SUMOPy user manual

Joerg Schweizer, University of Bologna, DICAM
2016-10-30
Version 1

Abstract

This document describes the capabilities and basic usage of the software SUMOPy. SUMOPy is intended to expand the user-base of the traffic micro-simulator SUMO (Simulation of Urban MObility `http://sumo.dlr.de/wiki/Main_Page`) by providing a user-friendly, yet flexible simulation suite.

A further scope of SUMOPy is to manage the huge amount of data necessary to run complex multi-modal simulations. This includes different demand generation models as well as a large range of modes, such as road transport, public transport, and bicycles. In the future also modes like self-driving cars and Personal Rapid Transit (PRT) will be supported.

SUMOPy consists of a GUI interface, network editor as well as a simple to use scripting language which facilitates the use of SUMO.

Contents

1 Introduction

SUMO rapidly developed into a flexible and powerful open-source microsimulator for multi-modal urban traffic networks [Krajzewicz (2003)]. The features and the number of tools provided are constantly increasing, making simulations ever more realistic. However, the different functionalities consist at the present state of a large number of binaries and scripts that act upon a large number of files, containing information on the network, the vehicles, districts, trips routes, configurations, and many other parameters. Scripts (mostly written in Python), binaries and data files exist in a dispersed manner. In practice, a master script is necessary to hold all processes and data together in order run a simulation of a specific scenario in a controlled way. This approach is extremely flexible, but it can become very time consuming and error prone to find the various tools, combine their input and output and generate the various configuration files. Furthermore, it reduces the user-base of SUMO to those familiar with scripting and command line interfaces. Instead, SUMO has the potential to become a multi-disciplinary simulation platform if it becomes more accessible to disciplines and competences.Scripts (mostly written in Python), binaries and data files exist in a dispersed manner.

This problem has been recognized and different graphical user interfaces have been developed. The *traffic modeller* (also named *traffic generator*) is a tool written in Java which helps to manage files, to configure simulations and to evaluate and visualize results.

SUMOPy is written entirely in the object-oriented script language *Python*, it uses *wxWindows* with *PyOPENGL* as GUI interface and *NumPy* for fast numerical array-type calculations. It is similar to the *traffic generator* in that it simplifies the use of SUMO through a GUI. But SUMOPy is more than just a GUI, it is a *suite* that allows to access SUMO tools and binaries in a simple unified fashion. The distinguishing features are:

- SUMOPy has Python instances that can make direct use of tools already available as Python code.
- SUMOPy has a Python command line interface that allows direct and interactive manipulation of SUMOPy instances.
- SUMOPy provides a library that greatly simplifies the scripting.

2 Installation

SUMOPy is a directory with python scripts. It is sufficient to unzip the latest version and copy it in a directory of your choice. Since SUMO-0.28, SUMOPy is inside the SUMO distribution and located in `SUMOHOME/tools/contributed`.

However, SUMOPy makes extensive use of Python packages which need to be installed before. The *required* packages to be installed are:

1. Python 2.7
2. numpy-1.10 or newer
3. wxPython2.8 or wxPython2.9 (wxPython3.x is currently not properly working with PyOpenGL-3.0.x)
4. PyOpenGL-3.0.x

The following packages are optional:

1. matplotlib-1.4 or newer, for high quality graphical output in different file formats.

2. PIL-1.1.7 or newer and basemap-1.0 (or pyproj) for downloading backround maps from mapservers.

The exact choice of package-versions and installation methods depend on the operating system. Below we give short recomandations regarding the choice of packages for different operating systems. In general, the 32-bit version is preferred as there are more pre-compiled packagesavaillable, but this may change over time.

2.1 Windows

For Windows, as required packages the following are recommended:

```
python-2.7.12.msi (32-bit or x86 preferred)
numpy-1.10.0-win32-superpack-python2.7.exe
wxPython2.8-win32-unicode-2.8.12.1-py27.exe
PyOpenGL-3.0.2.win32.exe
```

Optionally, install these:

```
matplotlib-1.4.3-cp27-none-win32.whl
basemap-1.0.8-cp27-none-win32.whl
PIL-1.1.7.win32-py2.7.exe
```

2.2 Linux

Python 2.7 comes with most Linux operating systems. All required additional packages are available in repositories:

```
python-numpy
python-wxgtk2.8
python-opengl
python-imaging
python-matplotlib
python-mpltoolkits.basemap
```

However, often Python 3.x is installed along the older version and may be the default Python interpreter. So make sure you run the sumopy scripts with Python 2.7

Another issue may be `python-wxgtk2.8` in repositories of more recent distributions, as for example Ubuntu-16.04. In this case, do the following safe operation to install `python-wxgtk2.8`:

```
echo "deb http://archive.ubuntu.com/ubuntu wily main universe"\
| sudo tee /etc/apt/sources.list.d/wily-copies.list

sudo apt install python-wxgtk2.8

sudo rm /etc/apt/sources.list.d/wily-copies.list

sudo apt update
```

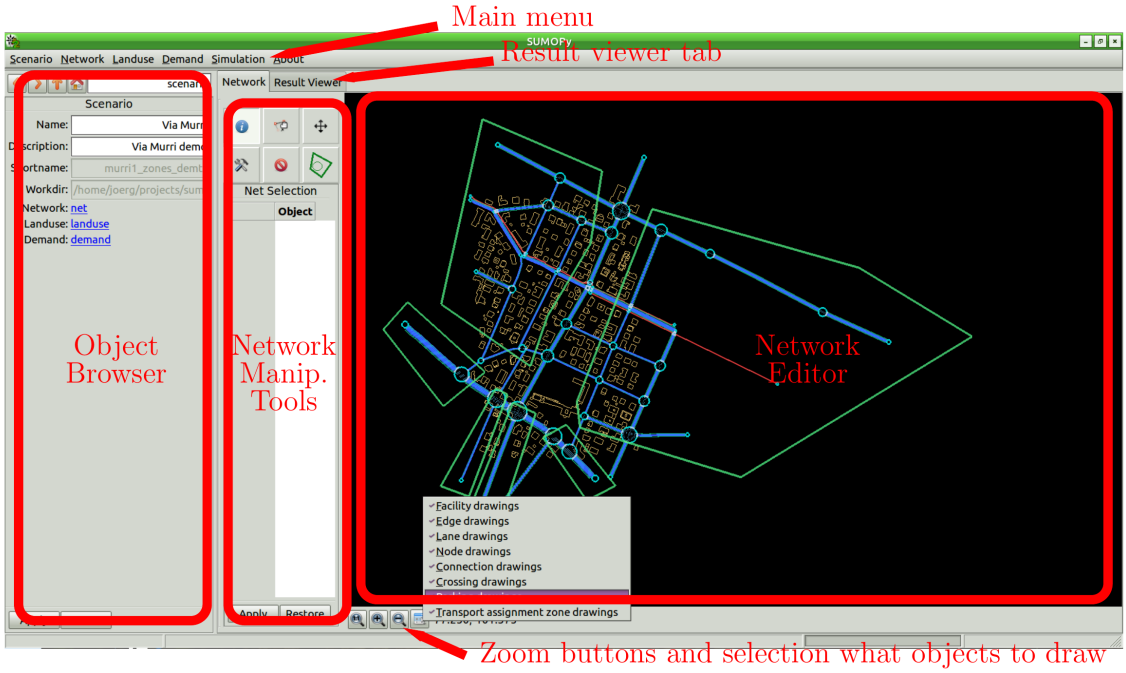

Figure 1: Main window of SUMOPy

3 The graphical user interface

3.1 Getting started!

3.1.1 Running SUMOPy

Start the script `sumopy_gui.py` by double-clicking on it in your browser. If this fails, use the command-line accessory change directory to

`SUMOHOME/tools/contributed/sumopy`

and run sumopy with

```
python sumopy_gui.py
```

If all required packages are installed correctly, you should see the main window as shown in Fig. 1, but initially with an empty network. The object browser shows initially the main object of SUMOPy: the *scenario*, which contains all other information.

3.1.2 Opening/creating a scenario

There is a test scenario in the SUMOPy distribution which is located in

`SUMOHOME/tools/contributed/sumopy/testscenario`

The quickest way to obtain results from a simulation is to import already existing `xml`-files. In case the following SUMO network, poly and route files

`demo.net.xml` `demo.poly.xml` `demo.rou.xml`

are located in directory

`SUMOHOME/tools/contributed/sumopy/testscenario`

then a new scenario can be created by importing these files into SUMOPy at ones: from the main menu, choose `Scenario>Create from xml...` and insert the scenario Shortname, and Workdir in the form as shown in Fig. 2 The options `Name` and `Description` are free text fields. After

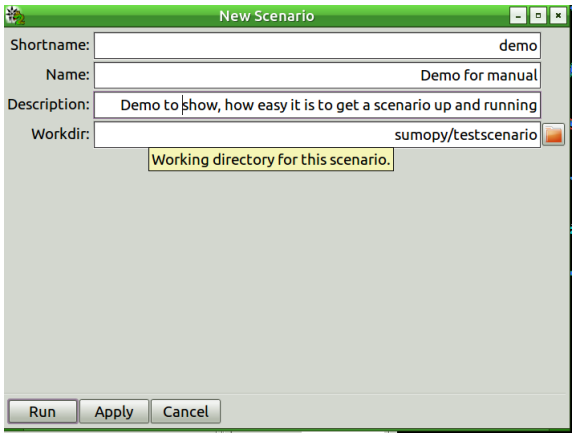

Figure 2: Creating a new scenario from SUMO xml files. Please note that the scenario short-name must be identical to the root-name of the xml files.

pressing the `Run` button, network, buildings and routes will be imported. In case the trip file `demo.trip.xml` exists, it will also be imported.

In the same way it is possible to crate an empty scenario under menu `Scenario>New...`.

3.1.3 Browsing the scenario

The object browser allows to navigate through all information of a scenario. To a certain extend, it is possible to modify data. The most important information are:

- The network with edges, nodes, traffic light systems etc.
- The landuse, containing also the building information from the `.poly` file as well as background maps (see later how to import them).
- The demand holds information on available vehicles types, trips and routes.

3.1.4 Navigating the Network

The network can be examined with the network editor. The initial editing tool allows to click on the different network elements and retrieve the respective information in the object browser.

With the zoom-buttons (+,-) located below the network editor, different zoom levels can be obtained The 1:1 button zooms the network to fit approximately the boundaries of the window. Next to the zoom button is a button which pops up a menu when pressed. From this menu, the network elements to be drawn can be selected or un-selected.

The following mouse-key combination allow to navigate the network:

Action	Key-Mouse
Zoom in/Out	Hold down <CTRL> + <Wheel>
Panning	Hold down <CTRL> + <SHIFT> + <Button-Left>

3.1.5 Running a simulation

From the main menu select:

`Simulation>Sumo>export routes and simulate...`

With this process, the current trips and routes in `demand.trips` will be automatically exported to a SUMO `.rou` file. Choose the desired simulation settings from the SUMO pop-up dialog, as shown in Fig. 3.

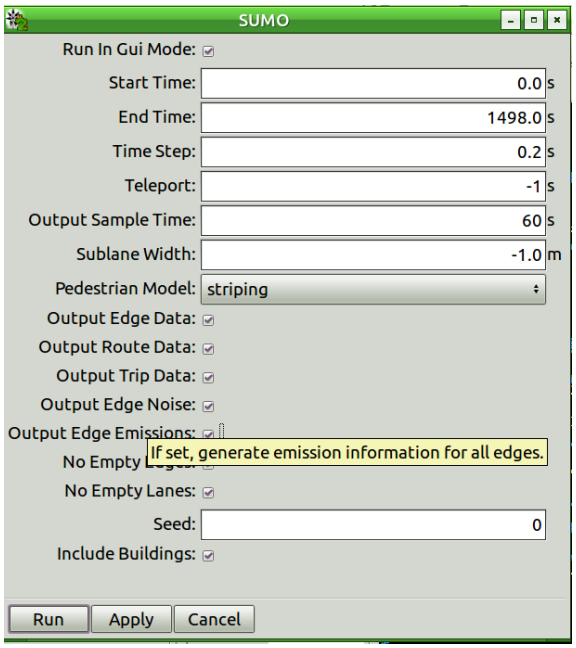

Figure 3: SUMO simulation dialog. Here all output data options are marked.

The simulation parameters are self-explaining, just hover with the mouse over the parameter name. The default parameters are typically suitable to run a first simulation. Select one of the output options in order to obtain specific simulation results. Regarding outputs, the `Output Sampling Time` for the different outputs may be of interest. If you observe in the simulations that vehicles get blocked at junctions for no obvious reason, then it is possible to resolve conflicts by setting the `Teleport` to a positive time (i.e. 10s).

After pressing the `run` button, the SUMO-GUI interface pops up, ready for simulation, as shown in Fig. 4. Adjust delay time and press the Start button. At the end of the simulation, confirm OK and close the SUMO-GUI window. Simulation results are now imported into SUMOPy and are ready to be examined, visualized and exported in various formats.

3.1.6 Viewing results

The results can be viewed in table format and graphically, see Fig. 5.

In the object browse, the results per trip and the results per edge can be viewed. Both, trip and edge oriented result-table can be exported in

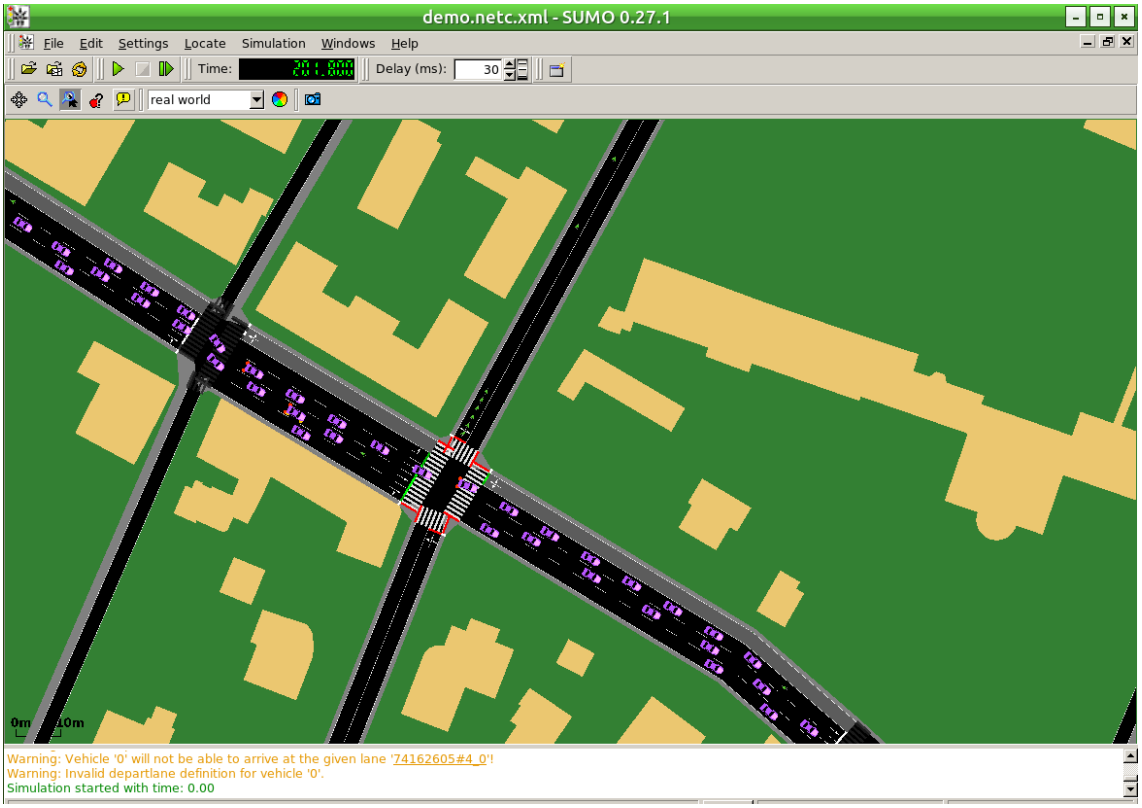

Figure 4: The SUMO-GUI. Do not forget to increase the delay time, otherwise vehicles may become too fast to be spotted.

CSV format, see the `Simulation>Results` menu.

Attention: Results are *not* saved when saving the scenario. Instead the results must be saved separately using

`Simulation>Results>Save as ...`

At any time, results can be reopened with the scenario with which they have been produced, using:

`Simulation>Results>Open ...`

In case the `Matplotlib` package is installed you can generate plots in various formats, choosing menu `Simulation>Results>Plot with matplotlib`. The pop-up dialog helps to configure the graphical details of the plot, an example is shown in Fig. 6.

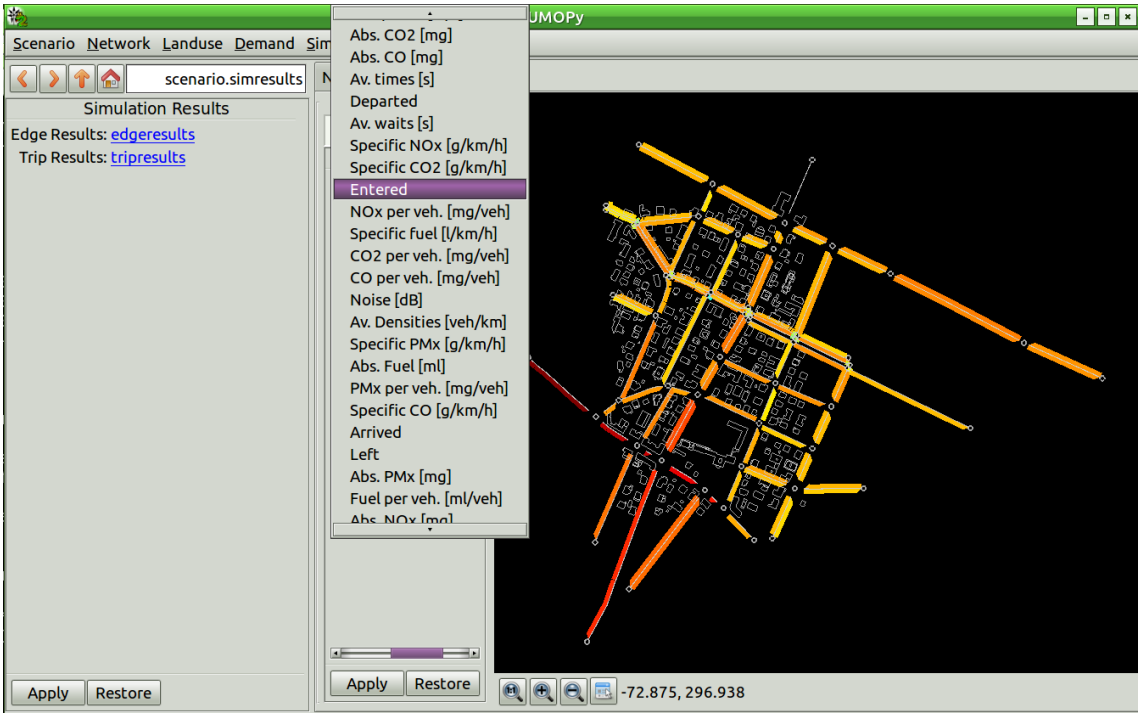

Figure 5: The SUMOPy result viewer. Edge and trip-oriented results in table-form can be viewed in the object browser. Edge result are visualized on a map to the right. Select the "Edge Quantity" from the menu to be plotted.

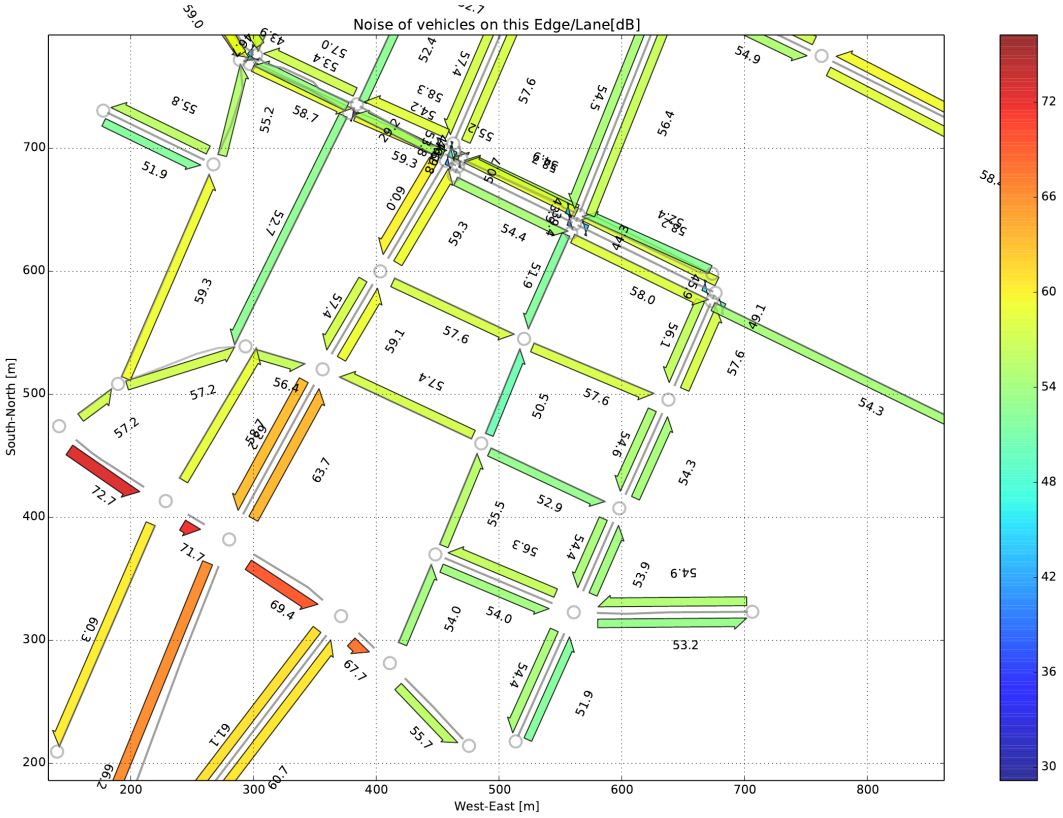

Figure 6: Example of a Matplotlib plot generated bu SUMOPy.

3.2 Launching SUMOPy

As explained in Sec. 3.1, networks can be created from SUMO XML files, or a binary file can be loaded. Both operations can be performed from the commend line.

3.2.1 Loading a binary scenario at start

A previously saved, binary scenario with filename `scenario.obj` can be loaded into SUMOPy at start using the commandline

```
python sumopy_gui.py  workdir/scenario.obj
```

3.2.2 Importing SUMO XML files at start

An initial import of existing SUMO XML files with rootname `scenario` and located in directory `workdir` can be accomplished with

```
python sumopy_gui.py  scenario workdir
```

3.3 Importing and Editing networks

Important notice: if needed, the network should be modified *before* moving on to demand modeling (see Sec.3.4).

3.3.1 Importing nets and facilities

Networks can be import from a SUMO net.xml file with

```
network>import>from sumo net.xml ...
```

Networks can be converted and imported from a previously downloaded OSM file, calling a wizard with:

```
network>import>from osm.xml ...
```

Note: the information on buildings (called "facilities") are a property of the landuse object and can be extracted and imported from an OSM file with menu item

```
landuse>facilities>import from osm...
```

Sometimes special characters in the XML file are offending the python XML parser, leading to errors. If this occurs, simply "clean" the OSM file with

```
landuse>facilities>clean osm file...
```

prior to importing it.

3.3.2 Editing with SUMO's Netedit

The recommended way to edit the network is via `netedit`, which is provided with SUMO from version 0.25. SUMO's `netedit` can be called by choosing the menu

```
Network>Edit with netedit
```

`netedit` will be fired up with the network ready to be edited. The use of `netedit` is documented here: `http://sumo.dlr.de/wiki/NETEDIT`. After editing, the network must be saved within `netedit` be pressing `<CTRL>`-s or with `File>save`. Then `netedit` can be closed and the modified network will be reimported into SUMOPy.

3.3.3 Editing with SUMO's Netedit on background maps

There is a possibility to edit the network with `netedit` (as explained in Sec. 3.3) on background maps. Currently these are Google Satellite maps. Before editing with background maps, the maps must be downloaded for the given network area. This is accomplished by a wizard which can be called selecting the menu item[1].

`landuse>maps>download...`

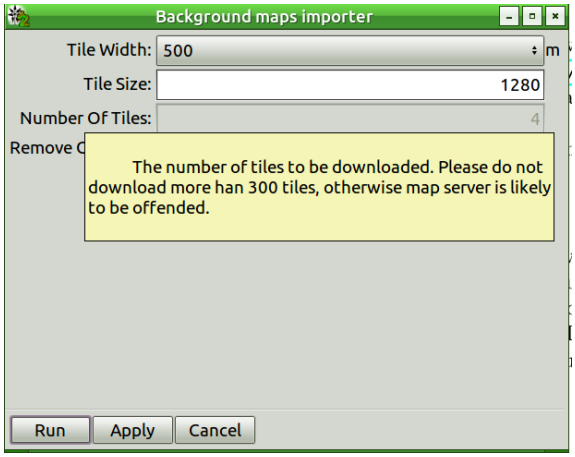

Figure 7: Map download wizard.

The wizard, shown in Fig. 7 helps to download the squared tiles which cover the network area. The resolution can be defined by setting the tile width (and height) in meters. The tile size in pixel is maximum 1280 (using the Google Map server). The resolution is then tile size/tile width in pixel per meter.

Attention: it is highly recommended to press the `Apply` Button prior to downloading the maps. This will calculate the number of tiles which are going to be downloaded. This is important to know, because Google maps prevent IPs from downloading too many maps of an area. Therefore do not exceed $\approx$ 300 tiles! Note further that, dependent on the performance of your computer, `netedit` may run into problems to handle too large map areas.

After the maps are downloaded, `netedit` can be called with

`Network>Edit with netedit on map`

in order to edit the SUMO network on a map.

3.3.4 Editing with SUMOPY Neteditor

The SUMOPy internal net-editor provides currently some limited capability to edit the network

- The geometry of edges and buildings can be manipulated: Activate the `Move` tool to move or the `Stretch` to change vertexes.
- With the menu `Network>Clean nodes` all edges entering a node are "cut back" to a certain radius. This measure may facilitate the verification of connections between lanes at junctions.

[1] In SUMOPy Maps are managed by the landuse object

3.4 Demand modeling

In the framework of SUMO, demand generation means essentially the generation of traffic participants (persons and/or different types of vehicles) and the generation of a route for each traffic participant. It is recommeded to start with demand generation only after the network has been edited. In any case it is good practice to save a scenario with the network only (without demand info).

All available vehicle types[2] can be browsed and modified under:

`scenario.demand.vtypes`

Note that each vehicle type belongs to a vehicle class, also called modes.

The different demand generation methods, as described below, will generate trips and routes, which can be browsed in

`scenario.demand.trips` and `scenario.demand.trips.routes`

In fact, the different demand generation methods could also be used in combination.

3.4.1 Zone to zone demand flows

We first explain the general concept of Zone to zone demand generation before we describe how to procede with SUMOPy.

1. *Zone definition*: Definition of Zones, in SUMO also called "Traffic Assignment Zones" (TAZ). A TAZ defines the area where participants depart (zone of origin) or arrive (zone of destination). A TAZ does typically contain several network edges.
2. *Zone-to-Zone flow definition*: Definition the number of trips between each zone of origin and a zone of destination (= OD-flow). This structure is widely known as the Origin-to Destination matrix OD Matrix. OD-flows are defined.
   - for different time intervals of the day.
   - for different transport mode.
3. *Trip generation*: Each OD-flow is disaggregated into a discrete number of individual trips, departing at different edges (and edge positions) within the zone of origin, and at different time instances within the specified time interval; and arriving at different edges (and edge positions) within the zone of destination.
4. *Routing*: A route is computed for each individual trip, connecting the edge within the zone of origin, with the edge within the zone of destination.

These steps can be performed with SUMOPy as follows.

Zone definition: In SUMOPy, Zone definition can be accomplished manually with the `Add zone tool` of the network editor, see Fig. 8. After giving the zone a name [3] a polygon can be drawn on the network with a series of `<Button-Left>` - clicks. Complete the zone with a final `<Double-Button-Left>` - click. A `<Double-Button-Right>` - click

[2] Of course in a traffic scenario we have usually more than one participant of each type

[3] zone names can added/modified later via browser

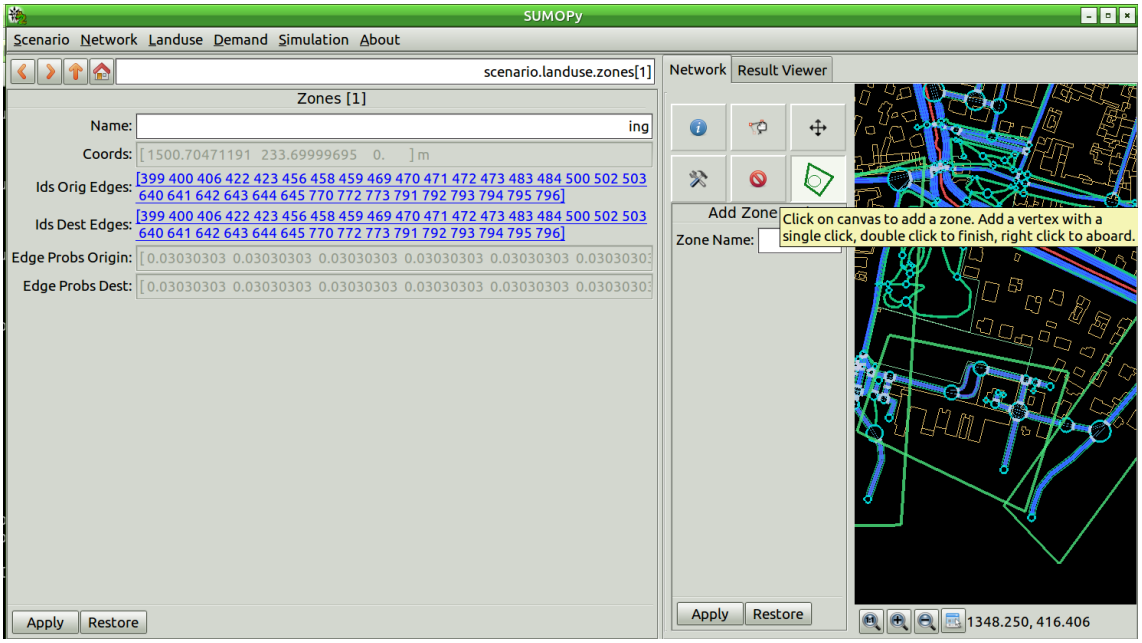

Figure 8: Traffic Assignment Zones in object browser and on network (in green). A new zone can be added with the Add Zone tool. On the object browser an existing zone is displayed. Note the identified zone edges available for departure and arrival of vehicles.

will aboard the current zone drawing. Currently zones must be convex, otherwise edge detection problems occur.

Important notice Only edges which are located entirely inside a zone are considered part of a zone. Only edges inside a zone are considered for departure or arrival of vehicles in the respective zone. After creating the zones you can identify all edges in each zone by selecting menu item

`Landuse>Zones>Identify zone edges`

It is possible to see all zone edges and and change zone names by using the information tool and by clicking on the green zone border. Zones are accessible under `scenario.demand.zones`. Zones can also be deleted with the `Delete tool` of the network editor.

Zone-to-Zone flow definition:

Zone-to-Zone flows can be added by selecting the menu item

`Demand>Zone-to-zone demand>Add zone-to-zone flows`

An "Add OD flow wizard" will pop up as shown in Fig. 9. On this wizard, specify the time interval (in entire seconds) and the transport mode. On the menu of the wizard select `Edit>Add OD-flow to table`. Then a new row will appear in the table. Enter the zones of origin and destination and the respective number of trips between them.

Instead of entering the OD flows manually, the wizard offers also the possible to import an OD-flows from a CSV file. Select from the wizard menu:

`File>Import CSV ...`

and choose a CSV file from the file-dialog window. The CSV file must have the following format:

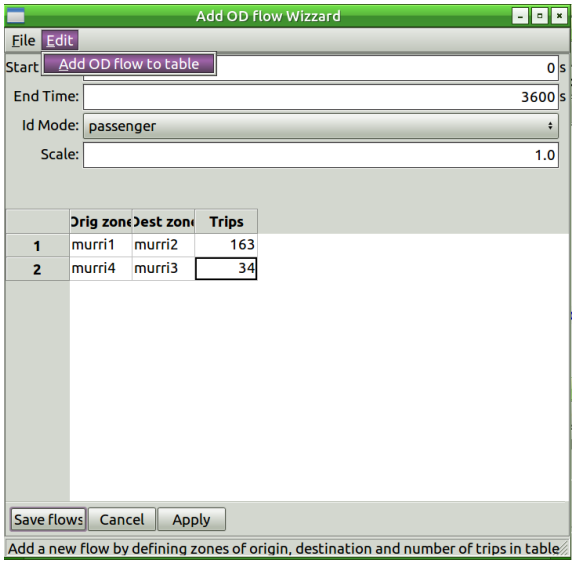

Figure 9: The OD flow wizard.

```
<zone name origin 1>, <zone name destination 1>,<trip number 1>
<zone name origin 2>, <zone name destination 2>,<trip number 2>
...
```

Once the OD flows are all entered, press `Save flows`. The scale factor can be used to multiply the entered trips with a constant (default 1) when saving. The demand flows are now saved to SUMOPy and can be browsed and modified under

`scenario.demand.odintervals`

Trip generation: Generate trips from OD flows, selecting menu item

`Demand>Zone-to-zone demand>Generate trips from flows`

The generated trips can be browsed and modified under

`scenario.demand.trips`

Routing: Perform a shortest path routing, for each trip by selecting

`Demand>Trips and router>Routing`

The generated route can be browsed under

`scenario.demand.trips.routes`

With this method, SUMO's duarouter is used to perform the routing. Note that each trip is now linked to a route (see ID route column in trips). If there is no route means that the edge in the zone of origin is probably not connected to the edge in the zone of destination [4]. The router does not route pedestrians. Their exact route will be determined only during simulation.

The scenario is now ready to be simulated by selecting

`Simulation>Sumo>export routes and simulate...`

[4] there can be several reasons for this, usually the destination edge is in access-restricted areas, or there are one-way roads, impeding access. Actually this should not happen too often, as the disaggregation algorithm should verify accessibility

Proceed as described in Sec. 3.1.5.

A test file for OD demand is located in

`SUMOHOME/tools/contributed/sumopy/testscenario/demo_dem_od_bikes.csv`

3.4.2 Turn flows

Trip generation with turnflows allows to model traffic flows in a precise way, for simple, possibly loop-free networks with few internal traffic generation. The general idea is to generate trips and routes for individual vehicles based on road traffic counts. These traffic counts can be performed at junctions. Ideally, the traffic counts should be performed simultaneously at all relevant junctions. However, some edge flows can be derived from other edge flows considering flow preservation lows at nodes.

Basically two types of flows are needed in order to reconstruct the routes in a predefined study area:

- The *generating flows* f_a on all edges a that *enter* the study area. [5]
- The *turn flows* f_{a_1,a_2} for all flows between links a_1 and a_2 at nodes with more than one exiting edge.

The choice of traffic flows that need to be counted is illustrated with the example network in Fig. 10. Neglecting all traffic generated within the

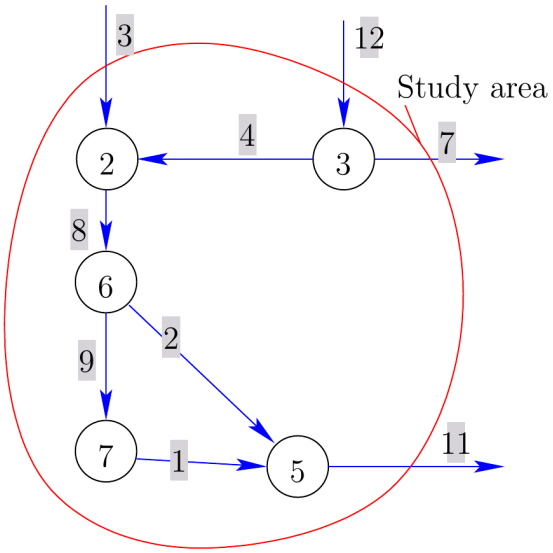

Figure 10: Network example to demonstrate the determination of generatin flows and turnflows.

study area, the following flows need to be counted:

- The *generating flows* f_3, f_{12} are entering the network.
- The *turn flows* $f_{8,9}, f_{8,2}$ at node 6 and $f_{12,4}, f_{12,7}$ at node 3. All other nodes have only one exiting edge.

[5] In addition, traffic flows can also be generated on edges within the study area, but in practice it is difficult to measure those flows because one would need to spot and count only departing vehicles along the roads of the study area. However, for some particular edges with many departing vehicles this may be necessary to do.

Once the necessary flows are counted for all considered transport modes and time intervals, the trips and routes of the vehicles can be generated by a special router. Below is explained how this process can be realized with SUMOPy.

Defining generating flows and turn-flows:

Import flows and turnflows from a CSV file by selecting the menu item

`Demand>Turnflows>Import turnflows`

The import turnflows dialog will pop up, as shown in Fig. 11. This

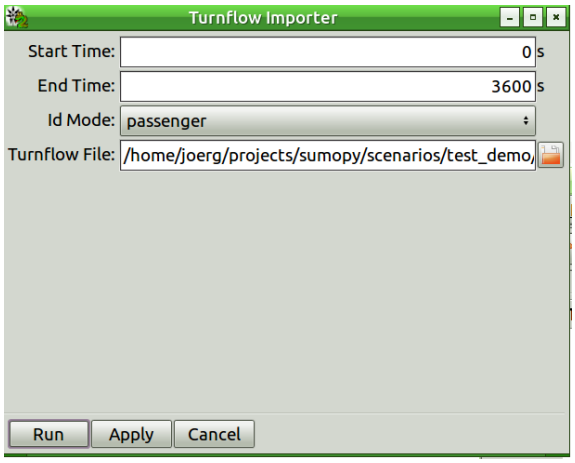

Figure 11: The import turnflows dialog.

dialog is similar to the OD-flow wizard explained in Sec. 3.4.1 and allows to import traffic counts, and associate them with a time interval and a transport mode.

The `Turnflow File` to be imported contains the traffic counts of both flow types, generating flows and turnflows. The turnflow file has the following format:

```
<ID1>, <count 1>, <ID11>, <count 11>,  <ID12>, <count 12>, ...
<ID2>, <count 2>, <ID21>, <count 21>,  <ID22>, <count 22>, ...
...
```

This notation has the following meaning:

`<IDa>` means the SUMO edge ID of edge a, where edge a is the edge entering a node.

`<IDab>` means the SUMO edge ID of edge ab, where edge ab is the edge outgoing from the node which edge a enters.

`<count a>` means the number of vehicles leaving edge a, in case the edge is generating flows, otherwise this count is zero.

`<count ab>` means the number of vehicles from edge a turning into edge b.

All counts in this file refer to the mode and time interval specified in the dialog box.

The turnflow file for the example network in Fig. 10, could look like this:

```
3, 1000
12, 800, 4, 200, 7, 600
8, 0, 9, 800, 2, 400
```

In this case, the generating flows $f_3 = 1000, f_{12} = 800$ and the turn flows $f_{8,9} = 800, f_{8,2} = 400$ and $f_{12,4} = 200, f_{12,7} = 500$ for a specific mode and time interval.

Hint: In order to quickly compile the turnflow file, open a text editor and the SUMOPy window next to each other. In the SUMOPy network editor, un-select lanes, connections and crossings by clicking on the button next to the zoom buttons below the network canvas. Then select the info tool (if not already active) and click on the edges you consider for the turnflow file. The respective edge will be highlighted and you can see all edge attributes in the object browser, as shown in Fig. 12. From

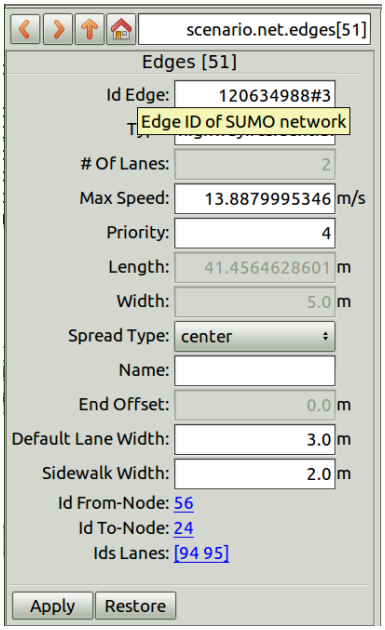

Figure 12: Object browser showing an edge. You can copy the SUMO Edge ID from the respective field.

there, copy the SOMO edge ID and paste it into your turnflow text file.

Defining destination zones: Optionally define destination zones. Within the turnflow demand model destination zones define edges where vehicle routes end. This may be necessary in order to prevent vehicles of making a loop on the network outside the study area and reentering the study area. Destination zones for turnflows are created in the same way as Traffic assignment Zones, see Fig. 8.

Routing flows and turn-flows: Generate directly route from flows and turn-flows information by selecting menu item

`Demand>Turnflows>Turnflows to routes`

This process is using SUMO's JTROUTER and will generate both trips and routes. The scenario is now ready to be simulated by selecting

`Simulation>Sumo>export routes and simulate...`

Proceed as described in Sec. 3.1.5.

A test file for turnflows is located in

`SUMOHOME/tools/contributed/sumopy/testscenario/demo_dem_tf_car`

3.5 Simulation processes

This section addresses some particular simulation issues.

3.5.1 Simulating Sublanes

With sublanes is new feature of SUMO since 2015. Sublanes renders traffic flows on roads more realistic. With sublanes, several vehicles can share side-by-side the same lane given there is enough room. For example a car can pass a bicycle on the same lane if the total lane width is larger than the bike width plus car width.

For sublane simulation with SUMOPy, simply browse to

`scenario.demand.vtypes`

and set the lanechange model model to `SL2015`[6]. Then run the simulation with

`Simulation>Sumo>export routes and simulate...`

Note that on the SUMO dialog, the sublane width is now positive (1m by default). This value (which can be changed) determines how many sublanes can stay within one lane. For example a 3m wide lane can have 3 sublane of 1m but only 2 sublane of width 1.5m.

The lanechange behavior with sublanes can be tweaked for each vehicle type with the sublane parameters in `scenario.demand.vtypes`.

References

[Krajzewicz (2003)] D. Krajzewicz, M. Hartinger, G. Hertkorn, P. Mieth, C. Rössel, P. Wagner, J. Ringel. The "Simulation of Urban MObility" package: An open source traffic simulation. In 2003 European Simulation and Modeling Conference (2003)

[6] SUMOPy applies the same lanechange model to all vehicle types