GIS-E4030 — GIS Development L

May, 29th 2020

Public traffic simulation project
Helsinki case

Intro

The following documentation describes a public traffic simulation in Helsinki, Finland attempted by a group of students of Aalto University as a course assignment in order to see how possible congestion scenarios change the dynamics of public transit in the city. The project documentation consists of a source code and relevant instructions that guide a user through the process of gaining open data from The Helsinki Regional Transport Authority's (HSL) website, processing the data, simulating transit traffic based on the input data using combined microsimulation (MSM) and agent-based modeling (ABM) approaches, interpreting and visualizing results.

Authors

Arbenina Mariia, Dobosz Natalia, Järvenpää Antti, Kivekäs Jarmo, Pisl Jan.

Aalto University | Helsinki | Finland
Department of Built Environment

Getting started

Along with this documentation there is a .zip distribution copy of the source code repository that can be downloaded from this link (the latest version). Once the source code is extracted, the software and all its dependencies can be installed using *pip*:

```
python3 -m pip install path/to/local/repository
```

The simulation command line tool is accessible via:

```
python3 -m transit_simulation.cli --help
```

To make sure the installation is configured and working correctly, a user can run the pytest unit test suite. In the root of the repository, run:

```
python3 -m pytest tests/
```

For configuring a full development environment, refer to README.md in the source code distribution.

Running a simulation

The distribution .zip package includes example simulation data in the *tests/test_data/simulation_** directories. In its simplest form, supply the input data directory to the command line script to run a simulation:

```
python3 -m transit_simulation.cli ./tests/test_data/simulation_1
```

The default configuration is to run a 24 hour simulation starting at midnight. The simulation time window can be constrained using the --start_time and --end_time parameters. To run a 500 second simulation, starting 10 000 seconds past midnight:

```
python3 -m transit_simulation.cli ./tests/test_data/simulation_5
--start_time=10000 --end_time=10500
```

Input data structure

A typical simulation data directory will have the following files:

- routes.geojson
- schedule.csv
- speed_limits.geojson
- snapshots.geojson
- snapshots.gpkg
- simulation_extent.geojson

The main inputs are the *routes* and *schedule* files. The routes file contains the predetermined routes that agents will move along (bus routes, typically). This is a file with line geometry, and each line shall have a unique *shape_id* attribute, which is used to refer to it in the schedule file.

The schedule file is a table with columns for departure time, shape id, and agent type. It is used to control at what time simulated vehicles are added to the simulation, and what their behavior will be. The speed limit layer is one of the environment layers, it is included in the determination of agents velocity. The snapshots files are output data of the simulation. A snapshot file represents the location history of all agents over the course of a simulation. The simulation extent file is specific to HSL data: it is used in the selection of the bus transit routes that are used in the simulation.

The simulation: a technical description of the project

In general, the simulation tries to predict the effects of congestion on the bus network during one day. The produced result shows the movement of each bus from the starting location to the end stop, so it can be visualized the locations where the congestion affects the network the most.

The simulation has been made having ABM approach, but as the model only contains public transport agents, those lack interactions between each other (because there are no implemented rules for agent interaction and the agents most probably do not encounter each other in meaningful measures), the model has more MSM approach to the problem.

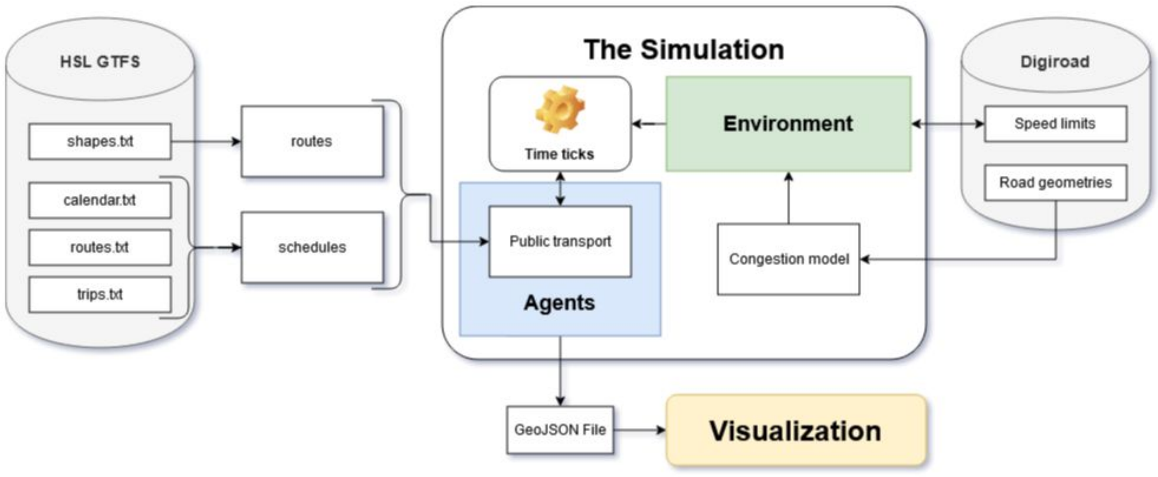


Fig. 01. General representation of the simulation structure

Data

Public transport agents

For public transport agents the data is extracted from gtfs-package hosted by HSL. One call of data fetching functions collects the needed information for one calendar day (current day by default) into two separate dataframes: routes and schedules. Routes are collected from shapes.txt file of the package.

For schedules three datatables are combined:

- trips.txt for daily timetable.
- calendar.txt for defining on which dates the route is operated.
- routes.txt for defining which vehicle type (tram, metro...) operates this route.

Processed route dataframe contains only route geometries. Schedule dataframe contains departure times and vehicle types. The data can be connected by common shape ids.

Environment initialization

The environment in this context is the external restrictions for an agent's movement on its path. In our implementation these are two variables: speed limit and congestion status. These are used to calculate the actual speed of an agent along its path. No actual data structure is maintained during the simulation to hold the environment, but speed limits are read from the digiroad speed limit file, and the congestion model is implemented in the simulation as discussed in the *Congestion model* chapter.

Simulation initialization

In the initialization phase of the simulation, agent objects are created from the route and schedule data described earlier. All individual objects have:

- Starting point (the beginning of their route).
- Departure time.
- Assigned route.
- Agent type (Only bus objects are implemented at the moment).

During the simulation agents are stored in one dataframe, where the *time tick function* is called for each object one by one.

Agents

In this simulation, an agent represents a bus. Each agent object is defined by following properties: route (a sequence of coordinates), location (a pair of coordinates) and speed.

The agent starts movement at the beginning of the route according to its schedule and travels along its assigned route until it reaches the last point of the route and only then it is removed from the simulation.

Time tick

One simulation step corresponds to constant change of time in reality. At each time tick the movement of agents is estimated as described in the subchapter below. In short, the simulation tries to estimate how the environment (speed limits, congestion) affects the movement on the agent's route.

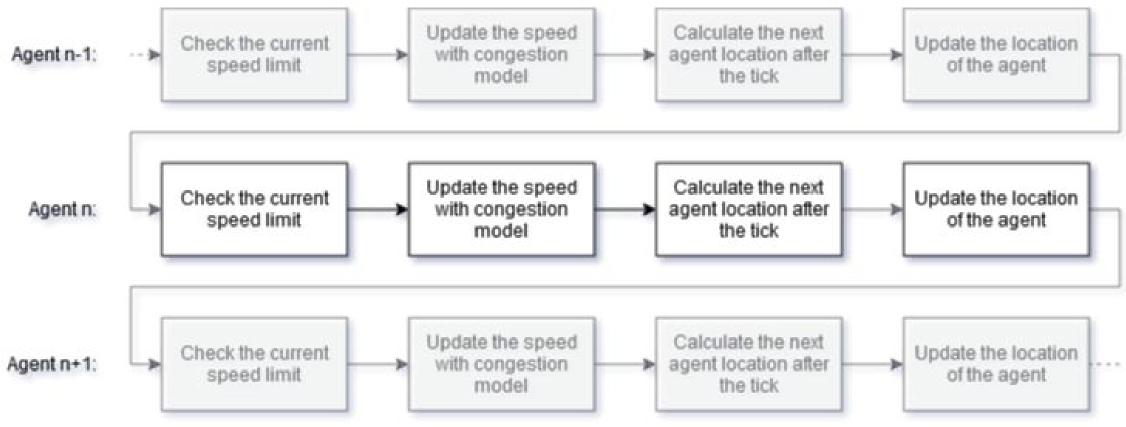

Agent n-1:
Check the current speed limit
Update the speed with congestion model
Calculate the next agent location after the tick
Update the location of the agent
Agent n:
Check the current speed limit
Update the speed with congestion model
Calculate the next agent location after the tick
Update the location of the agent
Agent n+1:
Check the current speed limit
Update the speed with congestion model
Calculate the next agent location after the tick
Update the location of the agent

Fig. 02. Simple representation how each agent's state is updated during a time tick

Interaction between agents and the environment

The interaction between agents and the environment is done via checking the speed limit for agents. In the application, each agent has a speed property. A speed value is calculated at every simulation tick from the speed limit at the current agent's location and corrected by congestion level dependent on the time of day and number of road elements surrounding the agent (more detailed information in the congestion model section). Speed limit value at a location is read from the road network vector dataset provided by the user to the application. In our example case, the speed limit layer from the Digiroad dataset is used. The dataset is preprocessed by clipping to a working extent of the simulation and saved as a GeoJSON file.

In the simulation, a buffer (with a default value of 100 m) around the input location is calculated. Next, all road LineStrings within the buffer are read to *geopandas* data frame. To calculate congestion at a given location, the number of road elements in the buffer is checked. Then, the distances between the input location and each road segment in the data frame are computed. The road segment with the shortest distance is selected. Lastly, the speed limit value of this road element is read and returned together with the number of road elements in the buffer.

Controlling the agent’s movement through time tick

The simulation is advanced with time ticks of a certain length. At each time tick, the agent checks the speed limit at its current position, corrects its value according to the congestion model and finally updates its speed property. Then, the distance that the agent can travel along its assigned route during one time tick with a certain speed is calculated. From there a point at the specified distance along the agent’s route is computed and used to update the agent’s current location property.

If an agent reaches the endpoint of the route assigned to it, the agent is removed from the simulation.

Congestion model

The congestion model estimates the speed of congested traffic flow from time of day and traffic network density. The idea of tracking these variables is that the congestion is caused by commuter traffic, and traffic must slow down in node locations where vehicles meet. The effects of different factors are multiplied together to form the overall estimate. The estimated congested speeds are basically used as an attribute of the environment.

Time of day

The traffic status at different times of day is modelled using the volume-to-capacity ratio that changes throughout the day. A problem of using time as input for modelling is the fact that time does not cause congestion but the interaction between vehicles. However, as the computation is done indirectly by estimating restricted speed from current volume to capacity status, the simulation could be changed relatively easily to take into account the modelled traffic load instead of time (although this approach would still miss the direct interaction between agents). The actual mathematical functions used are represented in the figure below. The development of VC-ratio is inspired by model results described in one OECD report (pp. 59, fig. 24).

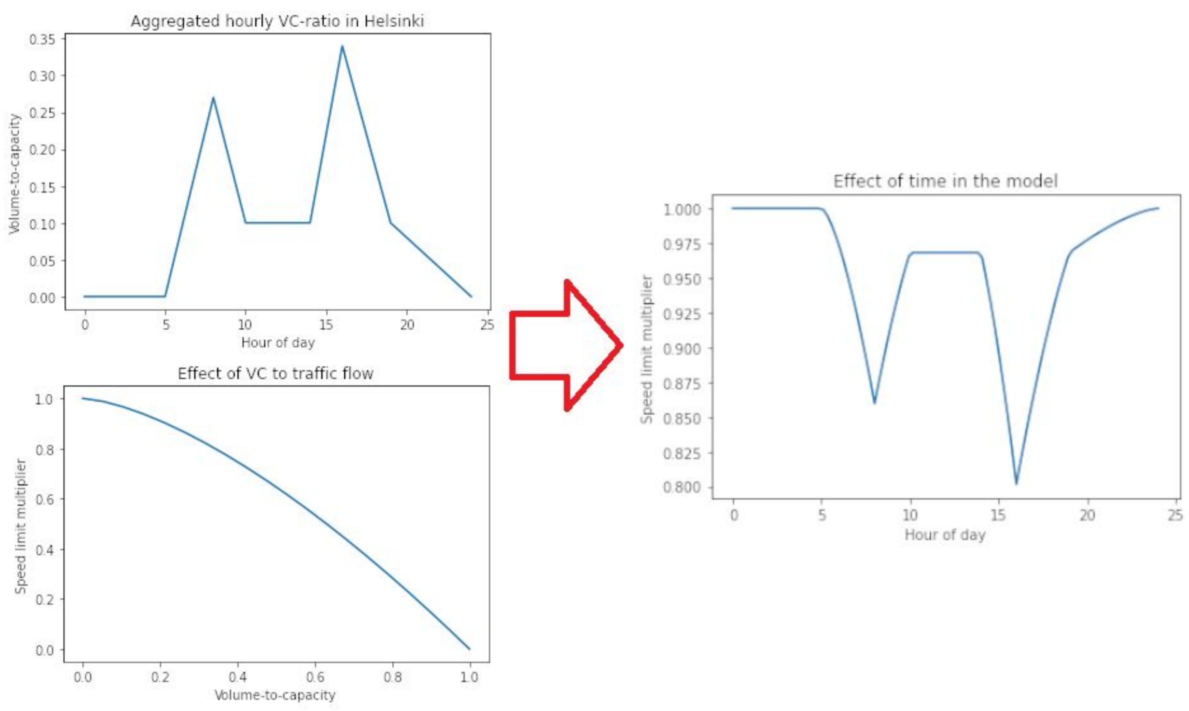


Fig. 03. A representation of the effect of volume-to-capacity ratio to traffic flow

Traffic network density

For estimating traffic network density, the number of unique elements in the digiroad dataset is used. One problem of this approach is that the intersections itself do not cause congestion, but the interaction between vehicles in them. The model does not make any difference if a vehicle is leaving or arriving at an area with multiple intersections (many road segments), but the speed is restricted in both cases. The effect of the number of elements is represented in the figure below.

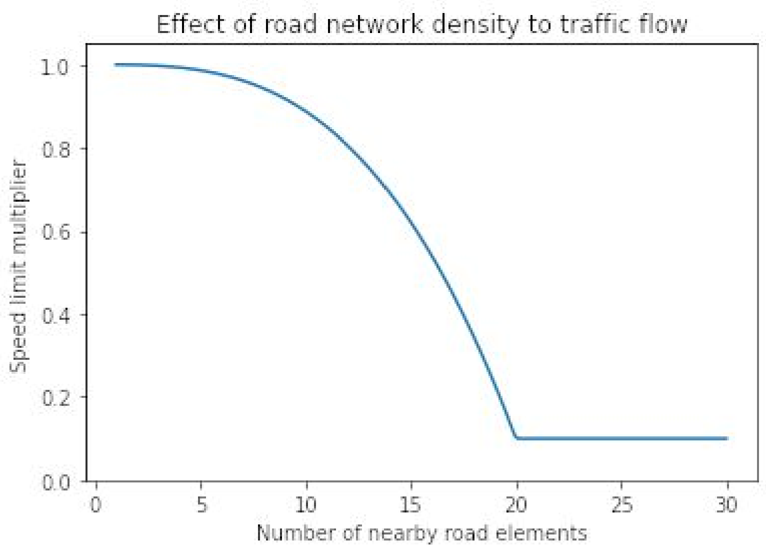
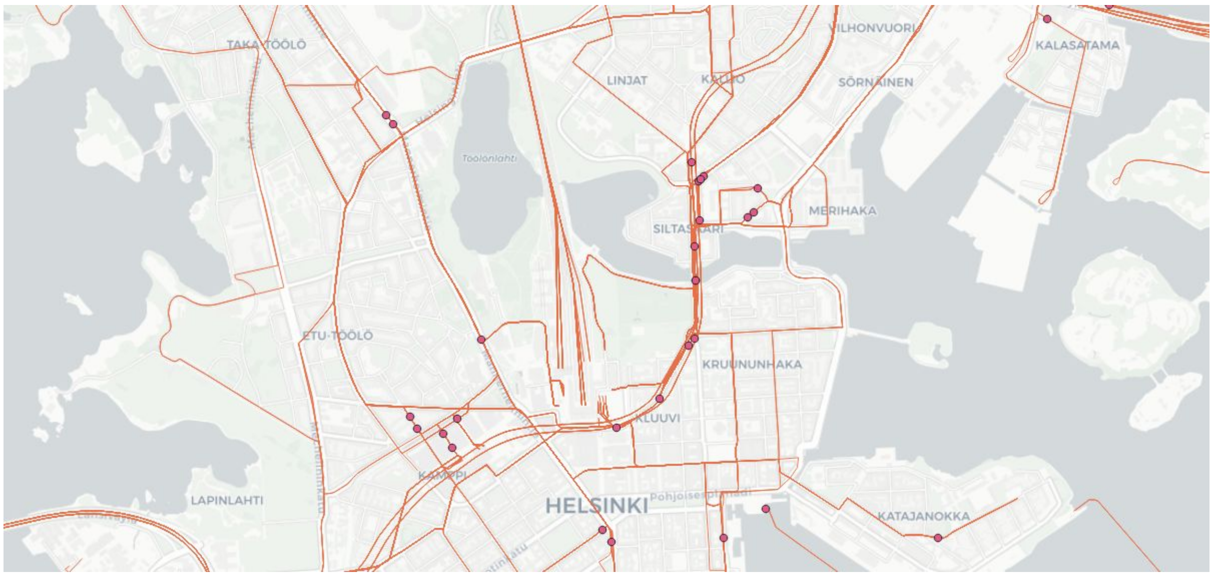


Fig. 04. A representation of the effect of the local traffic network density to traffic flow

Interpreting & Visualizing results

The end result of the simulation is a point layer which represents the location history for all agents over the duration of the simulation. The points include the timestamp representing a moment in the simulation.



Pic. 01 .Sliding window view of the resulting simulation data in QGIS Time Manager. See animated demo here

Visualization with KeplerGL

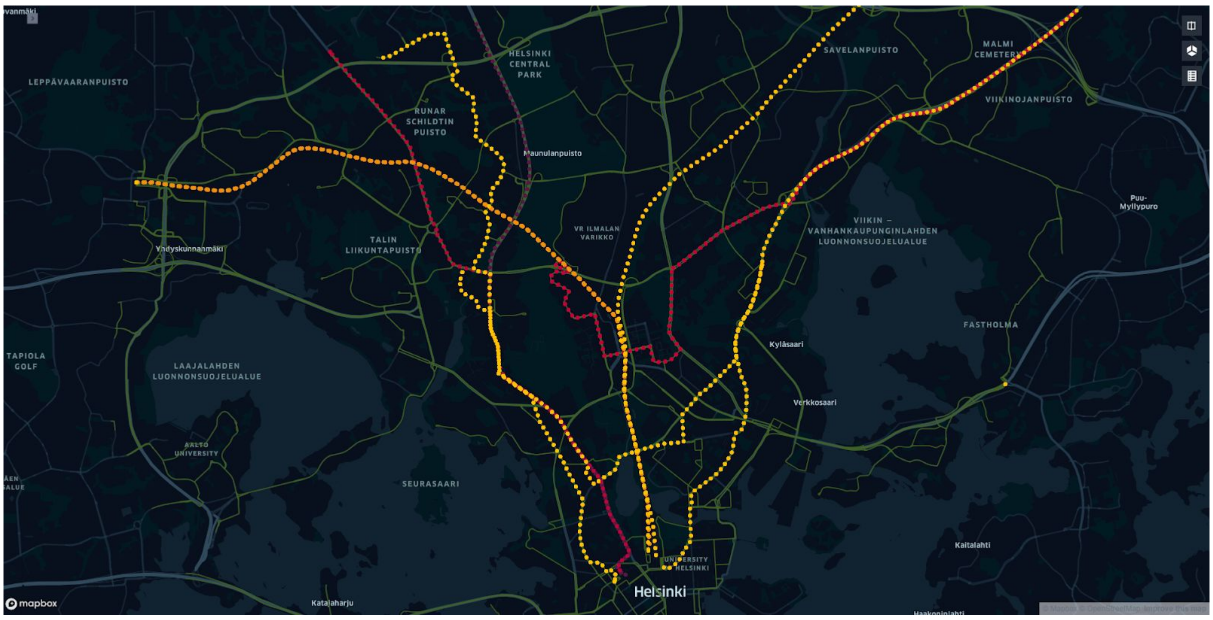
One way to visualize a simulation output is via KeplerGL open-source geoanalytics application for visualization and exploration of georeferenced datasets. It is meant to work with relatively big datasets as long as they conform to certain formats.

The tool supports CSV, GeoJSON, Pandas DataFrame or GeoPandas GeoDataFrame formats. The traffic simulation outputs a file in GeoJSON format with locations of traffic agents over the course of a simulation with respective timestamps. It then can be loaded to the *kepler* widget along with any other relevant layers, such as a road network dataset, bus stops, traffic lights, etc. All layers should have a common projection *WGS 84 / Pseudo-Mercator* to load properly.

The Jupyter Notebook incorporated in the source code (/VIZ_KeplerGL.ipynb) includes detailed instructions on how to visualize statically and possibly dynamically the output of the simulation. An attempted visualization of a simulation output should be considered exemplary for the project purposes.

The markdown cells of the notebook will guide a user through the sequence of steps to output an *interactive html* file with a visualization of the traffic simulation:

1. First KeplerGL should be installed using *pip* in the project environment: the notebook goes through the steps explaining how to install and enable the Jupyter extension for kepler.
2. Then the following code cells can be run to create a map object that will get loaded in the notebook using kepler widget.
3. More code is given to load GeoJSON files that need to be visualized: a simulation output, road network, and/or any additional data prepared by a user.
4. Instructions are given on how to interact with the widget in order to visualize all the layers, change color schemes and geometry representation, and then to store a map configuration for loading it later.
5. The resulting map then can be exported as an *interactive html* file that can be opened in a browser to observe a simulation output.
6. Additional instructions are provided on how to attempt a dynamic visualization with kepler. For that, a simulation output should be converted into a slightly different format (not realized in this project).
7. Throughout the notebook, a user guide of the kepler tool is linked with more detailed instructions on how to further customize a map and possible user interactions.



Pic. 02. Example of static visualization of agents' locations along several transit routes

Note that visualization results will vary depending on a simulation version, a chosen color scheme, and additional data loaded.

Expandability of the product: a discussion on further development

Multiple agent types

The simulation architecture allows using multiple types of agent in one simulation. The agent schedule input file includes a column where the agent type is specified. This means more complex simulations can be created by implementing agents with different logic.

More advanced movement logic

Now, in the simulation an agent's speed at a time is defined by the speed limit and the congestion status. In reality, a vehicle must accelerate and decelerate when e.g. leaving a bus stop or approaching speed bump. The estimation of a vehicle’s movement could be more realistic, if the speed was not only dependent on the limit, but would also take into account the acceleration of the vehicle by increasing or decreasing the speed of a previous time tick.

More environment elements

The simulation does not now take into account any other information about the road network but the speed limits. However, there is a lot more data available, e.g. in Digiroad and OpenStreetMap databases. A few examples of the possibilities could be:

- Buses not affected by congestion if a bus lane is available on the current road segment.
- Taking into account traffic lights.
- Taking into account bus stops and even passenger boardings.
- Taking into account speed bumps.
- Taking into account strict curves

Congestion state from the simulation itself

If other traffic than the public transportation was also modelled, the congestion status at a time tick could be calculated from the simulation itself. In the beginning of the project we actually had an idea to calculate the congestion status from the output data itself, but ran into problems with creating routes for arbitrary traffic. Without making really big changes, the volume-to-capacity ratio could be estimated from the density of agents on a road segment. This value could be used as input to the congestion model instead of time dependent status describing the state in the whole road network. Another possibility could be replacing the congestion model with collision avoidance rules between agents.

Case: Implementing other traffic

As mentioned above, at first we attempted to implement the simulation with a pure ABM approach. Apart from buses (and other means of public transport), cars were also modelled individually as agents. That way, congestion would be a natural output of the agents' behavior.

There were several decisions to make. The number of car-agents in the model had to be decided. Our approach was to generate a certain number of agents each time tick depending on the time of day. In peak hours, more cars would be in the model and therefore the probability of congestion would increase.

Also, the origin and destination of each agent had to be determined. For this, several approaches were tested. The simplest one was to generate agents' starting points randomly on the road network. Another option was to take into account the urban density. A more realistic option was to take into account also the type of buildings. That way, more realistic results could be achieved by e.g. assigning more starting points in residential areas in the morning where people typically commute to work. This, however, was out of scope of this project due to the complexity of the implementation.

The most challenging part turned out to be planning routes along which cars would move from their starting point to the destination. Given the high number of agents, it was not feasible to find a route for every agent "on the fly". As a workaround, we decided to create a database of routes from which every agent would simply fetch it's route based on the starting point and destination. For each starting point and destination, the fastest route would be computed. Since the number of possible starting points and destinations is infinite, we generated a number of points throughout the road network. The fastest routes were then generated between all pairs of these points. When a car agent was created, it fetched such a route which endpoints were closest to the agent's starting point and destination. The agent then moved first to the nearest point, from there followed the route to its end and finally reached its destination.

Unfortunately, generating fastest routes between a grid of points turned out to be too computationally heavy. It was done in QGIS, using a custom script based on a plugin QNEAT3. Even with a relatively low number of points (such as 30), the generation took about 48 hours. And, given the low number of points, the results weren't optimal. In the future, this could be done more efficiently if parallel computing was utilized or if there were a lot more of computational resources available (e.g. cloud computing).