LBA - Traffic Modeling

Modeling and Analysis of Complex Systems
Minerva University
Fall 2025
Tambasco, TTh@3PM UTC

December 7, 2025

Contents

1 Introduction

In 2014, the *Centre for Economics and Business Research* released a study that unveiled a shocking statistic on the economic costs of every commuter's nightmare: traffic jam. In the US, France, UK and Germany alone, traffic jams incurred $293 billion in foregone productivity (Deutsche Welle, 2014). It is well-known that congestion does not appear uniformly throughout cities, but most at certain 'focal points'. Identifying these hot-spots is the first step in improving traffic networks to reduce congestion. This analysis will explain and improve an AI-generated traffic simulation of Berlin's road network. Finally, we will derive and evaluate a theoretical solution to predict congestion beyond Berlin.

2 Explaining the Code

Disclaimer

In this section we describe the behaviour of our `TrafficSimulation` using the *bug-corrected* version of the starter code. The simulation uses

- a `networkx.MultiDiGraph` from OSMnx (Boeing, n.d.),
- shortest paths weighted by a per edge attribute `travel_time`,
- node based car movement with a simple congestion rule,
- an edge based congestion heatmap indexed by triples (u, v, k).

2.1 How does the simulation work?

Road network and (u, v, k) edges

We load a 1 km radius car network around an address in Berlin using `ox.graph_from_address`. OSMnx returns a `MultiDiGraph`. Each node represents an intersection or dead end, identified by a numeric id. Each directed edge represents a road segment from node u to node v. Because there can be several parallel segments between the same pair of nodes (for example divided roads or one way pairs), NetworkX uses an extra edge key k. When we iterate with

```
for u, v, k, data in G.edges(data=True, keys=True):
```

we get the endpoints u, v, the key k that distinguishes parallel segments, and an attribute dictionary `data` which stores the length, speed limit, OpenStreetMap id, and later our `travel_time`. Several edges can share the same `osmid` when a long OSM way is split into smaller pieces. To keep congestion information aligned with the actual graph structure we always index edges by (u, v, k), not by `osmid`.

Cars and routes

A car is a very small object:

```
class Car:
    def __init__(self, start, destination):
        self.current_location = start
        self.destination = destination
        self.path = []
```

For each car we store:

- `current_location`: the node where the car currently sits,
- `destination`: the target node,
- `path`: the remaining route as a list of future nodes.

We maintain the invariant that `path` never contains the current node. If the car is at node A and its route is $[A, B, C, D]$, we immediately drop A so `path` becomes $[B, C, D]$. At every time step, `path[0]` is the next node on the planned route. The `TrafficSimulation` object stores the shared state:

- the graph `G`,
- `num_cars` and `num_steps`,
- a list `self.cars` of `Car` objects.

All methods act on this shared list, which fixes the original bug where cars were created inside a function and were invisible to the plotting code.

Random feasible trips:

We first sample `num_cars` random trips. Starts and destinations are drawn uniformly from the node set. A `while` loop resamples until two conditions hold:

1. start and destination are different nodes;
2. there is a directed path from start to destination, checked with `nx.has_path`.

This prevents impossible trips and avoids `NetworkXNoPath` exceptions. After this step `self.cars` is a list of independent, feasible trips scattered across the network.

Edge travel times from length and speed

Next we attach a travel time to each directed edge:

```
for u, v, k, data in self.G.edges(data=True, keys=True):
    speed = data.get("maxspeed", 30)
    if isinstance(speed, list):
        speed = speed[0]
    speed_mpm = float(speed) * 1000.0 / 60.0 # km/h -> m per minute
    data["travel_time"] = data["length"] / speed_mpm
```

Here `length` is in meters and `maxspeed` is in km per hour. We convert speed to meters per minute and obtain a crude travel time, in minutes, for that road segment:

$$\text{travel\_time} = \frac{\text{length (m)}}{\text{speed (m/min)}}.$$

This attribute is used as the weight in shortest path calculations, so routes try to minimise driving time rather than only distance or hop count.

Movement and congestion rule

The movement logic for one time step is contained in `move_cars`. The high level structure is:

1. Loop over all cars.
2. Skip cars already at their destination.
3. If a car has no route, compute a shortest path from its current node to its destination using `travel_time` as weight, then drop the current node from the list.
4. If there is no node left in `path`, there is nothing to do.
5. Otherwise, look at the candidate `next_node` and decide whether the outgoing edge is congested.

The congestion check is where interaction between cars appears:

```
next_node = car.path[0]
cars_on_edge = sum(
    1 for c in self.cars
    if c.current_location == car.current_location
    and c.path and c.path[0] == next_node
)
```

We count how many cars currently sit at the same node and intend to move to the same next node in this step. That is, we approximate the demand for the directed edge (`current_location` → `next_node`). If more than five cars want that edge, the car waits:

```
if cars_on_edge > 5:
    # congested edge car stays at this node
    continue
```

If at most five cars share the edge, the car moves one intersection forward:

```
car.current_location = next_node
car.path.pop(0)
```

So every time step each car can advance at most one node along its route. Edges behave like simple capacity constrained channels: if too many cars aim for the same outgoing edge in the same step, some of them will still be queued at the intersection. Cars that have reached their destination never recompute a path and never move again, so finished trips disappear from the dynamic traffic load.

Edge-level congestion and indexing by (u, v, k)

To draw a heatmap of congestion we need a count of cars per road segment. The method `edge_car_counts` builds a dictionary `counts[(u,v,k)]` with one entry per directed edge:

```
counts = {(u, v, k): 0 for u, v, k in self.G.edges(keys=True)}
for car in self.cars:
    if car.path:
        u = car.current_location
        v = car.path[0]
        if self.G.has_edge(u, v):
            for _, _, k in self.G.edges(u, v, keys=True):
                counts[(u, v, k)] += 1
                break
```

We initialize all edge counts to zero. For each car that still has a remaining path we look at its current node u and next node v. That pair identifies the directed edge it will try to use next. The inner loop selects one concrete edge key k between u and v and increments the count for that particular (u, v, k). The `break` ensures a single car is assigned to only one lane among potentially many parallel lanes. It is important that the dictionary keys match exactly the triples we obtain from `G.edges(keys=True)`. If we instead used `osmid`, counts could combine several different segments and lose information about direction and exact endpoints.

Plotting the congestion heatmap

The final plot uses these counts to assign colours and widths to edges:

```
counts = self.edge_car_counts()
max_val = max(counts.values()) if counts else 0
max_count = max_val if max_val > 0 else 1

edge_colors = []
edge_widths = []

for u, v, k, data in self.G.edges(data=True, keys=True):
    count = counts.get((u, v, k), 0)
    edge_colors.append(plt.cm.Reds(count / max_count))
    edge_widths.append(1 + count)

ox.plot_graph(
    self.G,
    figsize=(10,10),
    edge_color=edge_colors,
    edge_linewidth=edge_widths
```

```
)
```

We first compute the maximum count so that congestion levels can be normalised to the interval $[0, 1]$. When building `edge_colors` and `edge_widths` we iterate over edges in the same order that OSMnx uses when drawing the graph. For each edge (u, v, k) we look up its count; an edge with no queued cars gets count zero. The normalised value `count/max_count` is passed to the `Reds` colormap. Edges with low or zero load appear almost white, and edges with the highest loads appear dark red. Line width is set to $1 +$ `count`, so heavily used streets are also visually thicker. The result is a static congestion map of the last simulated time step that highlights which specific road segments received the most demand under our random trips.

Full simulation loop

Finally, the global time evolution is controlled by

```
def simulate_traffic(self):
    for step in range(self.num_steps):
        self.move_cars()
```

Each iteration represents one discrete time step. At every step, all cars simultaneously apply the movement rule described above based on the state at the start of the step. Paths are recomputed only when needed, congestion is evaluated per outgoing edge, and cars either advance one node or remain at their current intersection. After `num_steps` iterations we have a final configuration of car positions and edge demands that we visualise with the congestion heatmap. Overall, the code implements a node based traffic model on a real `MultiDiGraph`: cars follow shortest time routes, move in discrete steps from node to node, and experience delays when too many cars choose the same outgoing edge. Indexing by (u, v, k), using random feasible trips, and weighting edges by travel time together provide a coherent explanation of the resulting traffic patterns on this local Berlin road network.

2.2 Which aspects of real roads and traffic does this simulation capture effectively?

This model captures several important qualitative features of real road traffic, despite its simplicity.

Realistic street network structure

By using OSMnx with `network_type="drive"`, we obtain a `networkx.MultiDiGraph` derived from OpenStreetMap (OSM). Nodes in this graph are actual intersections and dead ends in Berlin, while directed edges represent road segments that respect one-way restrictions. This means that when a car follows a route, it moves along geographically plausible driving paths that obey the direction of travel on each street.

Travel-time weighting that distinguishes road classes

The model distinguishes between different types of roads through travel-time weighting. For each edge, we compute

$$\text{travel\_time} = \frac{\text{length (m)}}{\text{speed (m/min)}},$$

using the edge length and the speed limit (or a default of 30 km/h if missing). Shortest paths are then computed with this `travel_time` attribute as the weight. As a result, cars naturally prefer shorter and faster roads over longer or slower ones, which mirrors real driver behaviour more closely than a model that simply minimizes the number of intersections.

Heterogeneous origin–destination routes

In the `initialize` method we draw random start and destination nodes for each car, but we enforce that there is a directed path between them. This avoids pairs that would crash the code, and it creates a diverse set of trips that cross the network in many directions. Some roads become natural "funnels" where many shortest paths overlap, which is exactly what happens in real cities where certain corridors serve as connecting routes between multiple neighbourhoods.

Local interaction and congestion formation at intersections

Cars interact through local competition for outgoing edges at intersections. The congestion logic

```
cars_on_edge = sum(
    1 for c in self.cars
    if c.current_location == car.current_location
    and c.path and c.path[0] == next_node
)
```

counts how many cars want to use the same directed edge from the same node in the current time step. If the count exceeds a threshold, the car waits. This captures the idea that road segments have limited capacity and that vehicles can delay each other when too many try to use the same approach. Congestion in this model is an emergent property of many cars interacting on shared edges, not something fixed ahead of time.

Intersections as natural bottlenecks

Cars live on nodes and move one edge per time step if allowed. The decision to move depends on the number of cars that want the same outgoing edge from a shared node. This is a coarse representation, yet it correctly highlights that queues and conflicts tend to form at junctions rather than at arbitrary interior points of road segments.

Visualisation of macroscopic patterns from microscopic rules

The `edge_car_counts` method converts individual car states into edge-level counts, and `plot_network_with_traffic` maps those counts to colour intensity and line width. This produces a congestion heatmap over the real street layout that highlights which corridors are most heavily used at the end of the simulation. It is a natural way to interpret where traffic is concentrating, given the origin–destination pattern and the route-choice behaviour encoded in the model.

2.3 Simplifications and limitations of the model

At the same time, the model makes strong simplifying assumptions that limit its realism.

Simplified car behaviour and lack of re-routing

Each car computes a single shortest-time route when its `path` is empty and never re-routes in response to congestion or delays. Real drivers, and especially navigation systems like Google Maps or Naver, often adapt their routes when they encounter heavy traffic. All cars are identical: there is no variation in driver preferences, desired speeds, or tolerance for congestion. This is highly unrealistic because some areas might be more residential, and drivers almost always have 'prefered' nodes like their home, workplace, or supermarket they might travel on more frequently. Movement is node based and discrete: a car either moves exactly one intersection or does not move at all. We do not track continuous positions along edges, and we do not model acceleration, deceleration, or car-following within a lane.

Coarse and global congestion model

The key rule

```
if cars_on_edge > 5:
    # jam: car waits
    continue
```

uses a hard threshold of 5 cars for every edge in the network. A small residential street and a major arterial road both "jam" at the same number of vehicles, regardless of their length, number of lanes, or posted speed. Congestion is binary: below the threshold cars move at full speed; above it cars do not move at all. Real traffic exhibits a smooth relationship between density and speed. Spillback effects, where queues on one link block upstream intersections, are not represented.

Simplified treatment of time

The `travel_time` attribute is used when choosing routes but not when determining how long it takes to traverse an edge dynamically. Every uncongested move advances a car exactly one edge per step regardless of length or posted speed. The discrete time step therefore has no physical interpretation. All cars spawn at time step 0; there is no time-varying demand such as a morning peak. Once a car reaches its destination it stops, and no new cars enter the system.

Local network only, with no surrounding-city dynamics

The model includes only the roads within a fixed radius around a Berlin address. Flow into and out of this region from the broader city is not represented. Trips may begin or end near the boundary by chance, but there is no mechanism for external demand loading major connectors that cross the boundary. Especially in large cities like Berlin, a significant portion of the travelers consist of commuters who live outside the city and might congest less central routes - these commuters are not included in this model.

No traffic control or turning priorities

There are no traffic lights, stop signs, or right-of-way rules. All outgoing edges from a node are treated symmetrically, aside from congestion and travel-time weighting. Turning left across traffic versus proceeding straight is treated identically, removing an important source of real intersection delay.

No lane structure or within-edge dynamics

Although the graph is a `MultiDiGraph` and the edge key k distinguishes parallel edges when counting, this is only for indexing. Multiple lanes, lane changes, overtaking, and lane-specific capacities are not represented. Congestion is tracked at the level of whole edges rather than lanes or segments.

No calibration or validation

The default speed of 30 km/h and the congestion threshold of 5 cars per edge are convenience choices rather than empirically grounded parameters. We do not compare simulated flows or travel times to observed data from Berlin, nor do we tune the model to match real performance metrics. The simulation is intended as a coherent and interpretable toy model rather than a predictive system.

2.3.1 Limitation Summary

Overall, the simulation captures the basic structure of real road networks, the idea that drivers seek time-efficient routes, and the emergence of congestion when many vehicles compete for limited-capacity links. It abstracts away most fine-grained physical, behavioural, and control mechanisms that shape real traffic, which is important to keep in mind when interpreting its results.

2.4 Modeling congestion: from a fixed threshold to density-based dynamics

How the current model detects congestion

In our current implementation, congestion is determined by counting how many cars want to use the same outgoing edge from a given intersection in a time step. Inside `move_cars`, we compute

```
cars_on_edge = sum(
    1 for c in self.cars
    if c.current_location == car.current_location
    and c.path and c.path[0] == next_node
)
```

and then apply the rule

```
if cars_on_edge > 5:
    # jam: car waits
    continue
else:
    car.current_location = next_node
```

```
car.path.pop(0)
```

The model treats the directed edge (`current_location` → `next_node`) as congested whenever more than 5 cars are queued to enter it in the same time step. Otherwise, every car that wants to use this edge moves forward by one intersection. Congestion is therefore defined by a global fixed threshold of 5 cars per edge, independent of the physical properties of that road segment. This choice is arbitrary for several reasons. It does not depend on edge length, number of lanes, or speed limit, and it creates a binary behaviour in which edges are either in free flow (all cars move) or completely jammed (no car moves). Real traffic exhibits a smoother relationship between traffic density and speed, with different capacities for different road types.

Improving congestion modeling with edge-specific density

A more realistic approach is to model congestion through the car *density* on each edge, rather than a raw count with a fixed cutoff. For each directed edge (u, v, k) with length L_{uvk}, we define an edge specific capacity

$$C_{uvk} = \frac{L_{uvk}}{\ell_{\text{veh}}},$$

where ℓ_{veh} is an effective vehicle length including spacing. In Europe, the average vehicle length is 4.536m (Munoz, 2024), while it is a common rule of thumb to keep at least half your vehicle's length as a distance to the vehicle in front of you. Hence, we define $\ell_{\text{veh}} \approx 2 \times 4.5 = 9$ meters). This gives an approximate number of cars that can reasonably occupy that road segment.

The instantaneous density on that edge is then

$$\rho_{uvk} = \frac{n_{uvk}}{C_{uvk}},$$

where n_{uvk} is the number of cars currently associated with that edge. This value is updated at each time step of the simulation, as cars move off or join onto a road at different rates, hence changing the density at each simulation step.

Speed as a function of density

In the Nagel-Schreckenberg traffic model, each car has a maximum speed at which it can move, and its current speed v_i depends on the the distance to the car in front of it, d. If $v_i > d$, then $v_{i+1} = d$, and if $v_i < d$ and $v_i < v_{max}$, then $v_{i+1} = v_i + 1$ (Nagel & Schreckenberg, 1992). This model suggests a relationship between speed and 'density', where traffic jams leads cars to slow down. This is a realistic assumption which was included into the traffic model by making the speed with which a car moves along a street dependent on the street's current car density. For this, the `position` attribute was added to each `Car` object, which stores the relative position of the car along the length of the street. If there is no traffic on that road, a car should move at the maximum allowed speed limit (which is extracted from the `osmnx` library). The more traffic there is, the slower the car is able to move, ultimately taking longer to drive along the entire road. While the Nagel-Schreckenberg models each car in one specific discrete position, our implementation uses the average car density on a road to determine the speed, which is adopted *uniformly* by all cars on that road. There is also the possibility to extend this by including random slowdowns by adding a probability that a driver drives more slowly than they are allowed to.

```
p_slow = 0.3
       if random.random() < p_slow:
           speed_factor *= 0.5
```

When a car leaves a node, it chooses the next edge on its shortest-time route and sets `edge` and `x = 0`. At each time step, we compute the density ρ_{uvk} on that edge and determine a congestion adjusted speed,

$$v_{uvk}(\rho_{uvk}) = v_{\text{free}}(1 - \rho_{uvk}),$$

in meters per time step. The car advances by

$$x_{t+1} = x_t + v_{uvk}(\rho),$$

and when x_{t+1} exceeds the edge length L_{uvk} it reaches the target node and selects the next edge along its path. This extension makes congestion and queues live *inside* road segments instead of only at nodes, matching even more closely the Nagel–Schreckenberg model where vehicles occupy and move between discrete cells.

This rule removes the global magic number 5 and replaces it with a density dependent move probability that varies by edge. A short residential street with small capacity will reach high density with only a few cars and will slow down quickly. A long arterial road will have a much higher capacity and can accommodate more vehicles before speeds drop significantly.

3 Empirical Analysis

To analyze which roads become the most congested, the traffic simulation was run several times. As the initialization of the simulation is random, this accounts for a variety in initial setups. While in reality, traffic patterns might be less random every day as people do not choose their start (e.g. their home) and destination (e..g their workplace or supermarket) locations randomly, this will reveal those streets that are most prone to congestion given that we do not know the 'real' traffic patterns in Berlin and hence model them randomly. For this, we compute the average congestion of each road by cumulatively computing the density of the road over all n simulations and m steps in each simulation, so over $n \times m$ steps in total.

3.1 Establishing Convergence

To establish that the results for the mean levels of congestion have converged to a single value, we let the simulation run until from one run to the next, the relative change in mean congestion for *every* one of the 240 streets becomes less than 1% for 10 consecutive trials. This way, we are confident that the resulting mean values are 'stable' and not just an intermediary result in a simulation that would yield different mean values if run for longer.

Before running the simulation, we would expect that congestion depends mostly on the length of the street where shorter streets get more congested than longer ones. This is because the density ρ_{uvk} is determined as the ratio between the number of cars on a road to its capacity, $\rho_{uvk} = \frac{n_{uvk}}{C_{uvk}}$, whereby the capacity of small streets is reached very quickly. This means that cars will start to move along the short streets very slowly and create congestion. Additionally, we expect congestion to occur mostly in more 'central' areas where more of the randomly created routes are passing through. While there are some short routes on the outskirts of Berlin, these can only connect to either a starting or an end node, meaning that cars will either pass through these edges as their first or last step, which is unlikely to occur for a lot of cars at the same time.

Therefore, we expect high congestion to appear along those edges that are marked as 'high-risk' areas (red) on both plots below: while a high centrality (as proxy-measured by its degree) might imply that many cars will want to pass through that road, a low street length means that the capacity is reached quickly. Together, these create high density values and thus are expected to have high congestion.

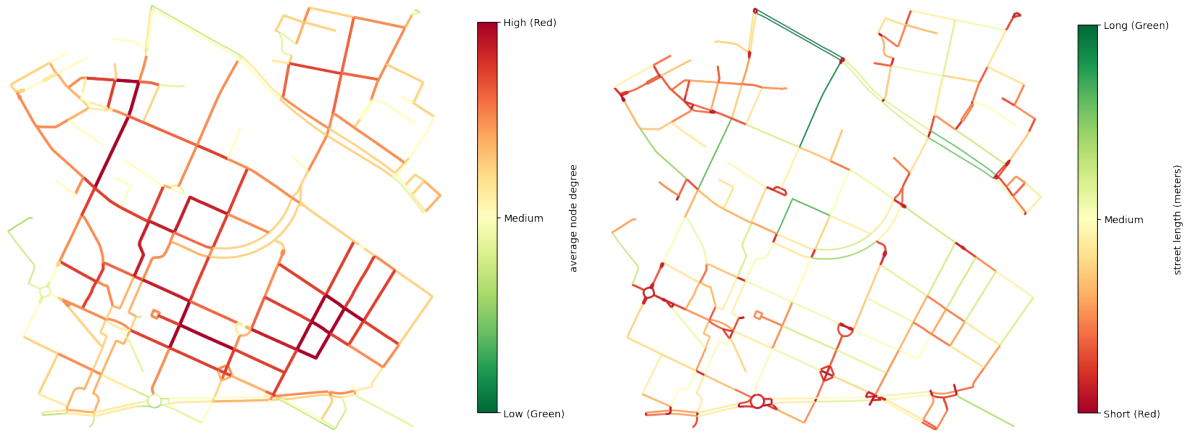

Figure 1: Left: Average Node Degree for the starting and end node of each edge showing higher degrees in central areas. **Right:** Street (Edge) Length. Roundabouts and large intersections constitute the shortest of the streets.

3.2 Empirical Results

First, the convergence check function was run on the Berlin network with 1,000 cars and a maximum of 100 steps for each simulation. Once all cars reach their destination, simulations are terminated even before reaching their maximum number of steps. While in reality, there are likely a lot more cars on Berlin's roads, `num_cars=1000` was chosen as an 'arbitrary' and number that was higher than the initial choice. The simulation had to be run $n = 960$ times until the maximum relative error was lower than 1% for 10 consecutive runs (Figure 2).

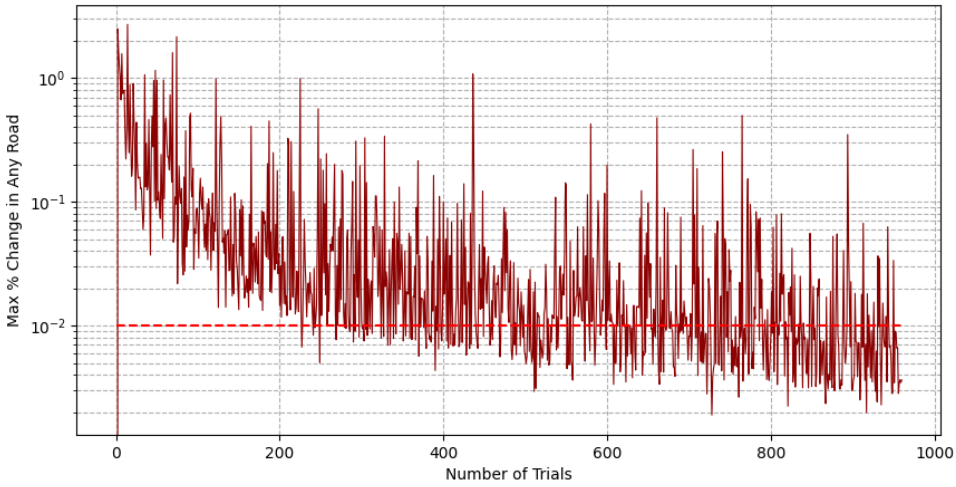

Figure 2: Output of the convergence check script showing the maximum percentage change in mean congestion for the Berlin road network. The percentage error is log-scaled and below 1% for 10 consecutive trials at $n = 960$ trials.

Finally, we can extract the names of the streets that have the highest average levels of congestion over the simulated trials. The table below shows the top 15 mostly highly congested streets. It is noticeable that some streets appear multiple times, e.g. "Kotbusser Tor" appears 5 times, amd "Waldemarstraße", "Inselstraße" as well as "Bethaniendamm" appear twice. This is because these streets are divided into different 'segments' (or edges).

To visualize the congestion on the network, a heatmap was created that assigns colors based on the average congestion relative to the maximum observed level of average congestion (Kottbusser Tor: 0.214406). Numbered labels showing the rank of that particular street (edge) were added. This shows that a lot of

Rank	key	Name	Avg Density	Length
1	29217327	Kottbusser Tor	0.214406	3.208236
2	29217332	Kottbusser Tor	0.213656	6.907021
3	11806567841	Kottbusser Tor	0.213385	15.797159
4	3352479275	Köpenicker Straße	0.202677	24.595416
5	29218291	Waldemarstraße	0.197713	157.299920
6	29276750	Oranienstraße	0.189583	10.745022
7	29217321	Kottbusser Tor	0.183177	27.314989
8	268466879	Kottbusser Tor	0.182698	26.118733
9	3463840860	Inselstraße	0.180854	4.284425
10	29273065	An der Schillingbrücke	0.170896	34.915946
11	29276213	Bethaniendamm	0.162326	85.773472
12	29218295	Waldemarstraße	0.158279	42.929962
13	28794518	Inselstraße	0.149698	7.422517
14	318545593	Wiener Straße	0.149104	25.728043
15	29276209	Bethaniendamm	0.148685	69.485601

Table 1: Top 15 streets ranked by average congestion.

congestion happens around the large roundabout at the bottom of the map ("Kottbusser Tor"), while most of the other labeled street segments are in central areas, too. It is also noticeable that the vast majority of the segments are very short (with the exception of ranks 5, 11, and 15)

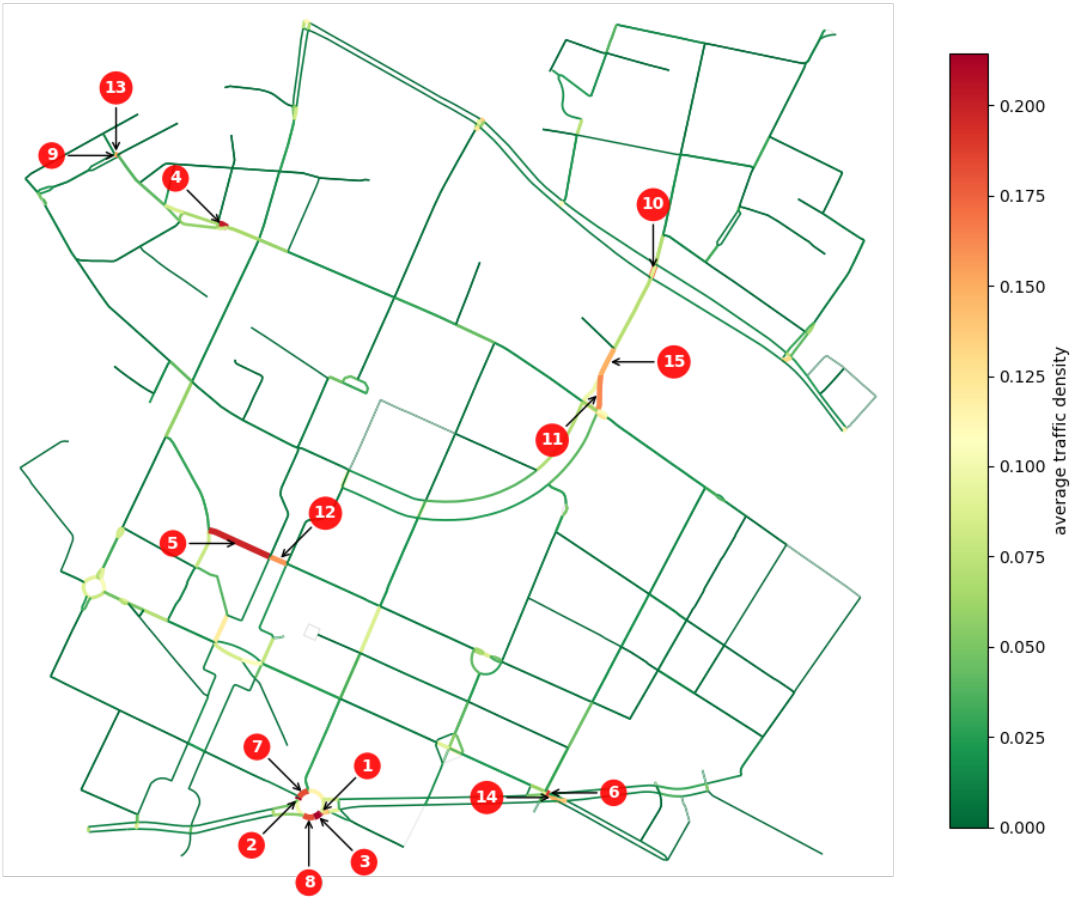

Figure 3: Heatmap of the converged average congestion over $n = 960$ trials. Labels were added for the top 15 most congested streets.

4 Theoretical Analysis

The empirical results match our expected results to some extent. However, some roads *surprisingly* appear in the list of the most congested roads despite not having a high degree. This suggests that taking the degree of a node as a metric might not be the most accurate predictive centrality metric.

4.1 Choosing a metric

Instead, it is likely more plausible to consider measures of **between-ness centrality** instead, as these describe how many of the shortest paths between any two nodes i and j run along a specific road.

In our model, congestion is modeled based on the density of cars in a specific street, relative to that street's maximum capacity, which depends on its length ℓ. In other words, for a specific street *uvk*,

$$\text{Congestion}_{\text{uvk}} \approx \text{Car Density}_{\text{uvk}} = \frac{\text{Car Demand for Street}_{\text{uvk}}}{\text{Street Capacity}_{\text{uvk}}}.$$

As the randomly initialized routes between the start and destination nodes are chosen using the `shortest_path` method, we can model the average demand of cars wanting to enter road uvk through the *edge betweenness*, let us call it ϵ_{uvk}. This is because in our model, ϵ_{uvk} represents the number of cars that will pass through a specific edge at some point of the simulation. It is simply computed as the sum of all node pairs' (i, j) shortest paths passing through node uvk (NetworkX Documentation, 2024):

$$\epsilon_{\text{uvk}} = \sum_{i,j \in N} \sigma(i, j|\text{uvk}).$$

While this is an average estimate because it does not accurately model the density at each specific simulation step, it is a useful proxy for how busy the street gets throughout the entire day. So,

$$\text{Congestion}_{\text{uvk}} \approx \frac{\text{Betweenness Centrality}_{\text{uvk}}}{\text{Street Capacity}_{\text{uvk}}} = \frac{\sum_{i,j \in N} \sigma(i, j|\text{uvk})}{C_{\text{uvk}}}.$$

This suggests that the chosen way of modeling congestion can be predicted by dividing an edge's betweenness-centrality by its capacity (street length). Because of the physical analogy, let us proceed to call this metric the **"Pressure"**. The difference between solely plotting ϵ versus plotting the capacity-adjusted pressure metric can be seen below.

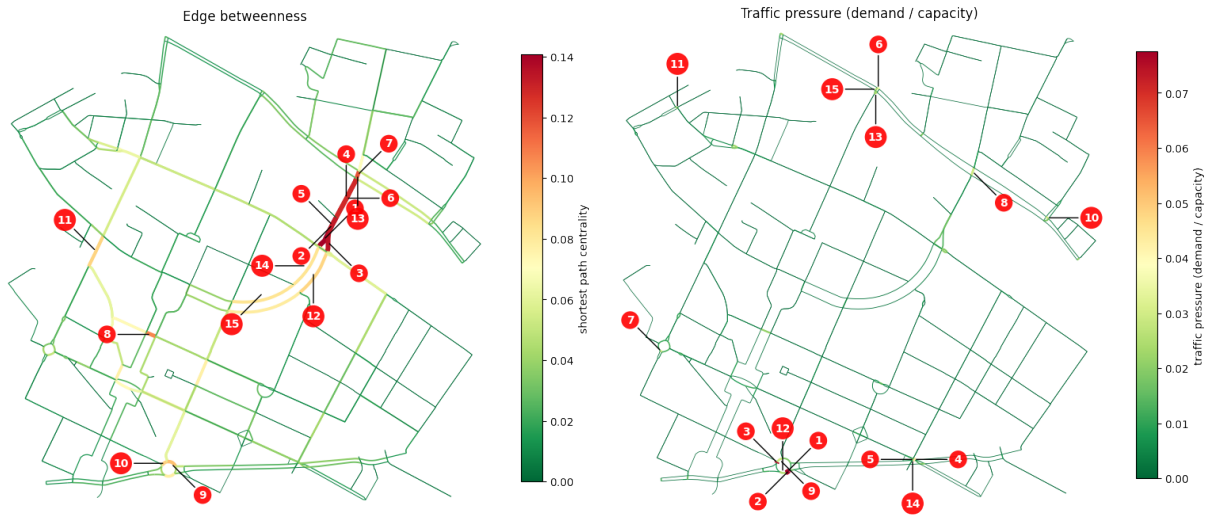

Figure 4: Left: Heatmap for the "Edge Betweenness Centrality" (ϵ). The 15 edges with the highest ϵ are labeled. **Right:** Heatmap for the adapted metric of "traffic pressure" (demand divided by capacity) with the 15 edges with the highest values being labeled.

4.2 Comparing theoretical to empirical results

Comparing these plots with our empirical results in Figure 3, we become more confident that this metric can help predict the congestion. In the table below, in the fourth column, we show the predicted congestion rank, and also show whether the new metric predicts a specific edge to be in the empirically tested top 15 (column 6). While the new metric correctly predicts 10 of the 15 to be part of the top-15, and the first three streets are predicted in the same order in both approaches, some streets are not well-predicted. Particularly, "Waldemarstraße" is ranked fifth in the simulation, but placed on rank 145 by the algorithm.

This discrepancy highlights a sensitivity bias in density-based metrics towards extremely short edges. In the simulation, short segments like "Kottbusser Tor" ($L \approx 3.2$m) have a capacity of effectively one vehicle ($C \approx 1$). A single car traversing this link instantly drives the density to $\rho = 1.0$ (saturation). Since the simulation imposes a minimum travel time of one time-step, these short links spend a disproportionate amount of time at maximum density, dominating the top rankings. At the same time, longer streets like "Waldemarstraße" ($L \approx 157$m) are "penalized" by the pressure metric ($P \propto 1/L$), which dilutes their high traffic demand over a longer physical space, resulting in a lower theoretical rank despite significant empirical congestion.

Name	Avg Density	Emp. Rank	Pred. Rank	Rank (ϵ)	In Top 15
Kottbusser Tor	0.214406	1	1	21	True
Kottbusser Tor	0.213656	2	2	25	True
Kottbusser Tor	0.213385	3	3	20	True
Köpenicker Straße	0.202677	4	13	30	True
Waldemarstraße	0.197713	5	145	26	False
Oranienstraße	0.189583	6	5	84	True
Kottbusser Tor	0.183177	7	9	10	True
Kottbusser Tor	0.182698	8	12	19	True
Inselstraße	0.180854	9	6	140	True
An der Schillingbrücke	0.170896	10	16	13	False
Bethaniendamm	0.162326	11	35	3	False
Waldemarstraße	0.158279	12	15	8	True
Inselstraße	0.149698	13	14	199	True
Wiener Straße	0.149104	14	38	90	False
Bethaniendamm	0.148685	15	27	5	False

Table 2: Comparing theoretical and empirical congestion ranking.

In order to use this metric for prediction-making, a relationship between the 'traffic pressure' and the average congestion should be determined. While plotting both metrics against each other on a linear plot reveals no easy-to-determine relationship, plotting them on a log-log plot shows a very linear relationship. This suggests that the relationship follows a **Power Law**. Running a linear regression on the log-log plot results in a coefficient of 1.0632, which translates to an exponent of 1.0632. This yields the proportionality:

$$\text{Congestion}_{uvk} \propto (\text{Pressure}_{\text{uvk}})^{1.0632}.$$

The exponent is very close to 1.0, which would signify a linear relationship whereby a doubling of the betwenness centrality would result in the doubling of the average congestion. However, as the exponent exceeds 1, we have a 'compounding' effect which occurs as the 'pressure' metric increases. This is plausible as cars slow down on congested roads, meaning they stay on these roads longer than they would have if the road was empty. At the same time, every additional car on a congested road also slows down *all* of the other cars, meaning that the density estimates on those roads are 'inflated'. So the congestion grows *faster* than the betweenness-centrality would predict alone. We observed a strong coefficient of determination ($R^2 \approx 0.7976$) and a high Spearman rank correlation ($\rho = 0.9766$). The high R^2 provides further evidence that the relationship between Pressure and Density follows a power law, supporting our derivation.

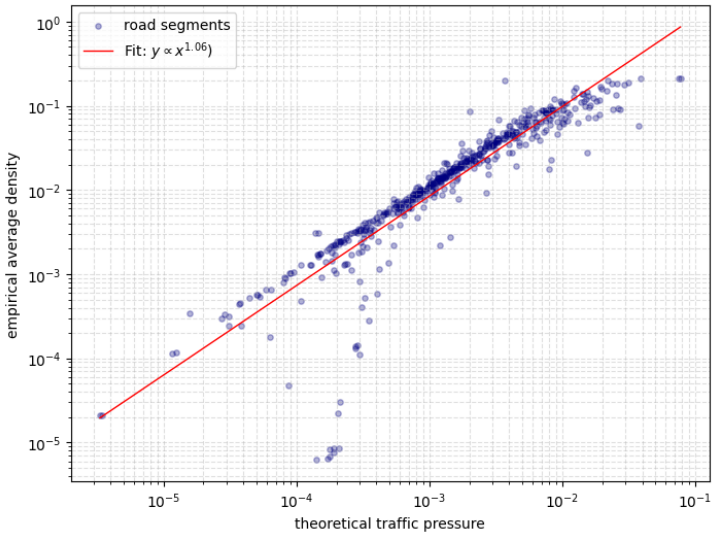

Figure 5: Data versus predicted average congestion values for the linear relationship visible on the log-log plot. While there are a few data points whose average density is a lot lower than their predicted values, the bulk of points are captured well.

4.3 Predicting congestion

Using the functions created for Berlin to analyze the road network empirically and theoretically, let us compare how well the relationship

$$\text{Congestion}_{uvk} \propto (\text{Pressure}_{\text{uvk}})^{1.0632}$$

can predict the congestion of an unseen network in Buenos Aires Capital, Buenos Aires Province, Argentina. Running the empirical simulation on the Buenos Aires street networks successfully converges after only 384 simulation runs. The plot below shows the top 10 high-congestion areas identified by the simulation.

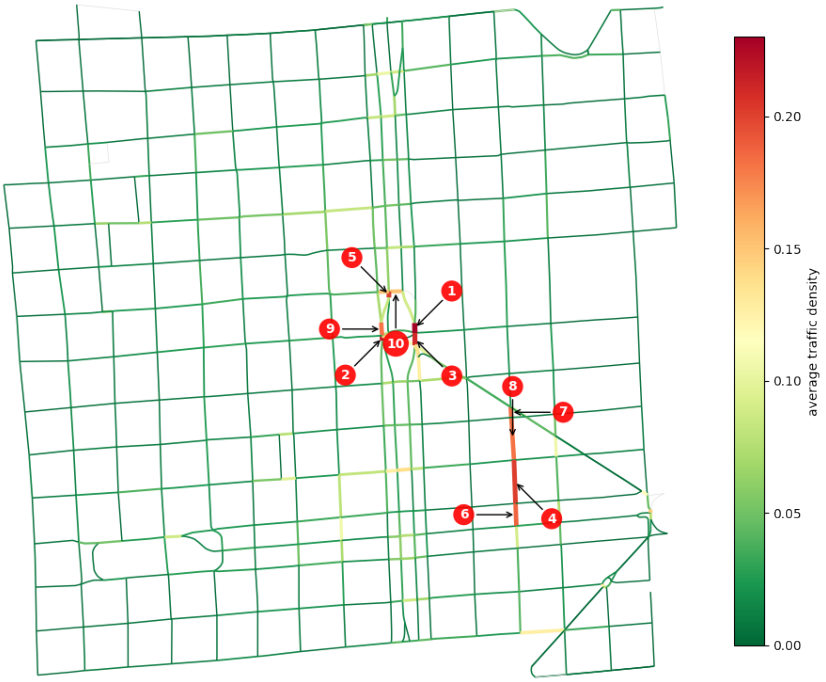

Figure 6: Heatmap for the adapted metric of "traffic pressure" (demand divided by capacity) with the 15 edges with the highest values being labeled.

Let us now compare how well the empirical results can be predicted with our theoretical derivation by plotting the theoretical results against the empirical average density. Figure 7 shows that our power law is a great fit for this relationship and that, again, the bulk of the results are clustered around the linear regression line on the log-log plot. When applied to a new city without re-calibration, the R^2 even increases slightly to 0.9213, likely due to the higher level of homogeneity between street lengths in this network. This is still a strong estimate showing that the power law applies to this road network, too. Similarly, the Spearman correlation remained robust at $\rho = 0.98$. This indicates that while the absolute density values may shift, the relative ranking of congestion (Spearman) remains predictable.

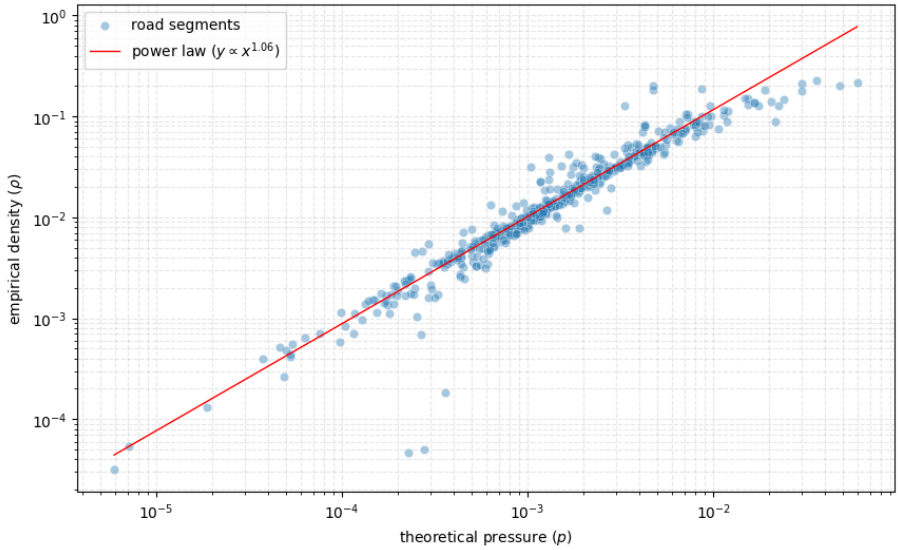

Figure 7: Comparing the predicted average density with the empirical average density. The results are very well predicted with the power law relationship determined in the Berlin network.

In summary, this shows that in our simplified model, we can successfully use the network metric of "edge-betweenness-centrality" alongside the length of the edge to predict the level of congestion. So, regardless of which city's road network we are modeling, the length-adjusted centrality of an edge measured by how many 'shortest paths' include it can predict its level of congestion. These roads could be targeted to add more lanes, or to inform navigation systems to recommend drivers to avoid these specific streets.

5 References

Boeing, G. (n.d.). *OSMnx documentation*. Retrieved December 7, 2025, from https://osmnx.readthedocs.io/en/stable/

Deutsche Welle. (2014, October 14). The cost of traffic jams. Dw.com; Deutsche Welle. https://www.dw.com/en/traffic-jams-a-severe-drain-on-economies/a-17994844

Munoz, J. F. (2024, February 10). Average Vehicle Size In The US And Europe Is Larger Than Ever. Motor1.com; Motor1.com. https://www.motor1.com/news/707996/vehicles-larger-than-ever-usa-europe/

Nagel, K., & Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. Journal de Physique I, 2(12), 2221–2229. https://doi.org/10.1051/jp1:1992277

NetworkX Documentation. (2024). edge_betweenness_centrality — NetworkX 3.4.2 documentation. Networkx.org. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.edge_betweenness_centrality.html

6 Acknowledgments

Class notes from Session 9 were used to implement elements of the Nagel-Schreckenberg model into this simulation.

AI was used only to format tables in Latex: https://chatgpt.com/share/6935bdc1-7ba4-8012-b276-b55a7f714cd0