Traffic Modeling with SUMO: a Tutorial

Davide Andrea Guastella*[1,2], Eladio Montero-Porras†[2], Alejandro Morales-Hernández‡[2], and Gianluca Bontempi§[2]

[1]Aix Marseille University, CNRS, LIS, Marseille, France
[2]Machine Learning Group, Université Libre de Bruxelles

Abstract

This paper presents a step-by-step guide to generating and simulating a traffic scenario using the open-source simulation tool SUMO. It introduces the common pipeline used to generate a synthetic traffic model for SUMO, how to import existing traffic data into a model to achieve accuracy in traffic simulation (that is, producing a traffic model which dynamics is similar to the real one). It also describes how SUMO outputs information from simulation that can be used for data analysis purposes.

1 Introduction

The concept of Smart Transportation has become more predominant over the past decade, encompassing innovative methods for the reduction of traffic congestion, traffic accidents, and air pollution, all of which engender excessive costs to society and impact the general well-being of citizens. This is necessary because of the increasing number of vehicles in urban environments. Although the awareness of city governments about sustainable mobility, such as investing in the design and development of mass transportation systems to reduce CO_2 emissions, still the high number of vehicles makes it necessary to analyze and implement policies for urban infrastructure management to optimally convey traffic and avoid congestion and the consequent air pollution. On the one hand, the impact of the control strategies on road infrastructures is not observable until they are deployed in the real world [5]. On the other hand, testing control strategies in real-life settings are expensive, risky, and often unfeasible [1]. In this context, urban traffic simulation models have become an indispensable asset, providing an *in-silico* environment where it is possible to design and assess alternative control strategies before the deployment. In this context, urban traffic simulation models have become an indispensable asset: these tools provide a lens through which it is possible to analyse control strategies *in silico*. They rely on computational models to test these strategies before deploying them in the real world.

Some of the advantages of traffic simulators are detailed as follows [3]:

- Using a simulation model, traffic management experts can assess decision-making tasks to reduce traffic congestion of certain sections of roads.
- A simulation model relies on an accurate topography of the city, consisting of buildings, roads, intersections, bridges. Using appropriate modeling software, it is possible to improve the geometric design of the road and see how these changes will affect the typical traffic flow.
- A simulation model allows evaluating the time and cost of the trip of vehicles. This is important when it is necessary to determine the economic assessment of a road infrastructure change. A specialist planning the transport work can conduct a comparative assessment of diverse options for traffic routes without significant material and time costs.

*davide.guastella@lis-lab.fr
†eladio.montero.porras@ulb.be
‡alejandro.morales.hernandez@ulb.be
§gianluca.bontempi@ulb.be

One of the challenges in traffic simulation lies in creating an accurate model of urban traffic: this requires several types of data, which are not always publicly available. The required data include socio-economic indicators as well as historical information about traffic flow. Having such information is crucial for defining accurate synthetic traffic models.

The goal of this document is to introduce traffic scenario modeling with the open-source simulation tool SUMO.

SUMO (Simulation of Urban MObility) is an open-source traffic simulation software designed for modeling and analyzing transportation systems. Developed by the German Aerospace Center (DLR), SUMO allows researchers, urban planners, and engineers to simulate realistic traffic scenarios, including private and public transport, pedestrian movement, and traffic light systems. It provides a microscopic simulation approach, meaning it models individual vehicle behavior based on car-following and lane-changing models. SUMO supports various input data formats, including real-world road networks imported from OpenStreetMap (OSM) and demand data from different sources, making it a versatile tool for traffic research and analysis.

One of SUMO's key advantages is its flexibility in integrating with external applications, enabling users to test traffic control algorithms, intelligent transportation systems (ITS), and autonomous vehicle coordination strategies. It provides detailed outputs on traffic dynamics, such as travel times, emissions, and congestion patterns for evaluating transportation policies and infrastructure planning.

The rest of the document is organized as follows: Section 2 introduces the common pipeline used to generate a synthetic traffic model for SUMO. Section 3 describes how SUMO outputs information from simulations, which can be used for traffic data analysis. Section 4 introduces the main tools that enable the automatic generation of synthetic traffic models for SUMO.

2 Creating a Synthetic Traffic Scenario

SUMO enables generating random road networks or converting OpenStreetMap (OSM) extracts to a specific format that can be used in SUMO, so to have a real-work representation of a road network. We will also focus on modeling synthetic traffic flow, which can be generated from either existing information about real traffic or generated randomly. Figure 1 shows the overall steps required to generate a synthetic traffic scenario using SUMO. Each step will be discussed in the following sections.

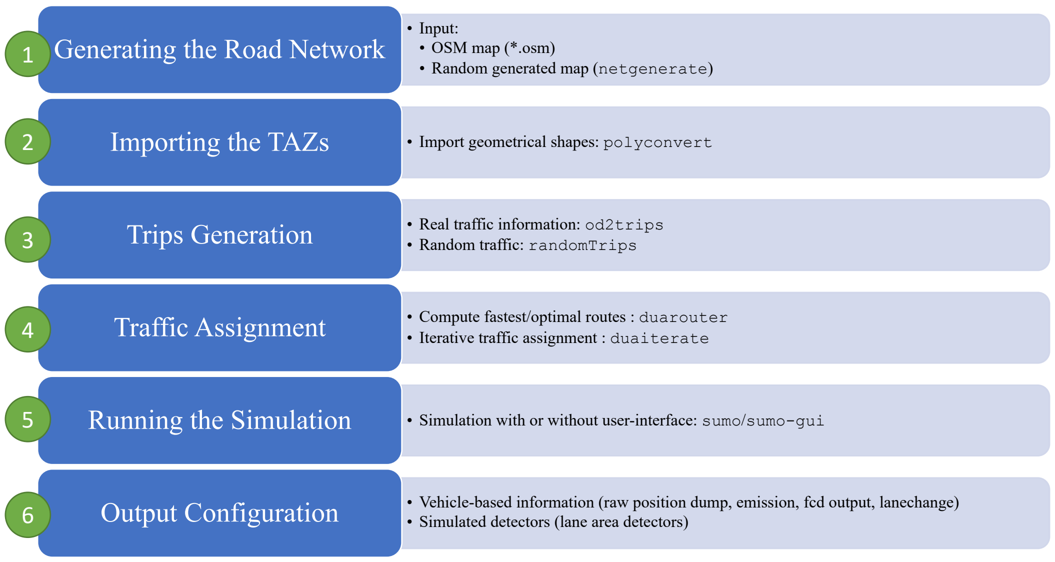

1 Generating the Road Network
• Input:
• OSM map (*.osm)
• Random generated map (netgenerate)
2 Importing the TAZs
• Import geometrical shapes: polyconvert
3 Trips Generation
• Real traffic information: od2trips
• Random traffic: randomTrips
4 Traffic Assignment
• Compute fastest/optimal routes : duarouter
• Iterative traffic assignment : duaiterate
5 Running the Simulation
• Simulation with or without user-interface: sumo/sumo-gui
6 Output Configuration
• Vehicle-based information (raw position dump, emission, fcd output, lanechange)
• Simulated detectors (lane area detectors)

Figure 1: Main steps required to generate and simulate a traffic scenario with SUMO.

Following, we briefly describe the steps required to generate and simulate a traffic scenario:

❶ **Importing the network**: this step produces a road network model that can be used in SUMO. The road network can be either generated randomly or converted from OpenStreetMap;

❷ **Importing the Traffic Analysis Zones (TAZs)**: import traffic analysis zones definitions. TAZs are polygons that delimits an urban environment according to some socio-economical indicators (such as average income, education level, traffic pressure, or simply administrative boundaries). TAZs are typically defined by governmental organizations. Dividing the environment into TAZs is useful to model the traffic flow from one local area of a city to another one;

❸ **Trips Generation**: assign an origin and a destination to each vehicle that should be inserted into the simulation. This can be done using either real or synthetic traffic data;

❹ **Traffic Assignment**: model the routes (herein a route is the complete path for going from origin to destination) for each vehicle in the simulation;

❺ **Simulation**: simulate traffic using the road network and the defined traffic model;

❻ **Output Configuration**: output information generated from the simulation that can be used for traffic analysis purposes;

2.1 Road Network Generation (step ❶)

This section introduces the tools available in SUMO to create a random road network and to import a road network from OpenStreetMap.

2.1.1 Random Road Networks

The command `netgenerate`[1] allows generating three types of abstract road networks: grid (using `--grid` parameter), spider (using `--spider` parameter) and random (using `--random parameter`). The use of randomly generated road networks is pertinent to validate data analysis techniques in traffic domain; the same technique that rely on traffic data can be evaluated on several simulated scenarios, each one having a specific topology, type of junctions, or traffic light options.

The following command creates a random grid-like road network topology:

```
> netgenerate --grid --grid.number=10 --grid.length=400
       --output-file=MySUMOFile.net.xml
```

which produces the output shown in Figure 2.

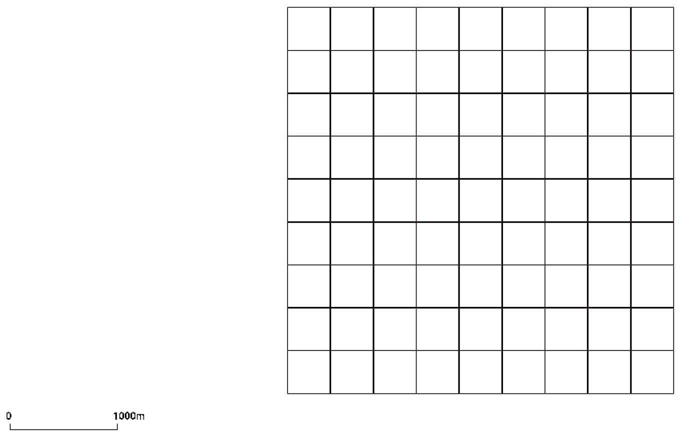


Figure 2: Random grid road network topology generated by `netgenerate`.

The following command creates a random spider-like road network topology:

[1]https://sumo.dlr.de/docs/netgenerate.html

```
> netgenerate --spider --spider-omit-center --output-file=MySUMOFile.net.xml
```

which produces the output shown in Figure 3.

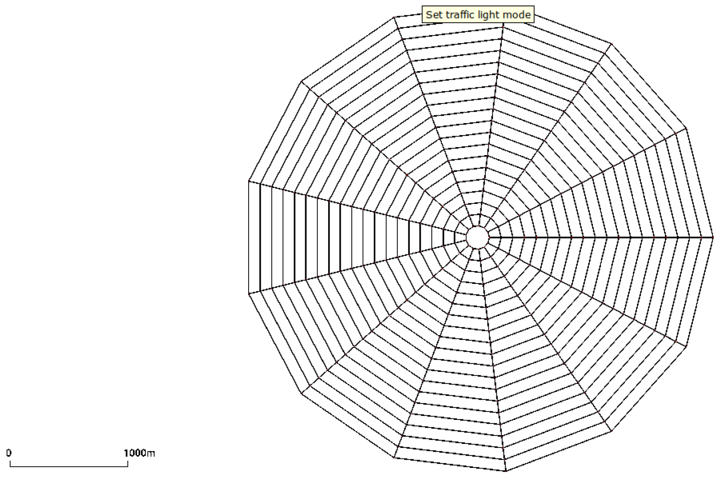


Figure 3: Random spider-like road network topology generated by `netgenerate`.

The following command creates a random road network topology:

```
> netgenerate --rand -o MySUMOFile.net.xml --rand.iterations=200
```

which produces the output shown in Figure 4.

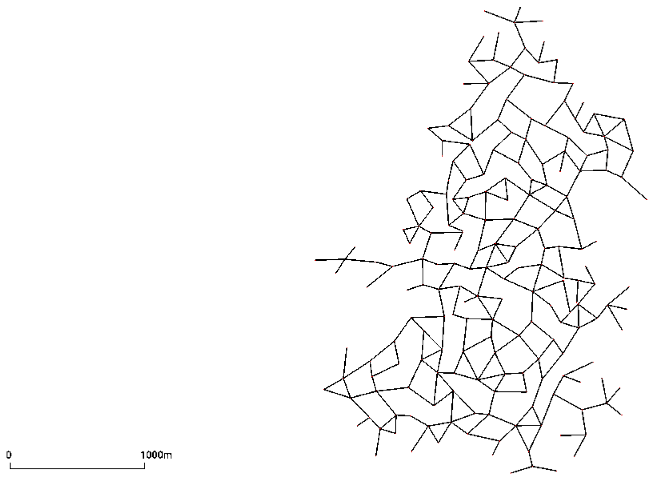


Figure 4: Random road network topology generated by `netgenerate`.

Additionally, by setting the option `--rand.grid`, additional grid structure is enforce during random network generation, which produces the output shown in Figure 5.

2.1.2 Extracting a Road Network Topology from OpenStreetMap

In SUMO, a model of a real road network can be defined manually. To do this, an XML file must be defined, containing elements such as the roads definition, intersections, and traffic lights. However, manually defining the road network topology for a real urban area is impractical due to the significant amount of time required. To tackle this issue, SUMO provides a tool to convert road network from OpenStreetMap (OSM) files.

The first step to import a road network from OSM in SUMO is to select the region to be modeled, as shown in Figure 6.

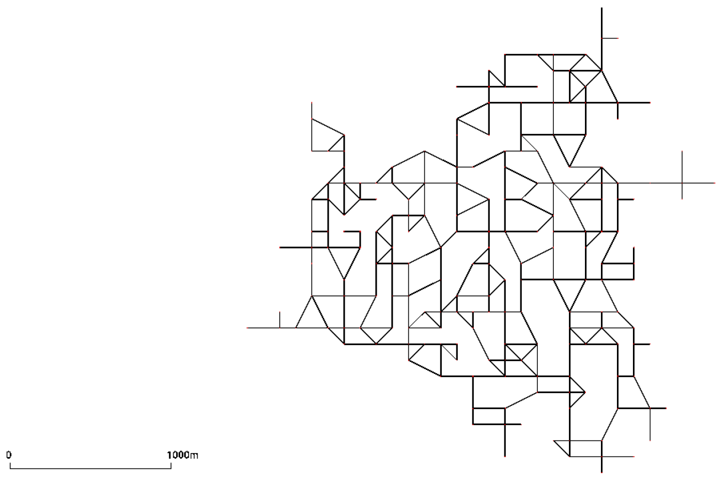
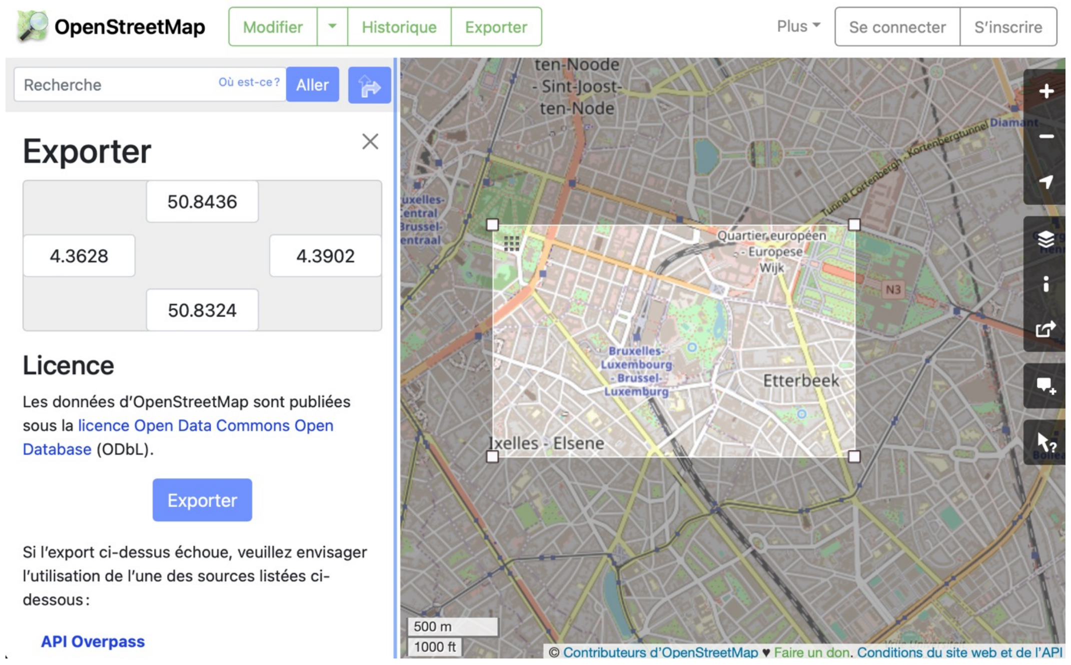


Figure 5



Exporter

	50.8436	
4.3628		4.3902
	50.8324	

Licence

Les données d'OpenStreetMap sont publiées sous la licence Open Data Commons Open Database (ODbL).

Exporter

Si l'export ci-dessus échoue, veuillez envisager l'utilisation de l'une des sources listées ci-dessous :

API Overpass

© Contributeurs d'OpenStreetMap ♥ Faire un don. Conditions du site web et de l'API

Figure 6: OSM allows exporting a selected area into an XML format.

The next step is to export and download the map. If the "Export" button is selected and the selected area is too big, then OSM will not export the selected area because the number of nodes within the area is above the limit (50000 is the maximum allowed nodes in a selected region for exporting). By selecting the "overpass API" link it is possible to overcome this problem. The output is an XML file containing the definition of all the features in the selected region. In OSM, a feature is any physical element (natural or human-made) in the landscape. Some examples of features are buildings, roads, vegetation, land use, railways, and waterways.

In some cases, it is required to model a specific part of the real environment. For this, exporting the OSM file from the website as discussed previously is not a pertinent solution, as this can include parts of the real environment that must not be modeled. Suppose we must model the road network for a part of the Ixelles municipality (Belgium, Figure 7):

By using the OSM tool, it is not possible to obtain a network containing only the roads included in the

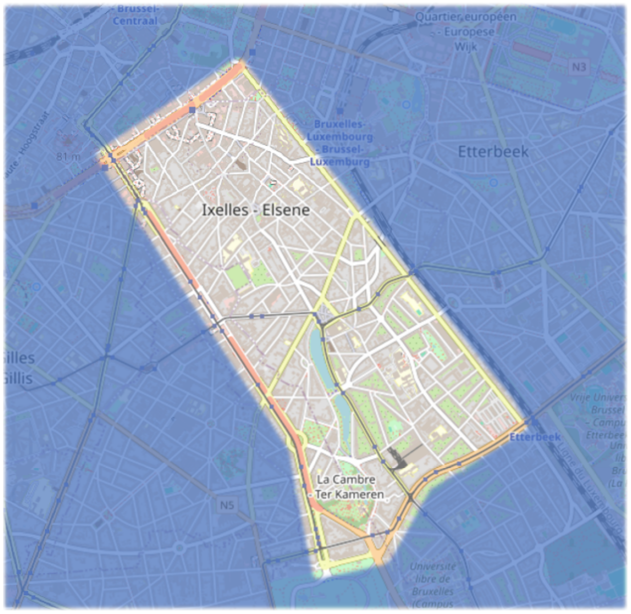


Figure 7: Part of the Ixelles municipality, Belgium

part of environment in Figure 7. To extract the road network for a specific part of the environment, first define the polygon in geoJSON delimiting the road network to model. Figure 8 shows a polygon enclosing a part of the Ixelles municipality. Any online geoJSON modeling tool can be used for this purpose. Herein we use `geojson.io`.

After generating the geoJSON containing the polygon that delimitates the road network to model, the following steps must be executed to obtain a road network for SUMO:

- Download the `*.pbf` file of the country that includes the road network part to model (for Belgium the file is available at `https://download.geofabrik.de/europe/belgium.html`)
- Install `osmium` and `osmfilter`. In Mac Os X, can be installed through `brew`[2].
- Extract the `*.pbf` containing only the elements included in the defined polygon:

```
osmium extract -p my_poly.geojson my_country.osm.pbf
    -o filtered.pbf --set-bounds --overwrite
```

  where `my_poly.geojson` is the geoJSON file, `my_country.osm.pbf` is the file of the modelled country, `filtered.pbf` is the filtered country `*.pbf` file referring to only the part included in the defined polygon.

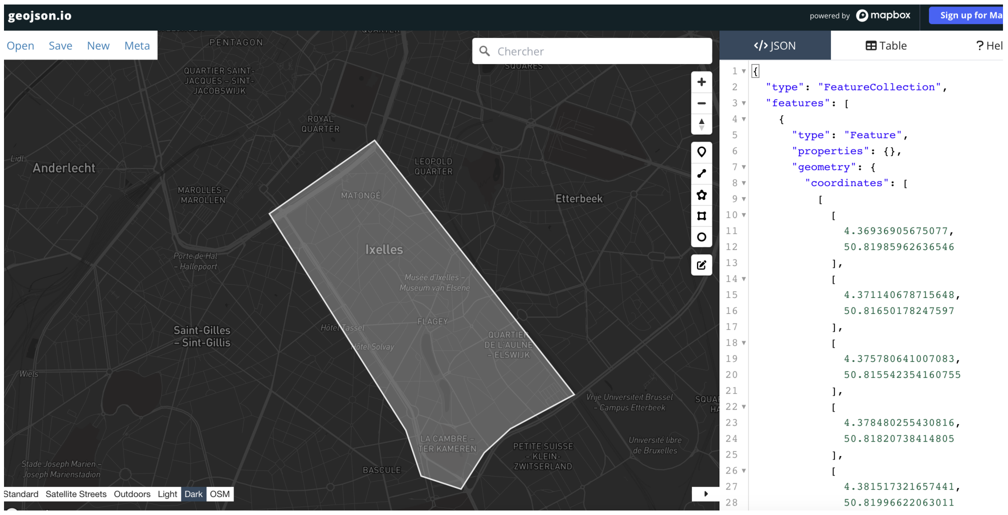
- Convert `*.pbf` to `*.osm`:

```
osmium cat filtered.pbf -o road_net.osm
```

- Filter only the road network. All unnecessary elements such as buildings or walkways are discarded:

```
osmfilter road_net.osm --keep="highway=* building=*" -o=road_net_filtered.osm
```

[2]https://brew.sh



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "coordinates": [
          [
            [
              4.36936905675077,
              50.81985962636546
            ],
            [
              4.371140678715648,
              50.81650178247597
            ],
            [
              4.375780641007083,
              50.815542354160755
            ],
            [
              4.378480255430816,
              50.81820738414805
            ],
            [
              4.381517321657441,
              50.81996622063011
```

Figure 8: Modelling a specific part of the real environment using a geoJSON online tool.

- Finally, the output file `road_net_filtered.osm` can be converted to SUMO format using the `netconvert` tool:

```
netconvert --osm road_net_filtered.osm  -o net.xml --ramps.guess
    --junctions.join --remove-edges.isolated --output.street-names
    --output.original-names
```

All the features in the OSM XML file are represented as "`node`" tags. Each node has an identifier, a position (in latitude/longitude), and other useful information that allows identifying the type of node, such as the type of road, building (apartments, offices, theaters, etc.), or bike lane.

Before converting the OSM file into a format compatible with SUMO, it may be useful for decision-making purposes to modify certain properties of the road network, such as the direction of the roads. Appendix B includes the Python code to reverse the direction of one or more roads in an OSM-format road network.

To use an OSM map in SUMO, it is necessary to convert the XML file containing the definition of the extracted urban area to a specific format for use with the simulator. SUMO comes with a tool named `netconvert`[3], that allows importing road networks from different sources such as:

- "SUMO plain" XML descriptions (*.edg.xml, *.nod.xml, *.con.xml, *.tll.xml)
- OpenStreetMap (*.osm.xml/*.osm), including shapes (see OpenStreetMap import)
- VISUM, including shapes and demands
- Vissim, including demands
- OpenDRIVE
- MATsim
- SUMO (*.net.xml)
- Shapefiles (.shp, .shx, .dbf), e.g. ArcView and newer Tiger networks
- Robocup Rescue League, including shapes
- a DLR internal variant of Navteq's GDF (Elmar format)

[3]https://sumo.dlr.de/docs/netconvert.html

In its most simple usage, netconvert takes in input only the XML file obtained from OSM (parameter `--osm`), and output a road network in XML file (parameter `-o`):

```
> netconvert --osm my_osm_net.xml -o my_sumo_net.net.xml
```

We suggest using `netconvert` with the following options:

- `--ramps.guess`: Enable ramp-guessing.
- `--junctions.join`: Joins junctions that are close to each other.
- `--tls.guess-signals`: Interprets tls nodes surrounding an intersection as signal positions for a larger TLS.
- `--tls.discard-simple`: Does not instantiate traffic lights at geometry-like nodes loaded from other formats than plain-XML.
- `--tls.join`: Try to cluster tls-controlled nodes.
- `--tls.default-type actuated`: Use traffic light programs that adapt to demand dynamically.
- `-t $SUMO_HOME/data/typemap/osmNetconvert.typ.xml`: use standard types for converting to SUMO format.
- `--remove-edges.by-vclass rail_slow,rail_fast,bicycle,pedestrian`: Remove edges where trains, bikes and pedestrian are allowed. This keeps only the edges for vehicular traffic.
- `--remove-edges.isolated`: remove isolated edges in the road network.
- `--output.street-names`: Street names will be included in the output.
- `--output.original-names`: Keep the original names of the streets in OSM.

2.2 Traffic Assignment Zones (TAZs, step ❷)

Traffic Analysis Zones (TAZs) are spatial units used in transportation modeling and traffic studies to represent areas with similar travel behavior. A TAZ typically models a neighborhood or district, and serves as a fundamental unit for estimating travel demand. Each zone can be associated with socio-economic and demographic data, such as population, employment, and land use. This information influences the number and type of trips generated within the area. TAZs are commonly used in trip-based models, such as the traditional four-step travel demand model (trip generation, trip distribution, mode choice, and route assignment).

Figure 9 shows the inner part of the route network of the city of Brussels, extracted from OSM and imported in SUMO using the `netconvert` tool. The area covers approximately an area of 24.000m^2. For sake of simplicity, we model only the road network, therefore excluding railways and other buildings.

We now evaluate the TAZs for the modeled area using the spatial boundaries of Brussels Capital Region's neighborhoods. This data is freely available from the computer center for the Brussels Region (Centre d'Informatique pour la Région Bruxelloise, CIRB)[4]. The data is provided as shapefile, a geo-spatial vector data format commonly used for Geographic Information System (GIS) software.

To use the shapefile in SUMO as TAZs, this must be converted in a proper format using the `polyconvert`[5] tool. This tool can be used to generate additional files for SUMO containing information about all the polygons (e.g., buildings, grounds, etc.).

The following command extracts the polygons from a shapefile and convert them in a format compatible with SUMO. The value "UrbAdm_Monitoring_District" is the prefix of the shapefile from CIRB, containing the spatial boundaries of Brussels neighborhoods.

[4]https://data.metabolismofcities.org/library/33895/
[5]https://sumo.dlr.de/docs/polyconvert.html

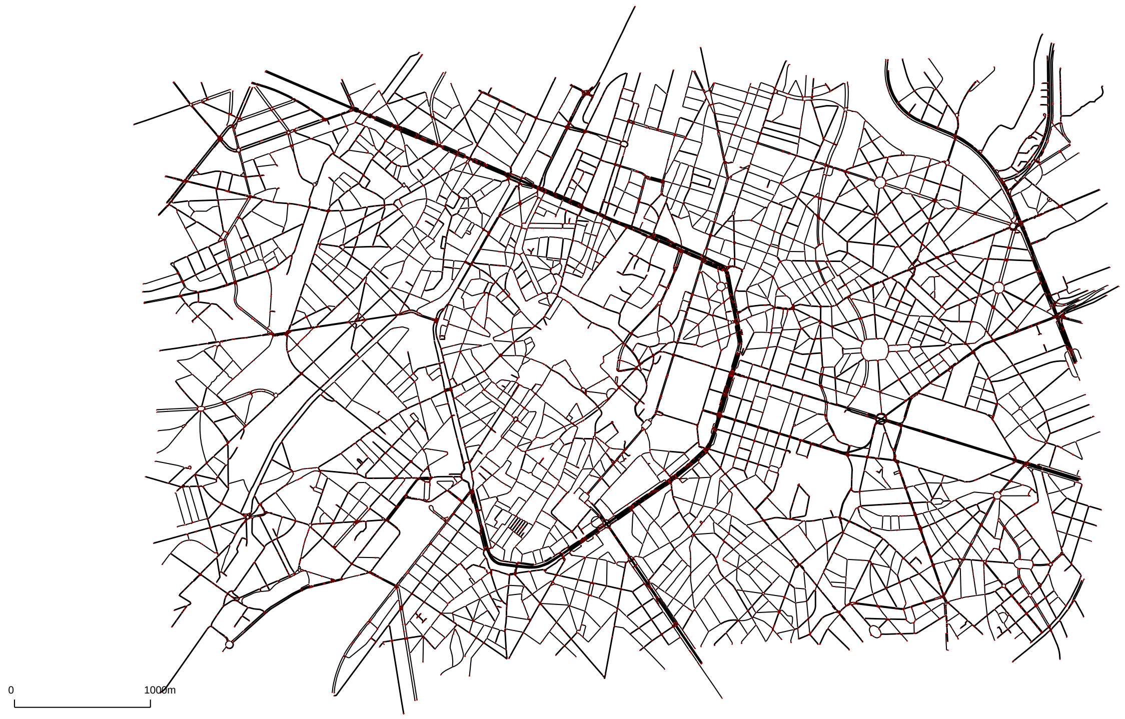


Figure 9: Road network of the city of Brussels.

```
> polyconvert --shapefile-prefix UrbAdm_Monitoring_District
       --shapefile.guess-projection true --shapefile.traditional-axis-mapping true
       --shapefile.id-column ID -n ../bxl.net.xml -o blx.poly.xml
```

where `UrbAdm_Monitoring_District` is the prefix of the shapefile. The parameter `--shapefile.guess-projection` takes a boolean value: if true, the program guesses the shapefile's projection. `--shapefile.id-column` is the name of the column containing the ID of each shape in the shapefile. The parameters `-n` and `-o` are respectively the road network definition and the output XML file that can be used with SUMO.

Figure 10 shows the road network of Brussels city and the polygons delimiting the neighborhoods, obtained from the shapefile of CIRB. Each neighborhood is colored randomly. We used the `netedit` tool to visualize the road network and the neighborhood.

SUMO provides the tool `edgesInDistricts`[6] to convert polygons delimiting the neighborhoods to TAZs. This tool reads the polygons from an input polygon files (created using `polyconvert`) and output an XML containing the TAZs definition. Each TAZ includes all the edges inside the polygon. The tool `edgesInDistricts` can be used as follow:

```
> python $SUMO_HOME/tools/edgesInDistricts.py -n bxl.net.xml
       -t taz.poly.xml -o TAZ.xml
```

The output file containing the TAZs has the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<tazs xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/taz_file.xsd">
    <taz id="ANDERLECHT CENTRE - WAYEZ" color="51,128,255" edges="-1019451816#0
             -1019451816#1 -106463402#0 -106463402#1 ..."/>
    <taz id="ANNEESSENS" color="51,128,255" edges="-1007289475 ..."/>
    ...
</tazs>
```

[6]https://sumo.dlr.de/docs/Tools/District.html

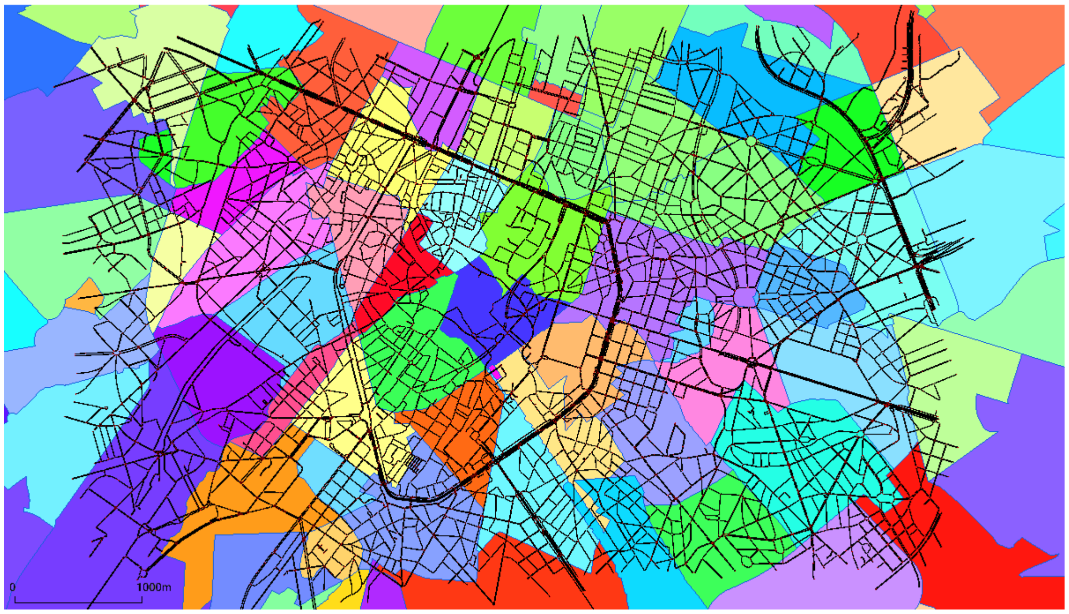


Figure 10: Spatial boundaries of Brussels capital region's neighborhoods in SUMO. Colors are assigned randomly to each neighborhood.

In case the polygons that separate the modeled urban area are not available, the TAZs can be generated randomly. SUMO provides two commands for this:

- `generateBidiDistricts.py`: create TAZs and assign to each one edges that are opposite of each other;
- `gridDistricts.py`: create a grid of TAZs for an input road network, each one with a specified size (in meters).

Figure 11 shows the TAZs generated randomly using the `gridDistricts.py` tool.

2.3 Trips Generation (step ❸)

Trips are the building blocks for modeling vehicular traffic. A trip is identified by a departure time, an origin and a destination point. A trip is associated with one unique vehicle.

2.3.1 Using OD-Based Traffic Demand

Trips in SUMO can be modeled starting from Origin/Destination (OD) matrices. The content of an OD matrix is the number of vehicles that flow from an origin to a destination during a specific time horizon.

The command `od2trips`[7] imports OD matrices and splits them into individual vehicle trips. In the output file, each trip is defined by an id with starting and ending time (included inside the given time-lapse), and the origin and destination edges in the road network. If different transportation modes are considered, then `od2trips` must be used for each transportation mode and for each modeled time horizon.

The following command produces a trip file for SUMO starting from an OD demand definition in file "`OD_matrix.od`":

```
> od2trips -v --taz-files TAZs.taz.xml --vtype passenger --prefix car
       --od-matrix-files OD_matrix.od -o output/output.odtrips.xml
```

[7]https://sumo.dlr.de/docs/od2trips.html

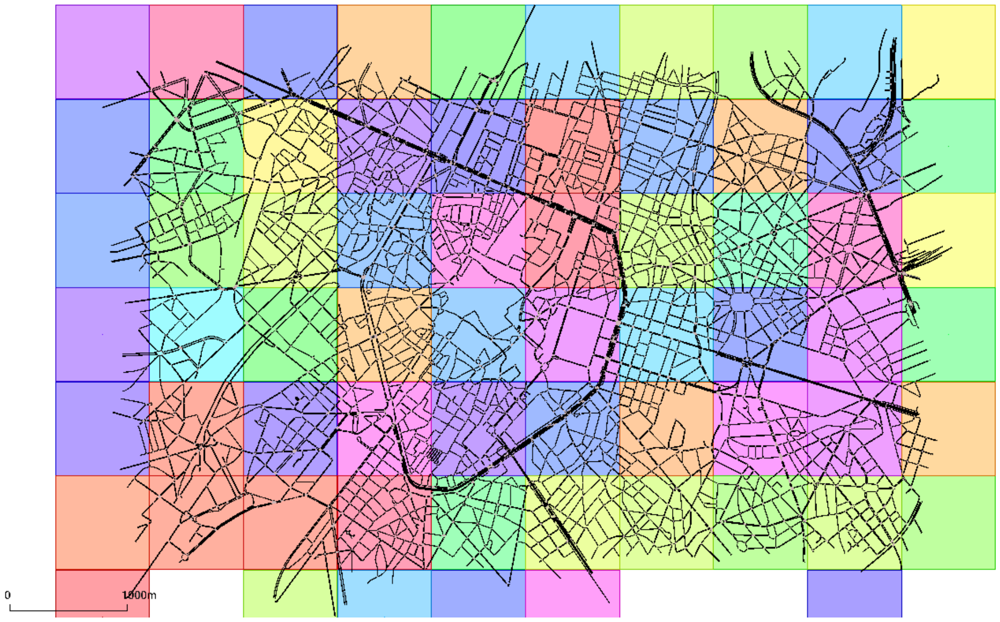


Figure 11: TAZs generated using the `gridDistricts.py` tool

An OD matrix is typically in O-format, that lists each origin and each destination together with the amount of vehicles flowing from origin to destination. Following, we show an example of OD matrix in O-format.

```
$OR;D2
* From-Time To-Time
7.00 8.00
* Factor
1.00
        1         1       1.00
        1         2       2.00
        1         3       3.00
        2         1       4.00
        2         2       5.00
        2         3       6.00
        3         1       7.00
        3         2       8.00
        3         3       9.00
```

where:

- The first line is a format specifier that must be included verbatim.
- The lines starting with '*' are comments and can be omitted
- The second non-comment line determines the time range given as `HOUR.MINUTE HOUR.MINUTE`
- The third line is a global scaling factor for the number of vehicles for each cell
- All other lines describe matrix cells in the form `FROM TO NUMVEHICLES`

A further TAZ format supported by SUMO is the **tazRelation**. The following example shows how to use OD traffic demand in the tazRelation format and generate a traffic demand compatible with SUMO. Figure 12 shows a sample road network. In this network, we consider eight edges as origins and destinations.

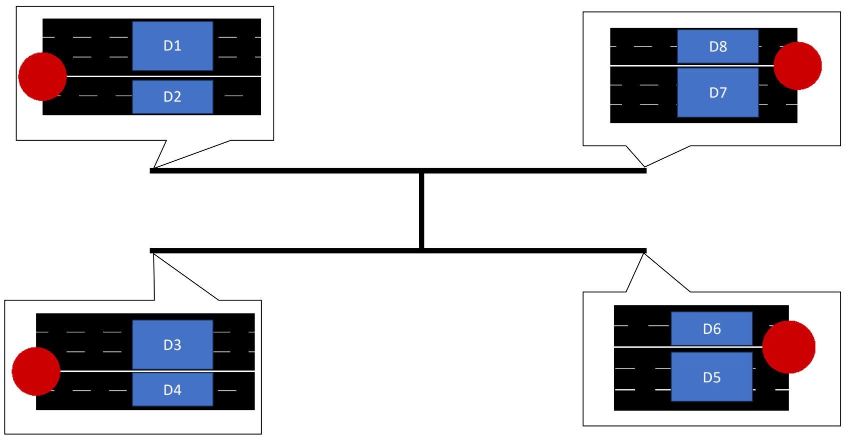


Figure 12: Road network example. Edges from D1 to D8 are used to model origin and destination of vehicles.

Following, we report an example of OD demand definition in tazRelation format for the road network in Figure 12.

```
<data>
    <interval id="0" begin="0" end="3600">
        <tazRelation from="D2" to="D7" count="100"/>
        <tazRelation from="D2" to="D5" count="30"/>
        <tazRelation from="D2" to="D3" count="90"/>
        <tazRelation from="D8" to="D1" count="90"/>
        <tazRelation from="D8" to="D3" count="200"/>
        <tazRelation from="D8" to="D5" count="50"/>
        <tazRelation from="D4" to="D1" count="10"/>
        <tazRelation from="D4" to="D7" count="170"/>
        <tazRelation from="D4" to="D5" count="90"/>
        <tazRelation from="D6" to="D3" count="110"/>
        <tazRelation from="D6" to="D7" count="30"/>
        <tazRelation from="D6" to="D1" count="80"/>
    </interval>
</data>
```

The `from` and `to` fields indicate respectively the origin and destination edges. The field `count` is the number of vehicles that must flow from the origin to the destination edge during the interval “0”. We now use the tool `od2trips` to imports OD demand in tazRelation and split them into individual vehicle trips. This tool generates an XML file that has the following format (using the previous TAZ definition):

```
<routes>
   <trip id="63" depart="0.79" from="D2" to="D3" fromTaz="D2" toTaz="D3"/>
   <trip id="282" depart="2.83" from="D4" to="D5" fromTaz="D4" toTaz="D5"/>
   <trip id="516" depart="4.77" from="D6" to="D1" fromTaz="D6" toTaz="D1"/>
   <trip id="327" depart="8.55" from="D4" to="D7" fromTaz="D4" toTaz="D7"/>
   <trip id="244" depart="11.12" from="D4" to="D5" fromTaz="D4" toTaz="D5"/>
   <trip id="88" depart="16.72" from="D2" to="D3" fromTaz="D2" toTaz="D3"/>
   <trip id="9" depart="18.39" from="D2" to="D3" fromTaz="D2" toTaz="D3"/>
   ...
<routes/>
```

Differently from the tazRelation format, the file generated by `od2trips` contains information about each vehicle that should be inserted into the simulation. Next, we use the trip file in the subsequent traffic assignment (Section 2.4), followed by the simulation step. The traffic assignment involves finding a path from the origin to the destination edge.

SUMO also provides a tool that allows generating OD matrices from TAZs and route definition files, named `route2OD.py`[8]. Following we show an example of, requiring a route file (-r), and a TAZs file (-a).

```
> python $SUMO_HOME/tools/route/route2OD.py -r duarouter.rou.xml
       -a TAZ.xml -o OD_matrix.xml
```

We provide a script to generate random OD traffic demand[9] (called ODDemandGenerator.py), which can be used as follow:

```
> python ODDemandGenerator.py -n my_net.net.xml -t taz.out.xml -p 10
 -x odpairs.xml -l 0 -j 10
```

The output of this script are two: the TAZ definition file and the OD demand definition.

2.3.2 Generating Random Trips

SUMO provides the tool `randomTrips.py`[10] to generate random trips for a given road network. It does so by choosing source and destination edge either uniformly or at random. The resulting trips are stored in an XML file that can be later used with `duarouter`[11], a tool available in SUMO for generating routes. The trips are distributed evenly in a temporal interval defined by begin (option -b, default 0) and end time (option -e, default 3600) in seconds. The number of trips is defined by the repetition rate (option -p, default 1) in seconds.

The following command generates random trips starting in one hour interval for a given road network (parameter `-n`):

```
> python tools/randomTrips.py -n <net-file> -e 3600
```

The `randomTrips` command allows specifying the density of traffic generated per unit of time. By default, one vehicle is added into the simulation at each second. The period at which vehicles are added into the simulation can be specified by using the `--period <FLOAT>` option: in this way, the arrival rate is of (1/period) per second. By using values below 1, multiple vehicles are added into the simulation at each second.

If several `<FLOAT>` numbers are provided, like in `--period 1.0 0.5`, the time interval will be divided equally into sub-intervals, and the arrival rate for each sub-interval is controlled by the corresponding period (in the preceding example, a period of 1.0 will be used for the first sub-interval and a period of 0.5 will be used for the second). There are two other ways to specify the insertion rate:

- using the `--insertion-rate` parameter: this is the number of vehicles per hour that the user expects.
- using the `--insertion-density` parameter: this is the number of vehicles per hour per kilometer of road that the user expects (the total length of the road is computed with respect to a certain vehicle class that can be changed with the option –edge-permission).

When adding option `--binomial <INT>` the arrivals will be randomized using a binomial distribution where n (the maximum number of simultaneous arrivals) is given by the argument to `--binomial` and the expected arrival rate is 1/period.

Let us suppose we want to let n vehicles depart between times `t0` and `t1` set the options, the following parameters must be provided:

```
> python tools/randomTrips.py -n <net-file> -b t0 -e t1 -p ((t1 - t0) / n)
```

[8]https://sumo.dlr.de/docs/Tools/Routes.html#route2odpy
[9]https://github.com/davide990/ODgenerator
[10]https://sumo.dlr.de/docs/Tools/Trip.html#randomtripspy
[11]https://sumo.dlr.de/docs/duarouter.html

By default the departures of all vehicles are equally spaced in time. Since the inserted vehicle are spread randomly over the whole network, this comes out as a binomial distribution of inserted vehicles for each individual edge which gives a good approximation to the Poisson distribution if the network is large (and hence the insertion probability of each edge is small). By setting set option `--random-depart`, the (still fixed) number of departure times are drawn from a uniform distribution over [begin, end]. This leads to an exponential distribution of insertion time headways between vehicles on all edges (which is the headway pattern of the Poisson distribution). Hence, this is useful to have a more varied insertion time pattern for small networks.

One interesting feature that is available in `randomTrips` is that it is possible to assign a unique weight to each edge into the input network by using the `--weights-prefix <STRING>` paraemter with the prefix value as argument.

The tool will load weights for all edges by finding a file (within the running directory) with extension .src.xml, .dst.xml or .via.xml. According to the file extension, weights are used differently for routes generation:

- .src.xml contains the probabilities for an edge to be selected as from-edge
- .dst.xml contains the probabilities for an edge to be selected as to-edge
- .via.xml contains the probabilities for an edge to be selected as via-edge (only used when option `--intermediate` is set).

2.4 Traffic Assignment (step ❹)

So far, we have described how to create trips, each specifying a vehicle's origin, destination, and the time at which it should be introduced into the traffic simulation. Now, we explain how to generate routes. A route consists of a sequence of edges that define the specific sections of the road network a vehicle must traverse during the simulation. In other words, a route must be assigned to each trip to ensure that vehicles can navigate from their origin to their destination.

Although in the scientific literature there are several vehicular routing algorithms [6], herein we will use the routing method provided by SUMO through the `duarouter` tool. By default, this tool uses the Dijkstra to generate routes, but there are other methods available: A*, CH (Contraction Hierarchies), CH Wrapper.

The following command shows how to use the `duarouter` tool to generate routes:

```
> duarouter --net-file osm.net.xml --route-files od2trips.out.xml
   --output-file duarouter.rou.xml
```

where `--net-file` is the file name of the road network, `--route-files` it the file containing the trips.

The routing process can also be performed iteratively by using the `duaIterate.py`[12] script. This executes iteratively the following steps, in order:

1. Execute `duarouter` to perform the (re-)routing
2. Calling SUMO to simulate travel times

To optimize traffic flow, the `duaIterate.py` tool executes several simulations (the number of simulation can be configured). For each simulation, the tool finds a set of alternative routes that reduce the travel time for all vehicles. These routes are defined based on the traffic information produced by the simulation.

The following step consists of choosing routes in a set of alternatives. SUMO includes two route choice models: Gawron and Logit, both considering a weight/cost function.

Gawron proposes a probabilistic approach in which each route choice is modeled by a discrete probability distribution [4]. The algorithm takes in input the following parameters:

- the travel time along the used route in the previous simulation step,
- the sum of edge travel times for a set of alternative routes,

[12]https://sumo.dlr.de/docs/Tools/Assign.html#duaiteratepy

- the probability of choosing a route in the previous simulation step.

In each simulation, a driver d chooses a route r from a set R_d (in which the driver is allowed to transit) according to the probability distribution p_d. Each route is associated with a cost $c_d : R_d \to \mathcal{R}_+$; this function estimates the time required by the driver to reach the destination. The driver cannot know in advance the amount of traffic in the road network; so, the idea is that the driver evaluate an estimate of the time needed, and updates incrementally (at each simulation) its knowledge about the time required to reach a certain destination.

In Logit method, the required amount of time to travel each road is calculated according to only the information from the last simulation: it ignores old costs and old probabilities and takes the route cost directly as the sum of the edge costs from the last simulation:

$$c'_r = \sum_{e \in r} w(e) \tag{1}$$

where c'_r is the updated cost of route r, $w(e)$ is the weight of the edge e calculated from the input weight/cost function. The probability p'_r for each route r is calculated from an exponential function with parameter θ scaled by the sum over all route values:

$$p'_r = \frac{\exp(\theta c'_r)}{\sum_{s \in R} \exp(\theta c'_s)} \tag{2}$$

where c'_r is the cost of route r, $w(e)$ is the weight of the edge e calculated from the input weight/cost function.

The following command shows how to use the `duaIterate` tool to generate optimized routes:

```
> python $SUMO_HOME/tools/assign/duaIterate.py -n road_net.xml
    -t trips.rou.xml -l 2
```

where `-n` is the file name of the road network, `-t` it the file containing the trips, `-l` the number of iterations (using a number of iterations of 1 is equivalent to calling `duarouter`). By default, the Gawron model is used as the default route choice model.

2.5 Running the simulation (step ❺)

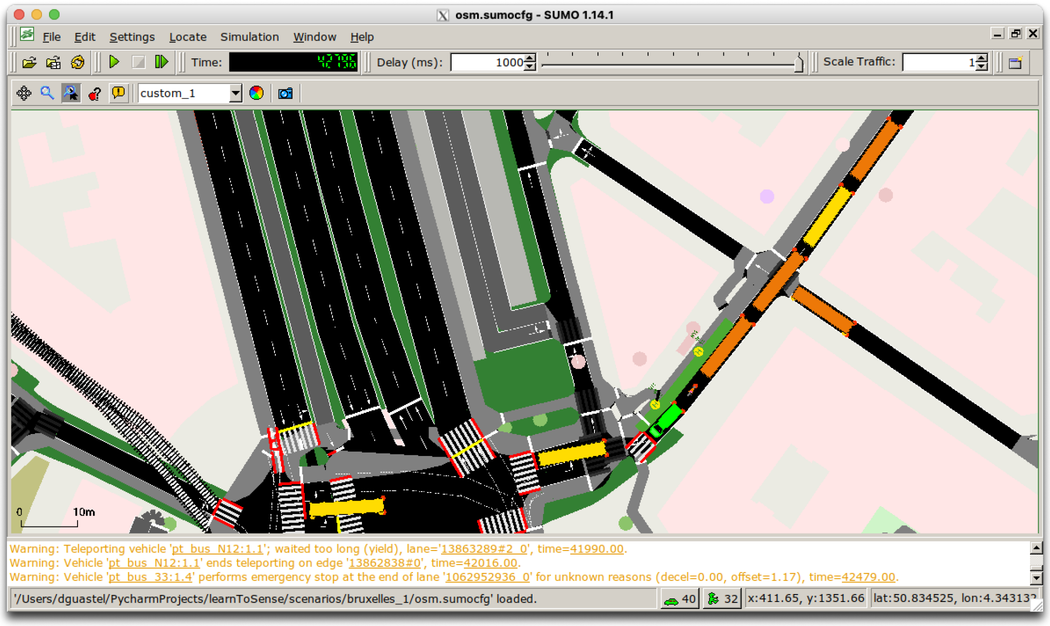
SUMO provides two ways of executing simulations: from the command line (using the `sumo` command) and from the graphical interface (using the `sumo-gui` command). Both commands use the same set of parameters, including the road network, routes, and the time horizon over which the simulation is conducted. Alternatively, a unique configuration file can be provided as parameter (using the parameter`-c`), containing all the required parameters and files for running a simulation. Following, we show an example of a configuration file:

```
<configuration>
    <input>
        <net-file value="test.net.xml"/>
        <route-files value="test.rou.xml"/>
        <additional-files value="test.add.xml"/>
    </input>
</configuration>
```

SUMO can be launched using the following command (assuming that the previous configuration file is named test.sumocfg):

```
> sumo -c test.sumocfg
```

Optionally, SUMO provides a graphical interface, available using the `sumo-gui` command (instead of `sumo`). The interface provides a graphical way to monitoring traffic during the simulation. Figure 13 shows the SUMO graphical user interface.



Time:

Delay (ms): 1000

Scale Traffic: 1

custom_1

```
Warning: Teleporting vehicle 'pt_bus_N12:1.1'; waited too long (yield), lane='13863289#2_0', time=41990.00.
Warning: Vehicle 'pt_bus_N12:1.1' ends teleporting on edge '13862838#0', time=42016.00.
Warning: Vehicle 'pt_bus_33:1.4' performs emergency stop at the end of lane '1062952936_0' for unknown reasons (decel=0.00, offset=1.17), time=42479.00.
'/Users/dguastel/PycharmProjects/learnToSense/scenarios/bruxelles_1/osm.sumocfg' loaded.   40   32   x:411.65, y:1351.66   lat:50.834525, lon:4.343133
```

Figure 13: SUMO graphical user interface.

3 Output Configuration (step ⑥)

SUMO allows generating output files containing several measures about traffic. These output files can be used to analyze traffic patterns or congestion phenomena.

A complete list of the output files that can be produced by the simulator is available at the SUMO website[13].

3.1 Virtual Traffic Monitoring Sensors

SUMO allows adding virtual monitoring devices that produce traffic counts. This is to simulate the behavior of traffic monitoring devices such as camera or induction loops. One type of virtual sensor, known in SUMO as **laneAreaDetector**, monitor traffic along one specific lane. Their functioning is similar to traffic cameras.

A laneAreaDetector can be defined using the `netedit`[14] tool, or using a specific XML definition. Figure 14 shows a map of Brussels city where multiple laneAreaDetector sensors (in teal color) have been placed arbitrarily.

After the definition of virtual sensors in `netedit`, the set of sensors can be exported as an XML file, which has the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<additional xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/additional_file.xsd">
    <laneAreaDetector id="e2_0" lane="-159327011#2_0" pos="0.0"
                length="138.630000" period="100" file="detectors.out.xml"/>
    <laneAreaDetector id="e2_1" lane="15497309#1_0" pos="0.0"
                length="1.940000" period="100" file="detectors.out.xml"/>
        ...
</additional>
```

where the parameter `file` is the path to the file that contains the traffic counts produced by virtual sensors. To use virtual sensors in SUMO, the XML file containing their definition must be provided as input to the

[13] `https://sumo.dlr.de/docs/Simulation/Output/index.html`
[14] https://sumo.dlr.de/docs/Netedit/index.html

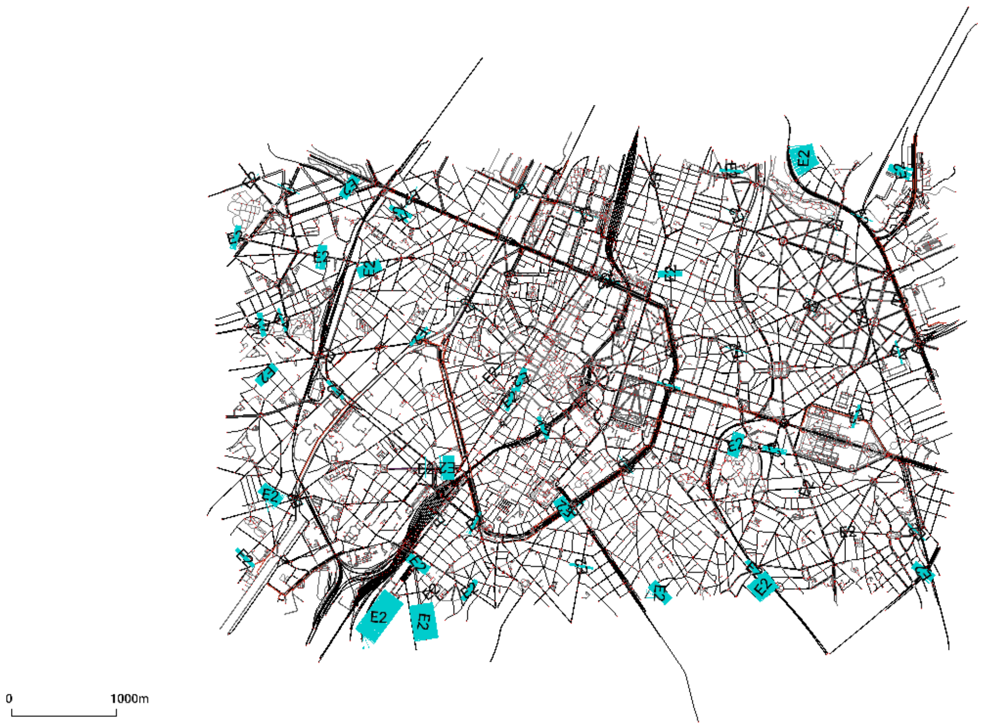


Figure 14: Position of lane area detectors (virtual sensors), colored in green, in a model of the city of Brussels.

simulator. At the end of a simulation, the file indicated in field `file` (in the previous example, "detectors.out.xml") contains the traffic counts observed by each sensor during the indicated time period. Table 1 reports the list of information generated by a virtual sensor.

Table 1: Subset of data types returned by each laneAreaDetector at the end of a simulation.

Name	**Unit**	**Description**
begin	(simulation) seconds	The first time step the values were collected in
end	(simulation) seconds	The last time step + DELTA_T the values were collected in (may be equal to begin)
id	id	The id of the detector (needed if several detectors share an output file)
meanSpeed	m/s	The mean velocity over all collected data samples.
meanTimeLoss	s	The average time loss per vehicle in the corresponding interval. The total time loss can be obtained by multiplying this value with nVehSeen.
meanOccupancy	%	The percentage (0-100%) of the detector's place that was occupied by vehicles, summed up for each time step and averaged by the interval duration.
maxOccupancy	%	The maximum percentage (0-100%) of the detector's place that was occupied by vehicles during the interval.
maxVehicleNumber	#	The maximum number of vehicles that were on the detector area during the interval.

To use the virtual sensors in a simulation, the path to the XML file containing the definition of laneAreaDetectors must be specified inside the `<additional>` tag of a SUMO configuration file (*.sumocfg), or passed as parameter (`-a`).

We developed a tool in python language that enables generating random laneAreaDetectors in a given road network[15]. The tool performs the following steps, in order:

1. extract the TAZs for the modeled road network (see Section 2.2);
2. extract all the edges and assign them to TAZs;
3. For each TAZ:
   (a) choose one random location where to put a laneAreaDetector with probability p;
   (b) Place the laneAreaDetectors into the network.

The probability p is calculated according to two strategies:

- **by number of lanes**: an edge has a probability of getting a virtual sensor proportional to the number of lanes.
- **by weight**: the probability of an edge of getting a virtual sensor depends on a parameter specified manually, and which values are in the 'edgedata' output file produced by SUMO[16].

4 Tools for the Automatic Definition of Scenarios

4.1 SUMO OSM Web Wizard

The OSM Web Wizard[17] is a web-based tool that offers an easy solutions to start modeling traffic scenarios with SUMO. The user can specify the area to model graphically through an openstreetmap map excerpt, and configure a randomized traffic demand and run and visualize the scenario in the sumo-gui. To run this tool, the following command must be run:

```
> python $SUMO_HOME/tools/osmWebWizard.py
```

Figure 15 shows the OSM Web Wizard interface.

4.2 SAGA

SUMO Activity GenerAtion (SAGA) is a user-defined activity-based multimodal mobility scenario generator for SUMO [2]. SAGA is capable of handling activity-based mobility from detailed information on the environment (e.g., buildings, PoIs), as well as the transportation infrastructure. SAGA is capable of extracting environmental information available automatically (from OSM) such as building and public transportation lines, generating all the configuration files required by SUMO, and fills the missing information with sensible default values. The output scenario is multi-modal, that is, including different types of transportation mean for population, and includes also parking areas, buildings, Points of Interest (PoIs).

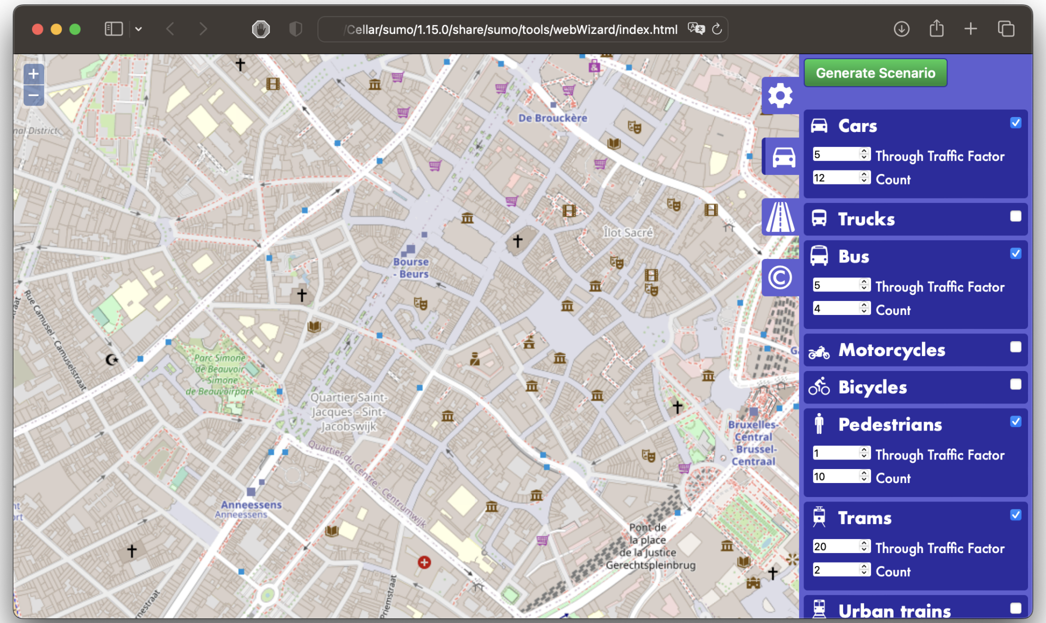
References

[1] J. Argota Sánchez-Vaquerizo. Getting Real: The Challenge of Building and Validating a Large-Scale Digital Twin of Barcelona's Traffic with Empirical Data. *ISPRS International Journal of Geo-Information*, 11(1):24, December 2021.

[2] L. Codeca, J. Erdmann, V. Cahill, and J. Haerri. SAGA: An Activity-based Multi-modal Mobility ScenarioGenerator for SUMO. *SUMO Conference Proceedings*, 1:39–58, June 2022.

[3] S. Dorokhin, A. Artemov, D. Likhachev, A. Novikov, and E. Starkov. Traffic simulation: an analytical review. *IOP Conference Series: Materials Science and Engineering*, 918(1):012058, September 2020.

[15] https://gitlab.com/traffic-sim/random-lane-detector-placer
[16] https://sumo.dlr.de/docs/Simulation/Output/Lane-_or_Edge-based_Traffic_Measures.html
[17] https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html


Cellar/sumo/1.15.0/share/sumo/tools/webWizard/index.html
Generate Scenario
De Brouckère
Ilot Sacré
Bourse - Beurs
Parc Simone de Beauvoir - Simone de Beauvoirpark
Quartier Saint-Jacques - Sint-Jacobswijk
Bruxelles-Central - Brussel-Centraal
Anneessens
Pont de la place de la Justice Gerechtspleinbrug

Cars ✓
5 Through Traffic Factor
12 Count

Trucks

Bus ✓
5 Through Traffic Factor
4 Count

Motorcycles

Bicycles

Pedestrians ✓
1 Through Traffic Factor
10 Count

Trams ✓
20 Through Traffic Factor
2 Count

Urban trains

Figure 15: SUMO OSM Web Wizard user interface.

[4] C. Gawron. An Iterative Algorithm to Determine the Dynamic User Equilibrium in a Traffic Simulation Model. *International Journal of Modern Physics C*, 09(03):393–407, May 1998.

[5] K. Kušić, R. Schumann, and E. Ivanjko. A digital twin in transportation: Real-time synergy of traffic data streams and simulation for virtualizing motorway dynamics. *Advanced Engineering Informatics*, 55:101858, 2023.

[6] V. Tran Ngoc Nha, D. Soufiene, and J. Murphy. A comparative study of vehicles' routing algorithms for route planning in smart cities. In *2012 First International Workshop on Vehicular Traffic Management for Smart Cities (VTM)*, pages 1–6, 2012.

A Full Step-by-Step Examples

A.1 Example 1

Step 1 – **Network generation**

```
netgenerate --rand -o MySUMOFile.net.xml --rand.iterations=200
       -j traffic_light --random
```

Step 2 – **Random trips generation**

```
python $SUMO_HOME/tools/randomTrips.py -n MySUMOFile.net.xml -e 3600
       -o trips.rou.xml --random
```

Step 3 – **Routing**

```
duarouter --net-file MySUMOFile.net.xml --route-files trips.rou.xml
       --output-file duarouter.rou.xml --ignore-errors
```

Step 4 – **Configuration file generation**

```
echo "<configuration>
   <input>
      <net-file value=\"MySUMOFile.net.xml\"/>
      <route-files value=\"duarouter.rou.xml\"/>
   </input>
</configuration>" | tee -a sumo.sumocfg
```

Step 5 – **Simulation**

```
sumo-gui -c sumo.sumocfg
```

A.2 Example 2

Step 1 – **Network generation**

```
netgenerate --rand -o MySUMOFile.net.xml --rand.iterations=300
       -j traffic_light --random --rand.grid
```

Step 2 – **Random trips generation**

```
python $SUMO_HOME/tools/randomTrips.py -n MySUMOFile.net.xml -e 3600
       -o trips.rou.xml --random --random-depart --binomial 20
```

Step 3 – **Routing**

```
duarouter --net-file MySUMOFile.net.xml --route-files trips.rou.xml
       --output-file duarouter.rou.xml --ignore-errors --route-choice-method logit
```

Step 4 – **Configuration file generation**

```
echo "<configuration>
   <input>
      <net-file value=\"MySUMOFile.net.xml\"/>
      <route-files value=\"duarouter.rou.xml\"/>
   </input>
</configuration>"  tee -a sumo2.sumocfg
```

Step 5 – **Simulation**

```
sumo-gui -c sumo2.sumocfg
```

A.3 Example 3

Step 1 – **Get the map from OSM**

```
python $SUMO_HOME/tools/osmGet.py --bbox="4.3583, 50.8362, 4.3696, 50.8455"
```

Step 2 – **OSM map to SUMO network**

```
python $SUMO_HOME/tools/osmBuild.py --osm-file osm_bbox.osm.xml
```

Step 3 – **Extract the polygons from the OSM file**

```
polyconvert --net-file osm.net.xml --osm-files osm_bbox.osm.xml
       --type-file $SUMO_HOME/share/sumo/data/typemap/osmPolyconvert.typ.xml
       -o polygons.poly.xml
```

$\boxed{\textbf{Step 4}}$ – **Generate random trips**

```
python $SUMO_HOME/tools/randomTrips.py -n osm.net.xml -e 3600 -o trips.rou.xml
       --random --random-depart --binomial 10 --validate
```

$\boxed{\textbf{Step 5}}$ – **Routing**

```
duarouter --net-file osm.net.xml --route-files trips.rou.xml
       --output-file duarouter.rou.xml --ignore-errors true
```

$\boxed{\textbf{Step 6}}$ – **Configuration file generation**

```
echo "<configuration>
   <input>
      <net-file value=\"osm.net.xml\"/>
      <route-files value=\"duarouter.rou.xml\"/>
      <additional-files value=\"polygons.poly.xml\"/>
   </input>
</configuration>"  tee -a sumo.sumocfg
```

$\boxed{\textbf{Step 7}}$ – **Simulation**

```
sumo-gui -c sumo2.sumocfg
```

A.4 Example 4

$\boxed{\textbf{Step 1}}$ – **Scenario Definition**

```
python $SUMO_HOME/tools/osmGet.py --bbox="4.3583, 50.8362, 4.3696, 50.8455"
python $SUMO_HOME/tools/osmBuild.py --osm-file osm_bbox.osm.xml
```

$\boxed{\textbf{Step 2}}$ – **TAZs Extraction**

```
polyconvert --net-file osm.net.xml --osm-files osm_bbox.osm.xml
   --type-file $SUMO_HOME/share/sumo/data/typemap/osmPolyconvert.typ.xml
   -o polygons.taz.xml --type taz
python $SUMO_HOME/tools/edgesInDistricts.py
   -n osm.net.xml -t polygons.taz.xml -o TAZ.xml
   -l passenger --complete
```

$\boxed{\textbf{Step 3}}$ – **Random Traffic Generation**

```
python $SUMO_HOME/tools/randomTrips.py -n osm.net.xml
   -b 0 -e 3600 -o trips.rou.xml --random --random-depart
   --binomial 10 --validate
```

$\boxed{\textbf{Step 4}}$ – **Random traffic to OD-matrix**

```
python $SUMO_HOME/tools/route/route2OD.py
   -r trips.rou.xml -a TAZ.xml -o routes.xml -i 5
```

$\boxed{\textbf{Step 5}}$ – **Importing OD-matrix**

```
od2trips -z routes.xml -n TAZ.xml -b 0 -e 3600
   -o od2trips.out.xml
```

Step 6 – **Routing and Simulation**

```
duarouter --net-file osm.net.xml --route-files od2trips.out.xml
    --output-file duarouter.rou.xml --ignore-errors true
sumo-gui +a TAZ.xml -n osm.net.xml -r od2trips.out.xml -b 0
    -e 3600 --edgedata-output edgedata.outsampler.xml --ignore-route-errors
```

A.5 Example 5

This example shows how to generate an inter-modal traffic scenario. We assume that the file "map.osm" is the map downloaded from OSM.

Step 1 – **OSM to SUMO, Public Transportation Data Extraction**

```
netconvert --osm-files map.osm -o bxl.net.xml --osm.stop-output.length 20
    --ptstop-output additional.xml --ptline-output ptlines.xml
```

Step 2 – **Find Travel Times and Create Public Transportation Schedules**

```
python $SUMO_HOME/tools/ptlines2flows.py -n bxl.net.xml -s additional.xml
    -l ptlines.xml -o flows.rou.xml -p 600 --use-osm-routes --ignore-errors
```

Step 3 – **Generate Random Traffic for Vehicles**

```
python $SUMO_HOME/tools/randomTrips.py -n bxl.net.xml -b 0 -e 3600
    -o passenger_trips.rou.xml --random --random-depart --validate
    --vehicle-class passenger
```

Step 4 – **Vehicles Routing**

```
duarouter --net-file bxl.net.xml --route-files passenger_trips.rou.xml
    --output-file duarouter_passenger.rou.xml --ignore-errors
```

Step 5 – **Simulation**

```
sumo-gui -n bxl.net.xml -r duarouter_passenger.rou.xml,flows.rou.xml -b 0
    -e 3600 --edgedata-output edgedata.outsampler.xml
    -a additional.xml --ignore-route-errors
```

B Reverting Road Direction in OSM

The following python method can be used to invert the direction of the edges whose ID is given in input. This method works only if the SUMO road network has been converted from the OSM using the `--output.original-names`.

```
def reverse_roads(edge_ids:list[String], osm_path:String) -> None:
tree = ET.parse(osm_path)
root = tree.getroot()
for child in root:
    if "id" not in child.attrib:
        continue
        for edge_id in edge_ids:
            if re.search(child.attrib["id"], edge_id):
                found_oneway_tag = False
                for tag_d in list(child.iter('tag')):
                    if tag_d.attrib["k"] == 'oneway':
                        found_oneway_tag = True
                        if tag_d.attrib['v'] == '-1':
                            tag_d.attrib['v'] = 'yes'
                        else:
                            tag_d.attrib['v'] = '-1'
                if not found_oneway_tag:
                    # maybe it's a 2direction street
                    removed = False
                    for tag_d in list(child.iter('tag')):
                        if tag_d.attrib["k"] == 'lanes' or tag_d.attrib["k"] == 'oneway':
                            child.remove(tag_d)
                            removed = True
                    if not removed:
                        item = ET.SubElement(child, "tag")
                        item.attrib["k"] = "lanes"
                        item.attrib["v"] = str(len(re.findall(child.attrib["id"], edge_id)))
                        item = ET.SubElement(child, "tag")
                        item.attrib["k"] = "oneway"
                        item.attrib["v"] = "-1"

    with open("{}.rev.osm".format(osm_path), 'wb') as f:
        tree.write(f, encoding='utf-8')
```

The method takes in input a list of SUMO edges ID, and the full path to the OSM map. A for loop iterates over all the OSM nodes having an ID. Then, we check if the ID of the current node is present in the input list `edge_ids`. If so, The node corresponds to an edge whose direction must be inverted. If the road is one way, we change the value from -1 to 'yes' (or the opposite) to invert the direction. If the road has more than 2 lanes, then we search for the XML tag 'oneway', and if it's found, it is removed from the XML tree. Otherwise, we change the value of 'oneway' attribute as before.

C SUMO Installation

Detailed installation instructions are available in the SUMO website: `https://sumo.dlr.de/docs/Installing/index.html`

For OSX, we suggest using Homebrew (`https://brew.sh/`) to install SUMO. We recommend to use the following commands, in order:

```
>       brew install --cask xquartz
>       brew tap dlr-ts/sumo
>       brew install --with-examples --with-ffmpeg
                --with-gdal --with-gl2ps --with-swig sumo
```

After the installation, the system variable `SUMO_HOME` must be set (system-wide) to the path containing SUMO. Typically the path is as the following: `/opt/homebrew/Cellar/sumo/1.XX.XX/`.

D Common Issues

D.1 Broken Folder Links When Using SUMO on OSX

When using SUMO installed through homebrew into the path `/opt/homebrew/Cellar/sumo/1.XX.XX/`, the following commands must be run to fix broken links in SUMO folders:

```
sudo ln -s $SUMO_HOME/share/sumo/tools/ $SUMO_HOME/
sudo ln -s $SUMO_HOME/share/sumo/data $SUMO_HOME/
```

5 Useful Resources

- `https://www.eltis.org/sites/default/files/tool/conduits_key_performance_indicators_its.pdf`
- `https://www.arcgis.com/home/item.html?id=f40c3fdb26c74e64af4f2cb8311c5559`
- `https://monitoringdesquartiers.brussels/maps/statistiques-population-bruxelles/evolution-populatio`
  `densite-de-population/1/2019/#`
- `https://www.acea.auto/files/ACEA-report-vehicles-in-use-europe-2022.pdf`
- Number of households with cars (Belgium): `https://statbel.fgov.be/en/themes/datalab/vehicles-household`