Assignment 3

Network Model Generation

EPA133a Advanced Simulation

Group 16

Rachel Delvin Sutiono, No. 6284736

Celia Martínez Sillero, No. 6222102

Daniela Ríos Mora, No. 6275486

Thunchanok Phutthaphaiboon, No. 6141153

Yao Wang, No. 6157513

Outline

1. Introduction

This assignment focuses on network model generation by integrating Mesa and NetworkX to simulate goods transportation across Bangladesh's road network. Expanding on our previous work, we model goods transport along N1, N2, and their major side roads, automatically generating road and bridge components from the infrastructure data. Through this assignment, we aim to build a transport simulation that provides valuable insights into the effects of infrastructure failures on traffic flow.

We begin by creating a demo file for model input (Chapter 2) and then implement NetworkX for shortest path routing (Chapter 3). Next, we modify the simulation components to allow for random origin-destination pairs (Chapter 4). We then run experiments to analyze the impact of different bridge failure scenarios on travel time and discuss the results (Chapter 5). The bonus questions are answered in Chapter 6. Finally, we reflect on model limitations, possible improvements, and additional insights (Chapter 7).

2. Demo File Creation

This chapter is to explain step-by-step on how we clean and combine bridge and road data to create a well-structured dataset (demo_with_intersection.csv) for simulation. The final dataset should have a specific structure like the example demo file (demo-4.csv). The source code for this process can be found in "EPA133a-G16-A3\notebook\CreateDemoFile.ipynb". Steps taken are as follows:

1. **Load data**

   We started by loading the bridge data from BMMS_overview.xlsx and road data from _roads3.csv. These files contain the necessary information for the simulation, including details about roads, bridges, and their conditions.

2. **Remove duplicates from BMMS_overview.xlsx**

   This part is where we made an assumption regarding L and R bridges. Although this might not perfectly reflect reality, for this assignment we assumed that there is one single bridge at a location reference point (LRP) to make the analysis more simple while keeping the most important aspect of the study, which is the overall effect of infrastructure damage on traveling time. If there are more than one bridge at a LRP, we decided to keep one of the 'left (L)' bridges that are in the worst condition because we want to account for the most vulnerable points in the network, which are more likely to cause delays in travel time.

3. **Remove duplicates from _roads3.csv**

   We then cleaned the road dataset to remove any duplicate values by checking road_chainage, road_lrp columns and by manually correcting certain duplicate values and removing redundant entries. Please note that we did not assign the model_type 'bridge' to all LRP ending in a letter (excluding E and S)" as mentioned in the Assignment 2's report feedback.

4. **Combine data from BMMS_overview.xlsx & _roads3.csv**

   Next, we merged the cleaned bridge and road datasets into one structured dataset. At this step, all relevant data is in a single, structured DataFrame, making it easier to be handled in the future steps.

5. **Build the demo data frame with 'sourcesink'**

   We structured the dataset to match the given 'demo-4.csv' format and set model_type = 'sourcesink' at the start and the end of each road.

6. **Filter only N1, N2, and their side roads (N1xx, N2xx) that are longer than 25 km**

   Following the assignment's instructions, we kept only major roads: N1, N2, and their 7 side roads (N1xx, N2xx) that are longer than 25 km. Ultimately, there are 9 roads: N1, N2, N102, N104, N105, N106, N204, N207, and N208.

7. **Define intersections**

   To identify intersections in the road network data, we began by visually inspecting the plot, identifying 13 intersections (13 pairs of LRP). Using the Haversine distance, we calculated, for each road LRP ('id' in 'road'), the nearest other road's 'id' and their respective distance, storing the shortest distance values and the corresponding ‘id’ in the new columns: ‘nearest_road’ and ‘distance_to_nearest_road’. This process was limited to ‘model_type’: link.

The data was then sorted in ascending order by ‘distance_to_nearest_road’, and the closest 50 pairs were extracted and plotted. 50 was chosen to make sure no intersections are overlooked. This was then confirmed by plotting them. However, the N106-N1 intersection was not detected due to an excessively large distance. To address this, we manually computed the distance between these specific roads using a “brute-force” algorithm and appended the first identified pair to the previous set. Duplicates, where the same two roads were paired, were dropped while entries with the smallest distance were kept. Single LRP (with no pair) were also removed.

The ‘model_type’ was then updated from ‘link’ to ‘intersection’. To meet the NetworkX requirement for intersections, the ‘lat’, ‘lon’, and ‘id’ are adjusted so that each pair shares the same value or name. The intersection new ‘id’ was generated by adding their former ‘id’ string. Finally, since the index remained unchanged from the original dataframe, the intersection data was easily merged back by replacing the corresponding rows in the main dataframe.

8. **Save demo file as a csv**

We saved the cleaned and structured dataset as ‘demo_with_intersection.csv’, making it ready to be used in the NetworkX construction and goods transportation simulation.

9. **Plot roads**

To confirm that everything was processed correctly, we plotted the roads and bridges to visually inspect the dataset (Figure 2.1 and 2.2). This visualization helps make sure that the dataset is ready for simulation and free of major errors.

Through these steps, we created a structured dataset based on the assignment and that meets the simulation’s requirements.

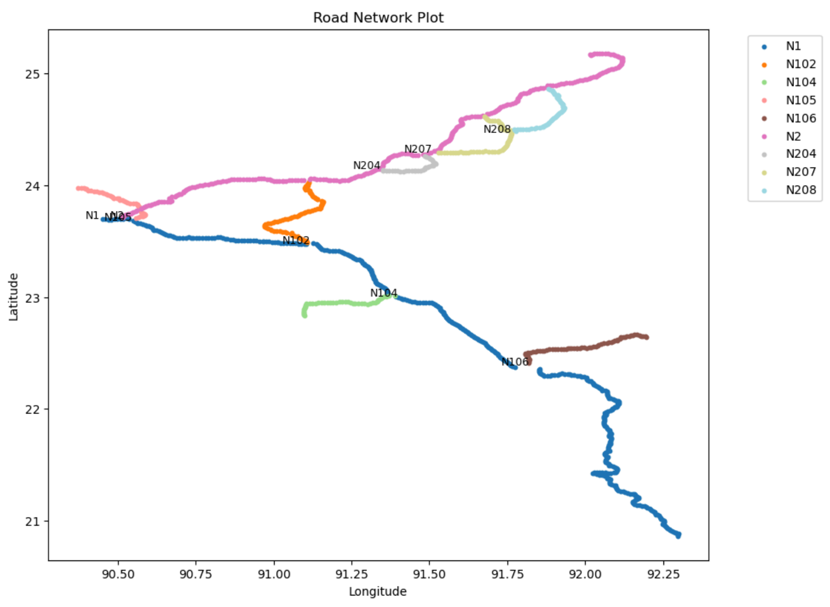

Figure 2.1: Infrastructure plot based on road numbers

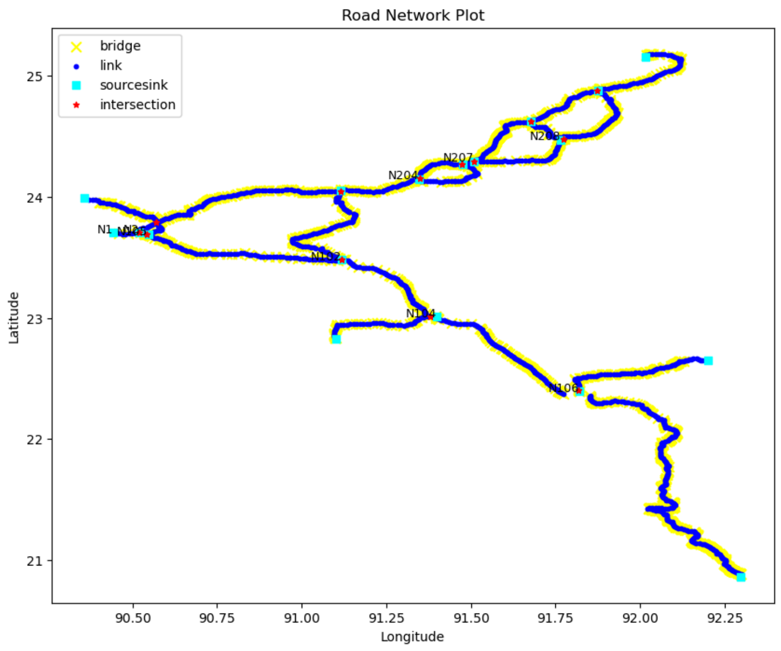

Figure 2.2: Infrastructure plot based on types of components

3. NetworkX and Shortest Path

3.1 Data Requirement for NetworkX

The dataset used for creating the network follows a structured format to ensure proper connectivity and sequencing of road segments. The key requirements for the data are as follows:

- All `id` values for each road must follow a sequential order based on `lrps`, ensuring the network is generated correctly by linking nodes within the same road until a new road begins.
- The start and end points of each road must have `model_type` set as "sourcesink" to define entry and exit points.
- Intersections between roads must share the same `id` to allow the network to detect and connect intersection points properly.

3.2 Network Construction (WIP)

The current code leverages the original provided by the course staff. The network is created in model.py. The graph is initialized in the first lines of the generate model function. It creates a non-directed, weighted graph with coordinates.

After creating the dictionary path, we set the nodes and edges. We decided to store per node relevant attributes like their ID, latitude and longitude (position),and the model_type. The weight used is the distance between Infrastructure objects provided by the input data file commented in previous sections. Despite not being critical for the sake of the code, we opted for a graph with coordinates to provide a more insightful representation in Figure 3.2.1. Thereby, it can be clearly observed N1,N2 and its secondary roads as in real-life. If the coordinates were not added, NetworkX would be still able to calculate the shortest path but the representation would not reflect real-life spatial distribution.

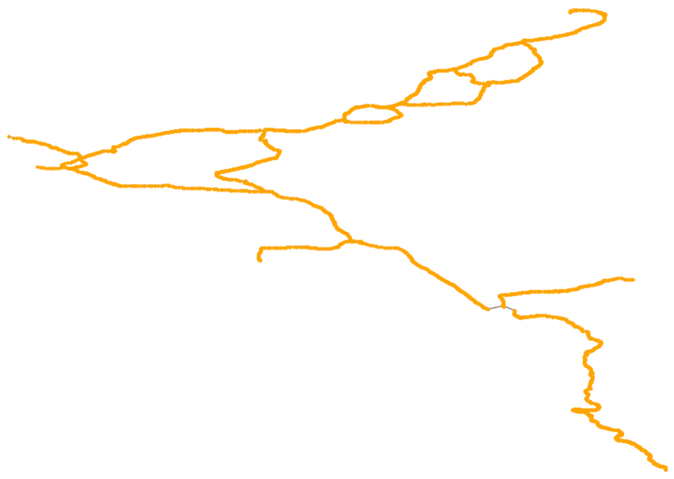

Figure 3.2.1: Network plot

3.3 Shortest Path

In this assignment, the logic to calculate the shortest path goes as follows:

1. A source generates an instance of class Vehicle and calls the Vehicle class's *set_path()* function.
2. Set path function call model's *get_route()* function
3. Model class's *get_route()* function randomly decides between the two route methods on this assignment: straight and random routes. There is 99,9% chance that the function selects the random route method
    a. Straight route: The vehicle will have as *Sink* the end of the road where it was generated. Thereby, the source and sink are on the same road.
    b. Random route: It selects a random *sink* instance in the overall network with the only condition that it cannot be the same as the source the *Vehicle* instance was generated by.
4. Both methods call the *compute_shortest_path_if_needed* function. Its inputs are the *vehicle* instance's source and previously assigned sink. This function only calculates the shortest path and distance if the dictionary path model attribute retrieves a value other than None when using the key (*Source, Sink*). This has been set this way to save the program from repeating unnecessary calculations. The *compute_shortest_path_if_needed* function's outputs are the calculated shortest path and distance or None. If calculated, once a given path is optimized, it is saved in the dictionary so it can be easily retrieved if it takes place after. On the other hand, the shortest path distance is appended in the model's attribute *driving_distance.*

3.4 Distance and Effective Speed

To calculate the average effective traveling speed (traveling distance / experiment traveling time) across different scenarios, we need to determine the distance and driving time for each vehicle. The process follows these steps:

1. Compute distance using NetworkX: When a vehicle is created, determine its shortest path and calculate the total distance based on road segment lengths and store it.
2. Track driving time: Record the time each vehicle starts and when it reaches its destination, storing the values.
3. Calculate speed and averages: Compute the speed of each vehicle using its total driving distance and driving time. Store individual speeds, calculate the overall average speed of all vehicles, and compare results across different scenarios.

4. Model and Component Modification

We built upon the model from assignment 2. Below are more detailed description of changes and additions we made for network model generation

4.1 BangladeshModel(Model)

Attributes

1. **self.sources:** List of all SourceSink components that will eventually be sources.
2. **self.sinks**: List of all SourceSink components that will eventually be sinks.
3. **self.condition_list:** List with the categories of the broken bridges
4. Self.driving_times: dictionary to store diving time for each truck id reaching its sink
5. self.driving_distances: dictionary to store distance traveled by each truck id reaching its sink

Methods

1. ***'generate_model'*** which reads a CSV file containing infrastructure components is slightly modified.
    a. It reads the new 'id' format, which contains the road and ID numbers, separated by an underscore (RoadName_IDNumber).
    b. Instead of reading default values for the roads, it stores the unique values from the road column.
    c. The attribute 'condition', is included when creating the Bridge agent.
2. ***'compute_shortest_path_if_needed'*** computes the shortest path between a source and sink using NetworkX, storing results to avoid redundant calculations. If a path was previously computed, it retrieves it from path_ids_dict; otherwise, it calculates and saves a new one. Calculate the distance of the shortest path for each vehicle and store.
3. ***'get_route'*** determines a vehicle's route. Currently it's set to always get_random_route: selecting a random sink. It retrieves or computes the shortest path as needed.
4. ***'get_straight_route'*** returns the shortest direct route from the given origin. ***'get_random_route'*** selects a random sink different from the source and computes the shortest path to it if not already stored.
5. ***'get_average_effective_speed'*** calculates the average effective speed for trucks reaching its sink.

4.2. components.py

4.2.1. Vehicle(Infra)

Speed is still set to 48 km/h, as the assignment requires.

Attributes

1. **self.sink:** stores the sink to where the truck is headed
2. **self.source**: stores the source the truck was generated
3. **self.distance_traveled**: stores the distance traveled by the truck

Method

1. '*drive_to_next*' which manages vehicle DRIVE and WAIT state behavior when reaching a new infrastructure. Now, we added a logic allowing vehicles to wait at broken bridges and track their delay time:
   a. Added a *while True* loop replacing the former recursion logic to avoid stack overflow and iteratively move through multiple infrastructure elements in a single function call.
   b. Delay time calculations only happen when the vehicle first reaches a broken bridge. If the bridge is broken, It checks if '*self.waiting_time == 0*' before assigning a delay to prevent overwrites, then computes and assigns a random wait time.
      ```
      if next_infra.unique_id in self.model.broken_bridges:
              if self.waiting_time == 0:
                      self.waiting_time = next_infra.get_delay_time()
      ```
   c. Added a bridge delay accumulation function which tracks delays per bridge, ensuring that each broken bridge's delay is only recorded once per vehicle. It updates and stores delays in '*self.model.bridge_delays*'.
      ```
      if next_infra.unique_id not in self.model.bridge_delays:
              self.model.bridge_delays[next_infra.unique_id] = 0
      self.model.bridge_delays[next_infra.unique_id]+=self.waiting_ti
      me
      ```
   d. A variable was added to store the index of the destination SourceSink, which is assigned to the variable end. Then, if the next instance is a SourceSink, its unique_id is compared with the stored destination. If they match, the truck has reached its final destination and is removed.
   e. Added lines to store driving time and distance of each truck when it reach its sink
      ```
      driving_time = self.removed_at_step - self.generated_at_step
      self.model.driving_times[self.unique_id] = driving_time
      Self.model.driving_distances[self.unique_id] = self.distance
              _traveled
      ```
   f. Added equations to update and keep track the traveled distance on every tick
      - `self.distance_traveled += distance`
      - `self.distance_traveled += next_infra.length`

4.2.2. SourceSink(Source,Sink)

This class is a child class of Source and Sink, so it can use the same methods from both parent classes. The methods were not changed.

5. Experiment Setup

This script model_run.py runs traffic simulations using the BangladeshModel under five different scenarios, each with varying probabilities for four categories (A, B, C, D). The simulation runs for 5 days (7200 minutes per run) with 10 replications per scenario, using different random seeds.

Each scenario is repeated using different random seeds. These seeds are the same generated across all scenarios to maintain consistency across replications.

During the simulation, key metrics are recorded, including average driving time, total and average waiting time, road condition categories, broken bridges, and truck speeds. These results provide insights into how different conditions affect transportation efficiency.

Finally, the collected data is stored in CSV files, with one file per scenario (`scenario0.csv` to `scenario4.csv`).

6. Analysis

In the previous assignment, we simulated trucks following a fixed route, allowing for a controlled analysis of travel time under specific conditions. However, in this assignment, the simulation covers multiple roads across Bangladesh, with randomly assigned origins (sources), destinations (sinks), and shortest paths for each truck. This means that every truck encounters different road conditions, varying numbers of bridges, and diverse travel distances, leading to different travel times across runs.

Given these uncertainties, a straightforward comparison of average delay times alone may not accurately reflect the true impact of infrastructure damage on the transportation network. To address this, we conducted a deeper analysis by

- **Infrastructure analysis:** assessing the number of bridges and overall condition of each road to understand the infrastructure quality in different locations
- **Scenario analysis:** comparing the average effective speed (traveling distance / traveling time) among scenarios to understand how damaged infrastructures affect traffic flows in Bangladesh. This provides a more standardized way to measure the impact of infrastructure damage, since speed inherently accounts for both distance and delay. We will define it in more detail in section 6.2.

6.1 Infrastructure Analysis

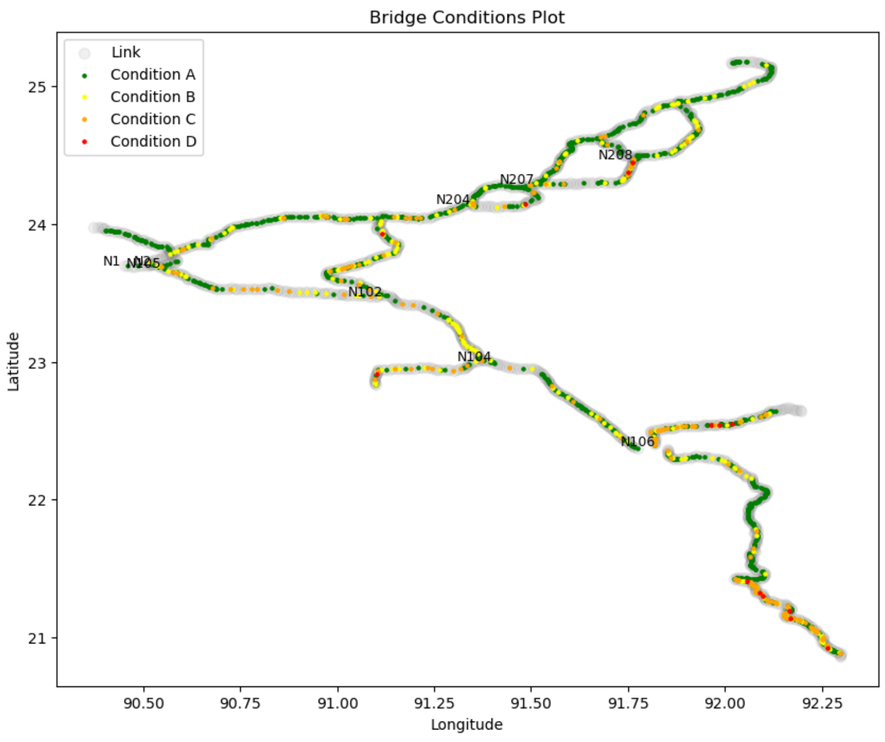

Figure 6.1.1: Bridge condition plot

Figure 6.1.1 shows the condition of bridges using different colors, making it easy to see which parts of the road network are more affected by infrastructure damage. Figure 6.1.2 further breaks this down by showing the number of bridges in each condition. Even though N1 and N2 are similar in length, N1 tends to have more damaged bridges, making it more vulnerable to delays. As a result, taking N1 might lead to longer travel times compared to N2, which is in better condition. Interestingly, this means that in some cases, choosing a longer route with no damaged bridges might be faster than taking a shorter road with many broken bridges. Thus, travel time is influenced by both the distance and the condition of the infrastructure. A shorter road isn't always the quickest if it has too many damaged bridges.

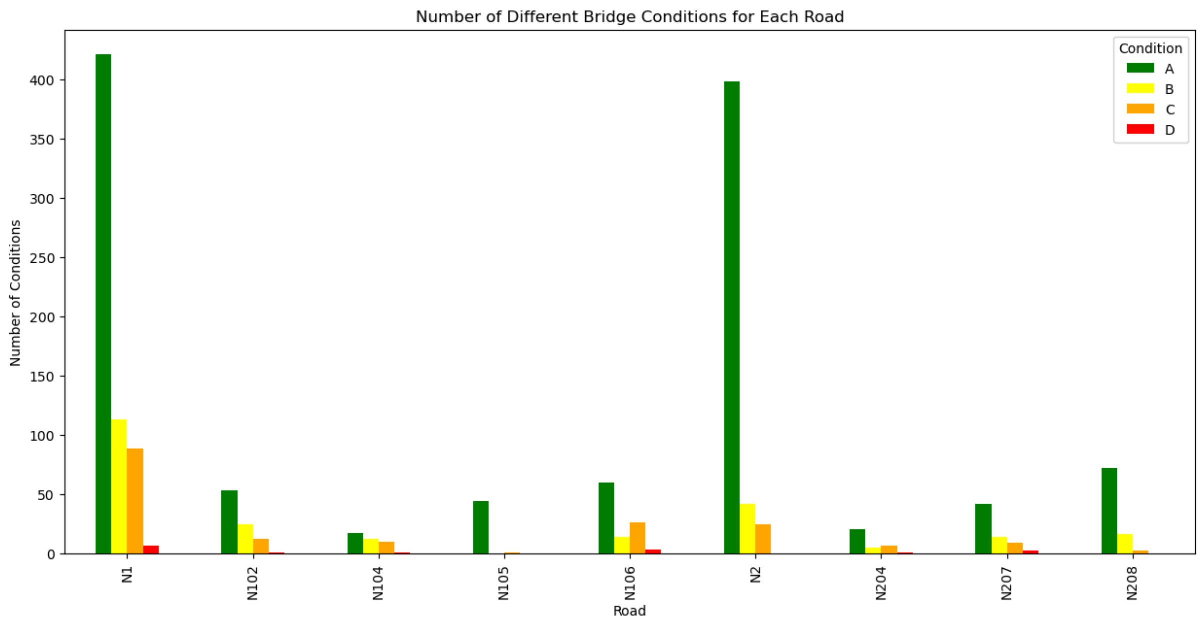

Figure 6.1.2: Number of different bridge conditions for each road

6.2 Scenario Analysis

The simulation results showcase the consequences of bridge failure probability. This effect can be studied by analysing the number of broken bridges, the average driving time and the Average effective speed.

Number of broken Bridges

As observed in assignment 2, the number of bridges increases as probability does across scenarios.

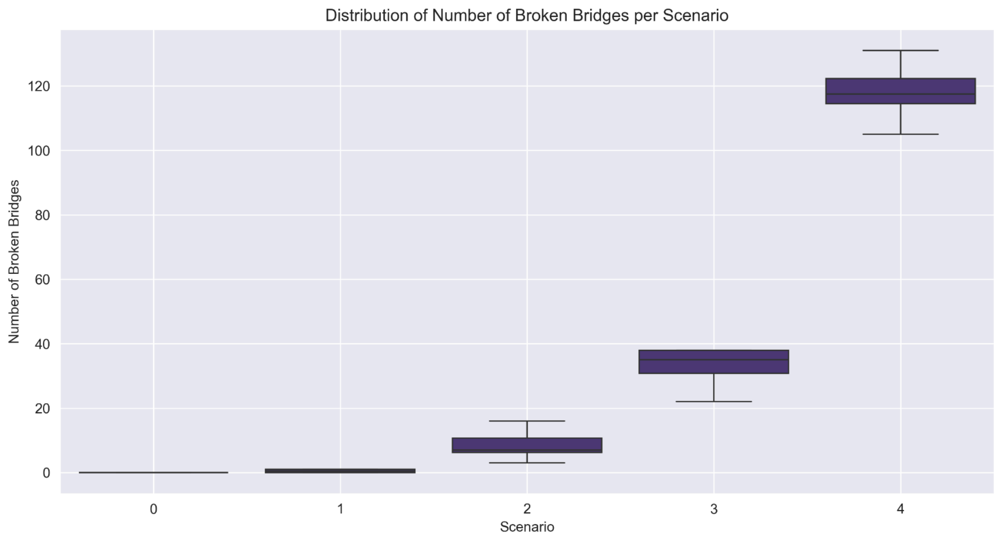

Figure 6.1.1: Distribution of Number of Broken Bridges

Average Driving Time

It is calculated by subtracting the removal time step minus the generation time step of the instance *Vehicle*.

We observe that the distribution of average driving time increases per scenario (Figure 6.2.1). Following our findings in assignment 2, this indicates that the network is highly sensitive to the number of bridges failing. In Figure 6.2.1, we plot the Average driving time per scenario. While Scenarios 0 and 1 show a short range, Scenarios 2 and 3 present an outlier and Scenarios 3 and 4 have larger ranges. This indicates that as more bridges are broken in later scenarios, the variability of the simulation increases, and each run can lead to a wider range of results.

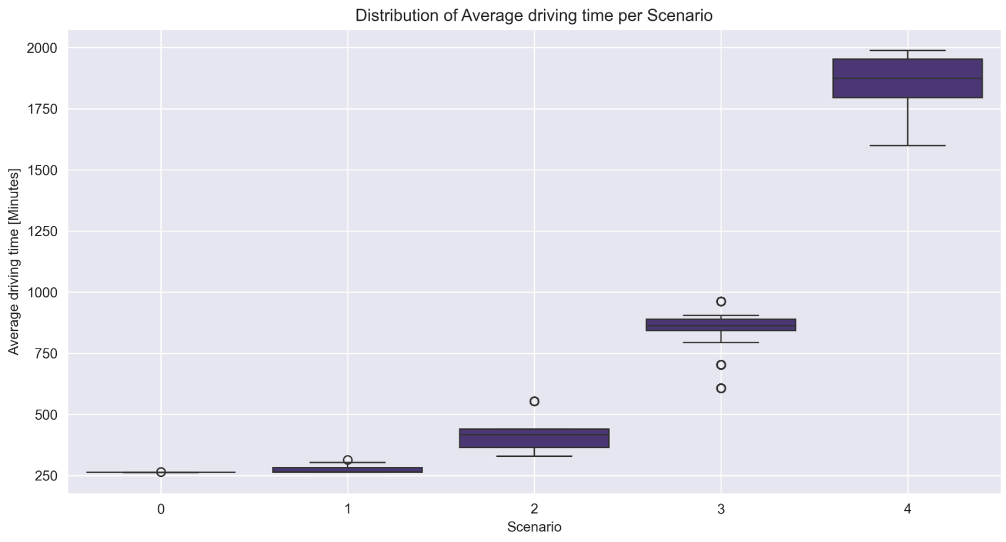

Figure 6.2.2: Average driving time distribution per scenario

Average Effective Speed

It is equal to the distance of the assigned shortest path divided by the travel time, which includes delay time caused by broken bridges. By effective we mean that it is a conceptual term. The Vehicle instances drive at 48 km/h or they are in WAIT state -therefore at 0 Km/h. By calculating the average effective speed, we can obtain a proxy to see how much has the majority of vehicle agents been affected by the random broken bridges per scenario. The average effective speed would also allow us to compare it with the initial determined speed, 48 km/h, giving measurable insights about how much delays from broken bridges would slow down trucks from reaching their destination. This added simulation output enables us to better comprehend and compare the results of each scenario.

The simulation shows that trucks are going *effectively* slower as more bridges are likely to be broken. Figure 6.2.2 reveals a sharp downward trend in the Average effective speed. Except for scenarios 2 and 3, the simulation outputs throughout the different replications are relatively short. This may imply that Scenarios 2 and 3 experience more variability, depending on whether the vehicles' routes encounter broken bridges. In Scenario 4, the likelihood of facing a broken bridge is markedly higher, affecting all replications. Additionally, we noted that in this scenario, many vehicles do not complete their routes within the 5-day runtime because of the delays encountered.

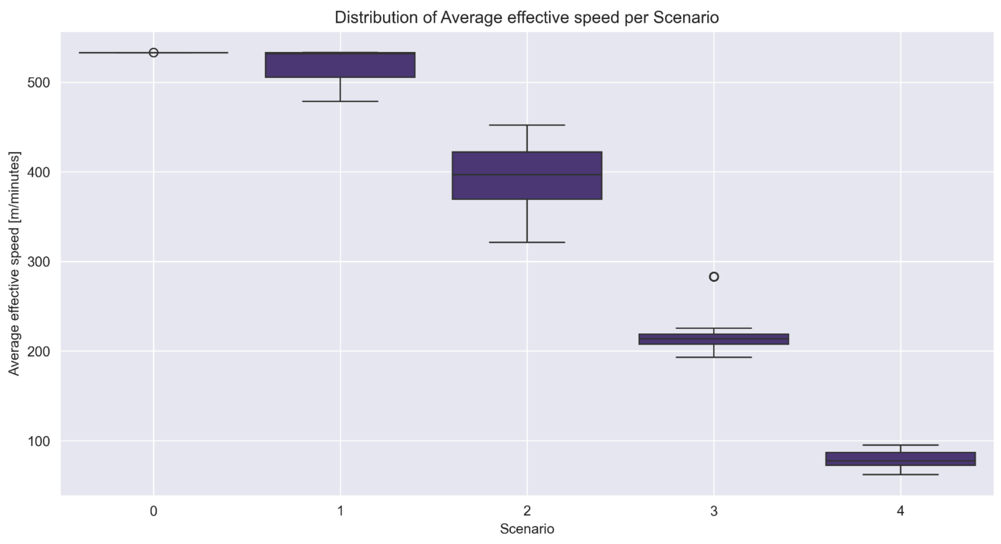

Figure 6.2.2: Average effective speed distribution per scenario

Key Takeaways

As the probability for bridges to suffer malfunctions increases, the network faces more disruptions and more critical. As stated before, scenario 4, the most disruptive, outputs all its replications with several vehicle agents not reaching their sink destinations by the end of the run time.

On the other hand, the variability of results across scenarios is different. In relatively medium probability - Scenario 2 and 3- the chance of disruption is subject to the stochasticity introduced by the random method to choose sink destinations. If fixed routes do not go through the broken bridges, the average drive time is significantly lower. This will be further discussed in chapter 7.

7. Bonus Exercises

1. **The latitude and longitude comparison can be seen on Table 7.1**
2. **Analysis:**

The difference in location accuracy impacts length, hence the driving time, since length is the only variable currently affecting the driving function, as well as the path ids sequence. An incorrect location of intersection may also misplace an intersection before a bridge on a certain road. If the bridge is broken, the truck does not encounter it and experiences no delay, whereas, in reality, it should have crossed the bridge before reaching the intersection. This inaccuracy could lead to underestimating delays and misrepresenting real-world conditions.

That being said, the impact of slightly inaccurate length wouldn't be significant on the simulation result as the average standard deviation is 0.28 km. The biggest contribution to that error is from the N1-N106 intersection, 1.46 km standard deviation. This was the consequence of the 'brute-force' intersection identification as the LRPs from both roads on the dataset are so far apart (Figure 7.1).

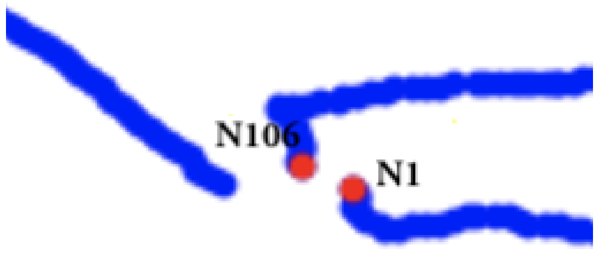

Figure 7.1. N1-N106 Intersection

Table 7.1: Coordinates comparison between the intersections we assigned in our analysis (experiment lat, lon), the intersection points in the given Java simulation, and estimated road intersections from Google Map.

Intersections	Experiment Lat, Lon	Java shapefile Lat, Lon	Google Maps Lat, Lon	Standard deviation (km)	Notes
N1-N106	22.402056, 91.819110	22.393, 91.822	22.369133, 91.833074	1.459	
N1-N104	23.009528, 91.380360	23.01, 91.381	23.009489, 91.381267	0.015	Not directly connected (roads are on different level)
N1-N102	23.478972, 91.117999	23.479, 91.118	23.479015, 91.118163	0.008	
N2-N102	24.042111, 91.113972	24.051, 91.115	24.050665, 91.114658	0.533	

Intersections	**Experiment Lat, Lon**	**Java shapefile Lat, Lon**	**Google Maps Lat, Lon**	**Standard deviation (km)**	**Notes**
N1-N105	23.691805, 90.542611	23.69, 90.547	23.690448, 90.546677	0.235	
N1-N2	23.706889, 90.520472	23.706, 90.521	23.706054, 90.521301	0.051	
N2-N105	23.789222, 90.568805	23.785, 90.569	23.785342, 90.568867	0.237	Not directly connected (roads are on different level)
N2-N204 (1)	24.153028, 91.348611	24.148, 91.347	24.147731, 91.346596	0.319	
N2-N204 (2)	24.267249, 91.474360	24.268, 91.477	24.267510, 91.477105	0.129	
N2-N207 (1)	24.295416, 91.510417	24.295, 91.51	24.294819, 91.510315	0.015	
N2-N207 (2)	24.623277, 91.678166	24.626, 91.678	24.626363, 91.677676	0.159	
N207-N208	24.475722, 91.772777	24.471, 91.766	24.471467, 91.765618	0.462	
N2-N208	24.877110, 91.872944	24.878, 91.875	24.877585, 91.875519	0.104	

8. Reflection and Limitations

Throughout our analysis, we identified several aspects that left some food for thought.

First, randomness does not guarantee the model better portrays reality. For example, in our current model, the function get*_random_route* is likely to give the same or more traffic to sources located either in remote villages or major cities. In real-life this combination is highly unlikely–trucks are more likely to depart from major cities. Therefore, reflecting on Jafino's work, this simulation tends to be an egalitarian approach, where weights are equally distributed. Despite its limitations, we can also argue that, since we don't have the data nor time to model such weights in a systematic way, using randomness prevents us from being led by our biases (Jafino et Al, 2021).

Secondly, we are not yet in the phase of analysis where we can determine the most critical bridges. The function *get_random_route()* introduces more stochasticity. Knowing this, we tried to introduce the *average* effective *speed* to provide further insight and mitigate the consequences of lack of replicability of the broken bridges and path variance per replication. Effective speed was calculated as a variable that is more comparable when analysed across the scenarios.

However, after numerous testing, we still can't realize this feature to its full potential. We hypothesize the error is coming from the use of list type as a model attribute to save the distance and time values from vehicle instances. We theorise that during the coding process there is a high chance that the indexes are not sufficiently coordinated. This leads to miscalculating the driving time per vehicle—mixing them up. It might also be the consequence of improper storing of paths and their length generated from NetworkX. We've tried to tackle these by creating dictionaries for driving time and distance to store the data for each truck id. This approach was promising but still needs improvements—or maybe an entirely different approach is needed.

The average effective speed our model currently returns is not accurate when being used to simulate the whole road network. 48 km/h is expected for scenario 0, but it's returning 32 km/h instead. On the other hand, when we tested it on only one road, for example on N1 alone, it returns 48 km/h for every replication in scenario 0. Nevertheless, we consider that it is still a useful variable for scenario comparison and analysis, since the resulting trend is replicably valid (even though the value per se is not).

Lastly, in the future it would be interesting to add more dynamism to the model. Currently, trucks can only stick to the route assigned. They must keep that route even if there are large delays due to broken bridges. In real-life, drivers will most likely try to opt for alternative routes the moment they know about the traffic situation. It would also be interesting to incorporate 'estimated_time' as a determining factor in the path selection to replicate truck drivers' accessibility to navigation services.

References

Jafino, B. A., Kwakkel, J., & Verbraeck, A. (2020).Transport network criticality metrics: a comparative analysis and a guideline for selection. Transport Reviews, 40(2), 241-264.

Acknowledgement

The use of AI

Use of AI in Code Development: In this assignment, AI tools such as GitHub Copilot and ChatGPT were used to improve code quality and generate ideas for implementation. These tools assist in structuring code, suggesting optimizations, and debugging issues. However, AI's role was strictly supportive—we conducted final decisions, implementations, and refinements to ensure accuracy and adherence to project requirements.

Use of AI in Writing and Editing: ChatGPT and Grammarly were used for text improvement, helping refine clarity, coherence, and grammatical correctness. It assisted in brainstorming alternative phrasings and ensuring correct writing standards. However, all AI-generated suggestions were critically reviewed, modified when necessary, and integrated into our work only after careful validation. This approach ensured that AI served as a tool rather than a replacement for our original analysis and insights.

Contribution of each member

Member	Contribution
Rachel Delvin Sutiono	- Demo file creation - Report - Experimentation and modeling
Celia Martínez Sillero	- NetworkX implementation - Report - Experimentation and modeling
Daniela Ríos Mora	- Experimentation and scenario setup - Report - Experimentation and modeling
Thunchanok Phutthaphaiboon	- Demo file creation - Report - Experimentation and modeling
Yao Wang	- NetworkX implementation - Report - Experimentation and modeling