consistent congestion and roads for interventions or widening.

Table 1: Top 10 most congested roads

Rank	Road	Car-steps	% of max
1	Engeldamm	55	100.0%
2	An der Schillingbrücke	53	96.4%
3	Bethaniendamm	53	96.4%
4	Adalbertstraße	48	87.3%
5	Bethaniendamm	48	87.3%
6	Adalbertstraße	45	81.8%
7	Bethaniendamm	45	81.8%
8	Bethaniendamm	39	70.9%
9	Bethaniendamm	39	70.9%
10	Adalbertstraße	36	65.5%

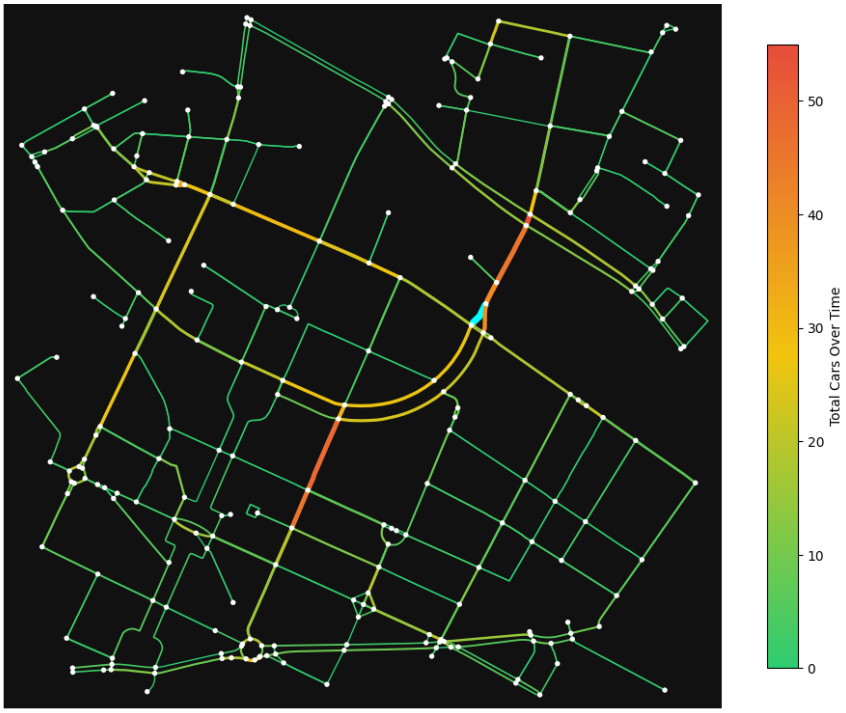

Figure 2: Cumulative traffic congestion on the Berlin road network around Kottbusser Tor. Green roads carried little traffic over the whole simulation, yellow roads carried moderate traffic, and red roads are the most heavily used. The short cyan marks the single most congested road in this local network.

3 Analyze and predict traffic congestion

3.1 Task A

We kept the improved simulation from Task 2, where congestion slows cars down gradually along each edge, and created a quieter version that does not print at every step. In each run the model picks random start and destination nodes for 100 cars, then moves them until they arrive or until we hit a step limit. At every time step we call `_update_congestion_history`, which adds one count for each car that is traveling along a given edge. This count is what we call a "car step". One car sitting on a road for ten steps adds ten car steps. Ten cars on the same road for ten steps add one hundred. This measure captures both how many cars use a road and how long they stay there.

To get stable congestion patterns we repeated this simulation 100 times on the same Berlin network. After each run we added its `congestion_history` to a global dictionary

called `aggregated_congestion`. In the end each edge in the graph has a single number that represents the total car steps it accumulated over all 100 simulations. Edges with high totals are roads that carry traffic in many different random scenarios and that tend to keep cars on them for a long time, which is exactly what we would expect from bottlenecks.

We then ranked all roads by their aggregated car steps and looked at the top ten (Table 2).

Table 2: Top 10 most congested roads

Rank	Road	Car-steps	% of max
1	An der Schillingbrücke	4945	100.0%
2	Bethaniendamm	4825	97.6%
3	Engeldamm	4788	96.8%
4	Köpenicker Straße	4307	87.1%
5	Bethaniendamm	4143	83.8%
6	Bethaniendamm	4135	83.6%
7	Bethaniendamm	4132	83.6%
8	Bethaniendamm	3876	78.4%
9	Köpenicker Straße	3125	63.2%
10	Heinrich-Heine-Straße	3114	63.0%

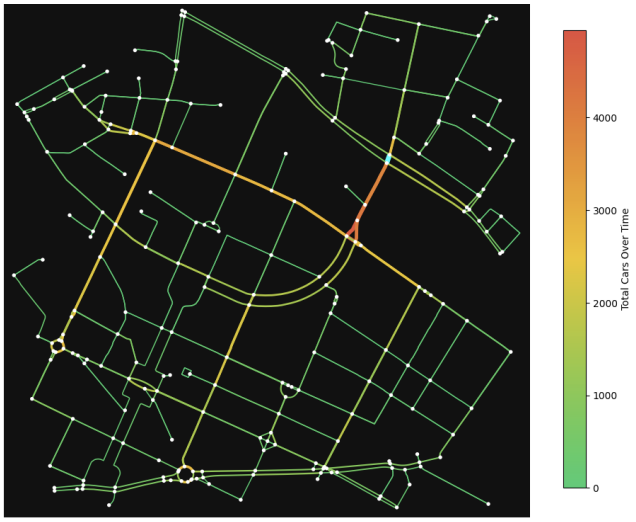

Figure 3: Cumulative congestion over 100 simulation runs on the Berlin network. Edges are colored by total car-steps: green roads carry little traffic, yellow roads carry moderate flow, and red roads are the main congestion, with the cyan segment marking the most used road.

These are long arterial roads that connect the bridge and the main east–west direction of the neighborhood (Figure 3). In the cumulative congestion map, these roads appear in yellow and red, and the single most congested segment is highlighted in bright cyan. Short residential streets inside the grid stay mostly green, which means they rarely carry much flow in this simple origin–destination model.

Overall, this empirical analysis suggests that the main connectors in the local network are also the main congestion hotspots in the simulation.

3.2 Task B

To connect the empirical congestion patterns with network structure we compared two metrics from Sayama's Chapter 17: edge betweenness centrality and a simple degree based measure. Edge betweenness centrality tells how often a road lies on shortest paths between all pairs of nodes. Roads with high betweenness are important "bridges" in the network, so our guess was that they would often become congestion spots. For the degree based measure we took the average degree of the two end nodes of each edge. The idea is that roads connecting big intersections might also be busy.

For every edge in the Berlin network we already had a congestion value from Task 4.a, measured as total car-steps aggregated over 100 simulations. We matched each edge with its betweenness and degree, then computed Pearson correlations between each metric and the congestion values. Edge betweenness had a correlation of about r = 0.63, which gives $R^2 \approx 0.39$. This means that around forty percent of the variation in congestion can be explained just by how central an edge is in terms of shortest paths. The p-value is tiny, so this relationship is statistically significant. The degree based metric performs much worse. The correlation between average endpoint degree and congestion is only $r \approx 0.06$, with $R \approx 0.004$, and the p-value is not significant (Table 3 shows the comparison).

Table 3: Correlation between network metrics and congestion

Metric	Pearson r	R^2	p-value
Edge betweenness centrality	0.6264	0.3924	5.72×10^{-60}
Degree centrality	0.0642	0.0041	1.37×10^{-1}

We also compared the top-10 lists. Among the ten most congested roads, six of them also appear in the top-10 edges by betweenness, while only one appears in the top-10 by degree (See Table 4). Putting all this together, the best predictor of congestion in my model is clearly edge betweenness centrality. This also makes intuitive sense. The simulation sends cars along shortest paths, so edges that lie on many shortest paths are used again and again. These edges act like structural chokepoints and naturally accumulate more car-steps, which shows up as both high betweenness and high congestion.

Table 4: Top-10 overlap with most congested roads

Metric	Top-10 overlap	Best predictor?
Edge betweenness centrality	6/10 roads	Yes
Degree centrality	1/10 roads	No

The scatter plots give a visual version of this. In the betweenness plot almost all points form a rising cloud. Most low betweenness edges sit near the bottom with low congestion, while edges with higher betweenness are much more likely to reach the upper part of the graph with thousands of car steps. The pattern is not perfectly linear, but there is a clear tendency that more central roads carry more traffic in the simulation. In contrast, the degree plot looks almost flat. Points with the same average endpoint degree can have very different congestion values, and there is no clear upward trend. Roads attached to large intersections are not systematically more congested than roads in smaller junctions. Seeing both plots side by side makes the difference between the two metrics quite clear and supports the choice of edge betweenness as the main predictor.

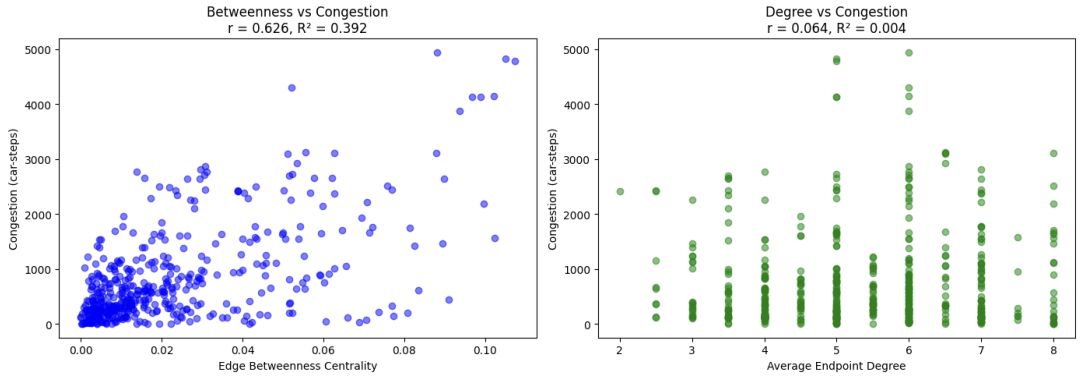

Figure 4: Scatter plots comparing network metrics with simulated congestion. Left: edges with higher betweenness centrality tend to have higher congestion (clear upward trend). Right: average endpoint degree shows almost no relationship with congestion, with points spread vertically for each degree value.

3.3 Task C

To see whether edge betweenness centrality is a general predictor of congestion, we repeated the same experiment in Buenos Aires. We used the road network within one kilometer of Plaza de Mayo and ran one hundred simulations with one hundred cars each, using the same improved traffic model as before. For every edge we again measured congestion

as total car-steps summed across all simulations and then computed edge betweenness on the Buenos Aires graph.

The first result is the correlation between edge betweenness and congestion. In Buenos Aires the Pearson correlation is about $r = 0.31$ with $R^2 \approx 0.09$, so betweenness explains around nine percent of the variation in congestion. The p value is extremely small, which means the relationship is still statistically significant. However it is much weaker than in Berlin, where betweenness explained almost forty percent of the variation. The scatter plot for Buenos Aires (Figure 5) shows this difference visually. There is a positive trend, but the cloud of points is more spread out and many edges with low betweenness still collect moderate congestion.

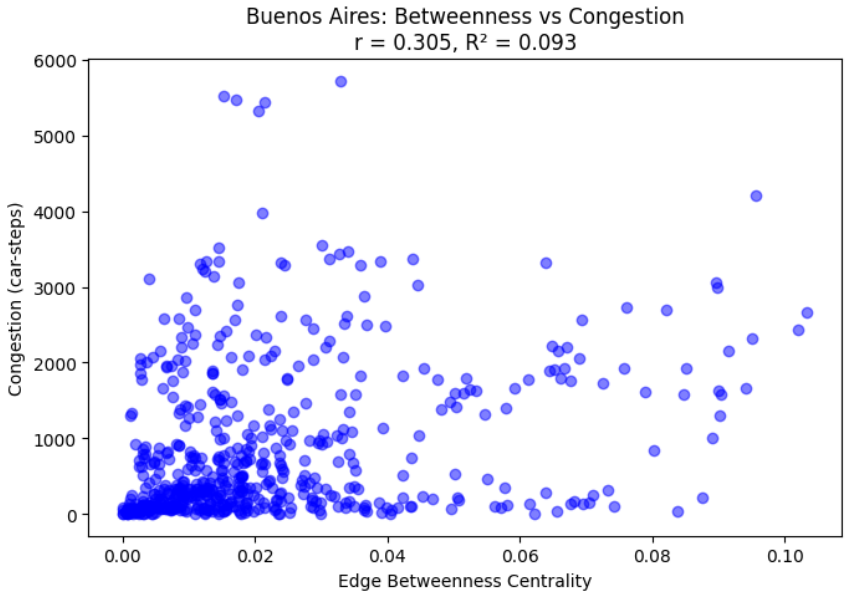

Figure 5: Relationship between edge betweenness centrality and simulated congestion in the 1 km road network around Plaza de Mayo, Buenos Aires. Each point is a road segment, with congestion measured as total car-steps over 100 simulations.

To understand where the predictor fails we compared the ten most congested roads with the ten edges that have the highest betweenness. The comparison table (Table 5) lists the top ten congested streets together with their betweenness rank in the full network. Most of the congestion takes place on wide avenues such as Avenida Rivadavia. These are major east–west and north–south arteries near the center of the city. They are used by many cars in the simulation but they do not always have the highest edge betweenness rank. Only one of the ten most congested roads appears in the global top–ten betweenness list.

Table 5: Top-10 comparison between congestion and betweenness ranks in Buenos Aires.

Road Name	Congestion Rank	Betweenness Rank
Avenida Rivadavia	1	116
Avenida Rivadavia	2	285
Bolívar	3	259
Avenida Rivadavia	4	200
Avenida Leandro N. Alem	5	209
Avenida La Rábida	6	3
Avenida Leandro N. Alem	7	205
Avenida Hipólito Yrigoye	8	129
Avenida de Mayo	9	301
Avenida Belgrano	10	107

The cumulative congestion map for Buenos Aires (Figure 6) helps explain this. The figure shows that congestion tends to concentrate along a few curved ramps and along long straight avenues that cut across the grid, especially where several lanes merge near Plaza de Mayo. In contrast the rest of the grid absorbs traffic through many alternative parallel streets.

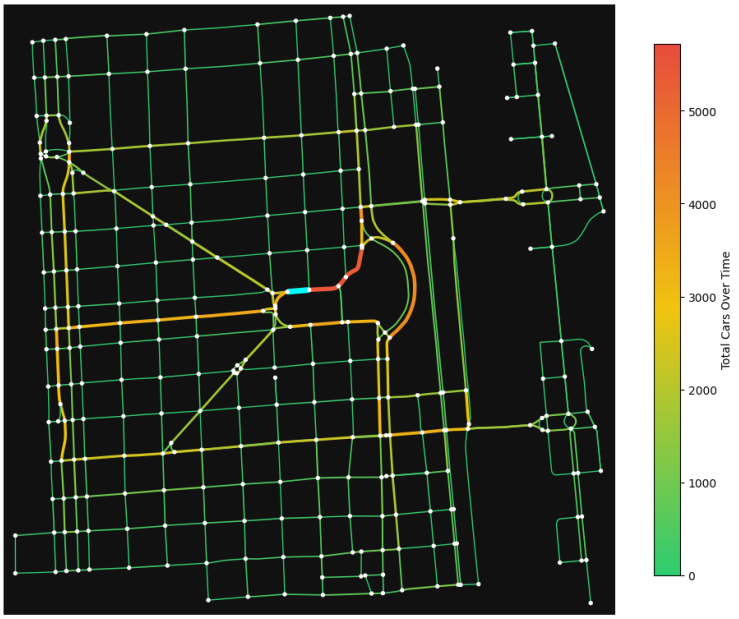

Figure 6: Cumulative traffic congestion in a 1 km road network around Plaza de Mayo, Buenos Aires. Colors show how many cars used each road over 100 simulations (green = low usage, yellow = moderate, red = high congestion, cyan = highest).

In a grid like this an avenue can carry a lot of flow without being the unique bridge between distant parts of the network, so its edge betweenness is not especially high even though it is busy in practice. This suggests that edge betweenness centrality is still a useful indicator of congestion in Buenos Aires, but it is not as strong as in Berlin and misses some important roads. The geometric layout of the city, with many near-parallel routes, seems to weaken the link between structural centrality and simulated traffic jams.

4 AI Statement

For this assignment we used Gemini in two ways. First we asked for guidance on how to add and format the color bar on the right side of the congestion map. It showed us how to connect the color scale to the edge congestion values, choose a colormap, and add a label. We then wrote and adapted the code so it would work with the plotting functions.

Second, after we finished a full draft of the report, we asked to check whether we had included all the required pieces of the assignment, such as the description of the model, parameter choices, experiments, results, and discussion. It confirmed that they were present and suggested some small wording changes, which I used only when they matched what I wanted to say. All modeling decisions, simulations, analyses, and interpretations in this report are our own.

5 Resources

Sayama, H. (2015). *Introduction to the modeling and analysis of complex systems.* Open SUNY Textbooks.