EPA133a. Advanced Simulation
Assignment 3. Network Model Generation

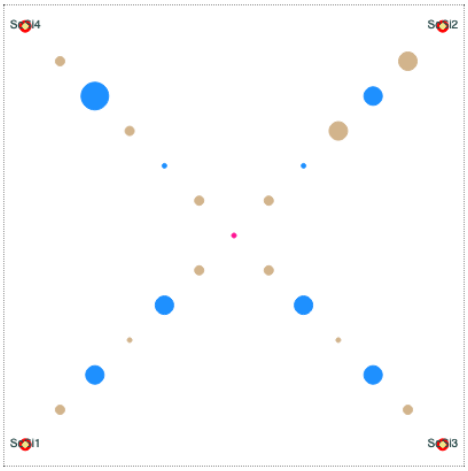
Introduction

In this Assignment, we will expand the model generation process we worked on in the previous assignment and see how a multi-modelling approach can help model generation. We will use Mesa 2.1.4 (https://mesa.readthedocs.io) and NetworkX (https://networkx.org/) in this exercise, automatically creating a Mesa model from data and overlaying it onto a NetworkX model. We will develop a small demo model of goods transport over N1 and N2 in Bangladesh, including the main side roads that are connected to N1 and N2.

Get started

The following instruction is written based on PyCharm CE 2020.3.3.

1. Download the "EPA133a-Gxx-A3.zip" file from BrightSpace. Unzip the file to the "EPA133a-Lab" folder you created for the last assignment. Rename "EPA1352-Gxx-A3" with your group number, e.g., "EPA133a-G01-A3".
2. Launch PyCharm. Open "EPA133a-Lab" as an existing project.
3. In the PyCharm Project window, click open the "EPA-133a-Lab>EPA133a-Gxx-A3>model" folder, right-click the "model_viz.py" file, and choose "Modify Run Configuration…".
4. Rename "model_viz" to e.g. "A3 model_viz"; click "Apply", then "OK".
5. Click the green Run arrow icon at the top right of the window.
6. A tab will open in your default browser. The demo model looks like this:



7. Now the demo simulation model visualisation is launched, and you can click the Start, Step, and Reset buttons (top right of the browser window) to see what the visualisation does. You can also use the slide bar to adjust the speed of the simulation. (You can also run "model_run.py" instead of "model_viz.py". It launches simulation without visualisation.)

In this Mesa simulation demo (demo-4), vehicles (trucks) are generated every 5 ticks at each corner node (red/green dot) by a component named SourceSink. The vehicles drive straight through the links and bridges (links visualised in beige dots, bridges in blue), and end their trips at a corner node (red/green dot) once they reached a SourceSink again. A SourceSink acts as a source and a sink at the same time. In this demo, in addition to Links and Bridges, there is an intersection component (orange-red dot) at the centre. This intersection component connects one road in the demo to the other. As in

Assignment 2, the trucks move (straight) at a certain speed and wait at bridges with a certain delay time. You can find all the component definitions in the "components.py" file.

Study the Model and the Data file

Your first task is to study the model and to see what changes are made in the components and the model generation. You need to gain a good understanding of how the demo model works to continue the assignment! Use the debugger to inspect the model. Examine the data input "demo-4.cvs". Check what differences it has compared to the data files used in Assignment 2. How does the model generator read the information in the data and generate the model accordingly? The vehicles in the demo-4 model would only drive straight. Why?

Ask the TAs when you have questions.

Description of implementation of routing mechanism in the model

The overall code structure remains the same as in the previous Assignment, except for the following changes:

- `generate_model(self):`

First, a list of roads is created (you can modify this to choose the subset of roads you want to explore), and all the columns associated with the set of roads are chosen from the input data frame and saved to another data frame called `df_objects_all`. The key change in this function is that of setting a path for vehicles. A path in this model is conceptualised as a set of ids that the vehicle will follow from its point of generation to the destination. These ids (in the correct order) are specified by you when you create the input generation file.

1. First, get the series of object IDs from your input csv in the original order.
2. A dictionary called `path_ids_dict` is created. For every road, its starting point and ending point are stored as keys, and the set of ids leading from the start to the end are stored as a list. For example, the entry for N1 will look as follows:

path_ids_dict = {

[1000000,1000012]: [1000000, 1000001, … ,1000011,1000012]

}

3. In order to facilitate vehicles to drive both ways, we also input the reverse order in the dictionary, as the next step.

path_ids_dict = {

[1000000,1000012]: [1000000,1000001, …, 1000011,1000012],

[1000012,1000000]: [1000012,1000011, …, 1000001,1000000]

}

4. We add another set of keys for the same source with a `[Source, None]` combination. This means if the destination is `None`, the default path is a straight route (right up to the end of the road). The `path_ids` for this key pair will be defined in the same manner as the `path_ids` above. This is repeated for the reverse direction as well.
5. Because we now have added the feature of "2-way traffic", the locations of `Source/Sink` will be able to generate trucks as well as remove trucks which have reached their destinations.

The new class called `SourceSink` is defined under components.py. In your csv input file, the starting and ending of the roads shall be specified with this new class.

- `set_path(self)`

This method definition is located under `Vehicle` class in `components.py`. After a vehicle agent is instantiated at a Source, the `set_path` method is called (within the Source class). The `set_path` method sets the origin and destination path of the vehicle, by further calling the `get_route` method which takes in a single argument: the id of the Source where the vehicle was generated. Now, given the point of generation of a vehicle, we can do several things to determine its route:

a) Get the straight route to the end of the road: `get_straight_route`

b) Get a random route by randomly choosing a sink: `get_random_route`

c) Any other predefined choice of destination by your definition.

In this model, both a) and b) have been implemented in a simple manner. Go through the two functions mentioned, located under the `BangladeshModel` class. These two routing functions can be called from the `get_route` method, which takes in a single argument: origin-id of the vehicle (id of the Source). By default, this function implements the straight-route method in this model. You are expected to modify this.

Both the `get_straight_route` and `get_random_route` methods look up the `path_ids_dict` dictionary (given the origin-destination pairs), which returns the `path_ids` that have been loaded into it.

Assignment and Deadline

In the previous Assignment, you generated a model of N1 from Chittagong to Dhaka. For this Assignment, you will include N1 and N2 as well as their side roads (N roads) that are longer than 25 km in the model generation to create an example network. Note that N1 and N2 are connected. To create such a road network, the intersections of the roads need to be placed at the right locations. (Hint: first, think about how to generate a data file following the example given in "demo-4.cvs". If there is no information about the exact location of an intersection, you can use an approximate location of a reasonable choice.)

You will generate vehicles from each end of the roads in the model. This simply means that create SourceSink instead of Source or Sink during the model generation. When a vehicle is generated at a "source", it shall drive to a random end location ("sink") in the network. To achieve this, you need to generate a NetworkX model of the road network. (Hint: import NetworkX into model.py and create a NetworkX model with corresponding nodes and links when the Mesa model is generated).

The NetworkX model shall be accessible by the Mesa model. Given an origin and a destination, the NetworkX model can return the shortest path between the two nodes so that a vehicle knows its path to arrive at the destination. To avoid looking up the shortest path in NetworkX each time a vehicle is generated, you can save the paths that were found into the path_ids_dict each time when a new path is discovered. So, the next time a vehicle is generated that wants to traverse the same path, it will look up the dictionary first to see if a path between its origin-destination pair exists. Only if the path does not yet exist, will it compute the shortest path using the NetworkX model.

Send trucks (entities) every 5 minutes from each end of the roads to measure delays and travel time in the network. Use the same speed setting and bridge breakdown settings as in Assignment 2. Run the model for 5 x 24 hours runtime. When you run the experiments, visualisation is not recommended. Run experiments for the following scenarios (with different physical bridges breaking down in each replication):

Scenario	Cat A %	Cat B %	Cat C %	Cat D %
0	0	0	0	0
1	0	0	0	5
2	0	0	5	10
3	0	5	10	20
4	5	10	20	40

Name the experimental output files as "scenario0.csv", "scenario1.csv", etc. Place these files in the "experiment" folder of the submission folder "EPA133a-Gxx-A3". (The xx shall be replaced with your group number.) Study the effects of each scenario on the travel time, analyse the travel time in the network in a meaningful way and discuss the results in a short report. Place the report in the "report" folder.

Bonus Exercises It is not necessary to do the bonus exercises. If you choose to, discuss the outcome in the report.

1. To obtain more accurate locations of the intersections, you can use the shape files (in the java model folder from Assignment 1) and find out where the roads intersect. (You can use, e.g., GeoPandas.)
2. Compare the new locations you found with the approximate locations you used. Reflect on the impact of the difference on the experimental outcome. (You do not need to run the scenarios. Just discuss this analytically.)

Hand in the "EPA133a-Gxx-A3" Zip file with the model, data, experiments, and a short report in pdf of how you prepared the data, designed your model, chose your way of analysis, and what the results of your experiments were. The recommended report length is 3-5 pages (this means 1500-2500 words), excluding visuals such as images and tables. Prepare the Zip file according to the Submission Guidelines. Upload the Zip file to the file area on Brightspace.

Time to spend and Support

There are 8 lab hours dedicated to completing the lab assignment. In addition, you are each expected to spend another **8 hours maximally** per person on carrying out the exercise. Don't overspend your hours and see how far you can get in 16 hours total. You could already get a passing mark if you prepare the data for the network model generation, generate the model with N1 and two of its main side roads that are longer than 25 km. (You are allowed to add the intersections in the data file manually if you wish.) Divide the work well within your group, and make sure you use the available hours of all team members combined well, through good collaboration and communications. We expect all team members to be able to contribute equally.

The deadline for handing in the Zip file of Assignment 3 is Friday in week 6 at 18:00.

Only upload using the Assignment function. Do not use the File Locker or email to hand in – we will base the grading on what you hand in as Assignment 3 on Brightspace.