SUMO 快速入门

Author: Wang Maonan

The Chinese University of Hong Kong, Shenzhen
Shanghai AI Laboratory
maonanwang@link.cuhk.edu.cn

Table of Contents

What is SUMO

"Simulation of Urban MObility" (SUMO) is an open source, highly portable, *microscopic* and continuous traffic simulation package designed to handle large networks.[1]

It is mainly developed by employees of the Institute of Transportation Systems at the German Aerospace Center.

SUMO Homepage: `https://sumo.dlr.de/docs/index.html`.

Enjoy SUMO!

[1] 1, "Microscopic traffic simulation using sumo", 2018.

What You Will Learn in this Course

由于“SUMO” 的内容很多，我们这次会选我觉得最为基础和重要的内容。希望讲的这些内容都是干货，讲完之后大家可以能在实际中用到。

本次内容主要分成三个大部分，最后会有一个综合练习：

- “SUMO” 的基础使用（安装，文件介绍）；
- “SUMO” 的进阶使用（复杂路网和流量）；
- “Traci” 的使用（获得信息与控制环境）；

所有讲到的内容，包含PPT，路网文件，代码等，都会上传Github，Traffic Alpha。

欢迎大家一些来学习交流！

TRAFFIC-ALPHA

Install SUMO

“SUMO” 有两种安装的方式，分别是（1）直接使用安装包或是（2）通过python package进行安装。

- 如果使用第一种方式，点击下面链接SUMO Installing，接着选择对应的版本进行安装即可（安装完毕之后会有图标）。
- 我个人推荐第二种方式，可以直接使用下面的命令进行安装：

```
pip install eclipse-sumo
pip install traci libsumo
pip install sumolib
```

Install SUMO

安装完毕之后，在终端输出“sumo -V”，如果出现下图的界面，则安装成功。

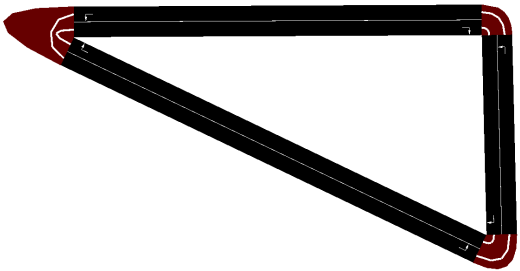
```
(traffic) WMN@WMN-PC:~$ sumo -V
Eclipse SUMO sumo Version 1.16.0
 Build features: Linux-5.15.0-1031-azure x86_64 GNU 10.2.1 Release FMI Proj GUI Intl SWIG GL2PS
 Copyright (C) 2001-2023 German Aerospace Center (DLR) and others; https://sumo.dlr.de

Eclipse SUMO sumo Version 1.16.0 is part of SUMO.
This program and the accompanying materials
are made available under the terms of the Eclipse Public License v2.0
which accompanies this distribution, and is available at
http://www.eclipse.org/legal/epl-v20.html
This program may also be made available under the following Secondary
Licenses when the conditions for such availability set forth in the Eclipse
Public License 2.0 are satisfied: GNU General Public License, version 2
or later which is available at
https://www.gnu.org/licenses/old-licenses/gpl-2.0-standalone.html
```

Netedit

使用Netedit 可以用来编辑路网文件。在终端中输入“netedit” 打开路网编辑器。

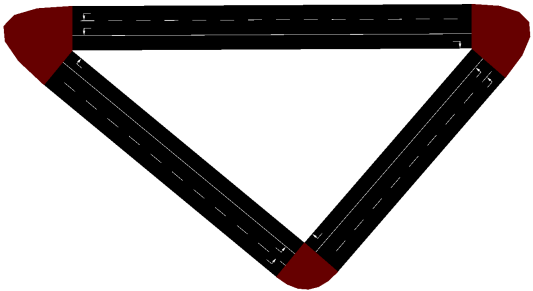
- 点击“Set Create Edge Mode” 创建节点；
- 接着创建三个节点，节点之间两两相连；
- 保存为“.net.xml” 文件；



Netedit

如果想要继续编辑某个路网，可以使用“netedit xxx.net.xml” 进行编辑。

- 查看node，edge 等的信息；
- 修改edge 信息（长度，车道数，限速，限行）；
- 修改节点位置（move mode）；



Route File

现在我们已经有了一个network 文件。这个network 文件相当于是路的基础结构，现在我们需要定义一些route 的信息。route 文件中主要包括车辆的信息，和车辆是如何行驶的（具体的路径）。

在SUMO中，一辆车有一些基本的属性，例如长度（length），加速与减速（acceleration and deceleration），最大速度（maximum speed）。

关于SUMO 中车辆属性的设置，可以查看链接Definition of Vehicles, Vehicle Types, and Routes。

Route File

我们新建一个“sumo.rou.xml” 的文件。里面有一辆车的信息，和两辆车行驶路径的信息。

```
<routes>
    <vType accel="1.0" decel="1.0" id="Car1" length="5.0"
       maxSpeed="10.0" sigma="0.0" />
    <route id="route1" edges="E4 E5 E6"/>
    <route id="route2" edges="-E6 -E5 -E4"/>
    <vehicle depart="1" id="veh0" route="route1" type="Car1" />
    <vehicle depart="2" id="veh1" route="route2" type="Car1" />
</routes>
```

SUMO Configuration

最后我们配置configure 文件。这个文件主要用来指定network 文件，route 文件，additional 文件（例如探测器等），仿真开始和结束的时间等。

```
<configuration>
    <input>
        <net-file value="sumo.net.xml"/>
        <route-files value="sumo.rou.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <end value="50"/>
    </time>
</configuration>
```

TRAFFIC-ALPHA

SUMO GUI

在定义好配置文件之后，就可以进行仿真。在终端输入：

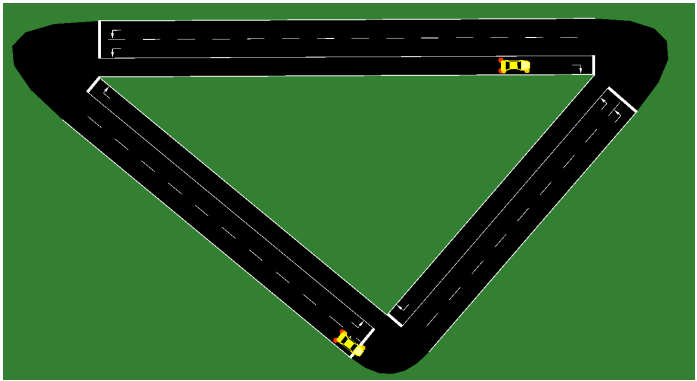
```
sumo-gui -c sumo.sumocfg # with gui
sumo -c sumo.sumocfg # without gui
```

其实我们可以不写配置文件，直接在命令行的参数中指定路网文件和车流文件：

```
sumo-gui -n sumo.net.xml -r sumo.rou.xml
sumo -n sumo.net.xml -r sumo.rou.xml
```

SUMO GUI

运行上面的命令，则会出现如下的界面。可以选择不同的可视化风格：



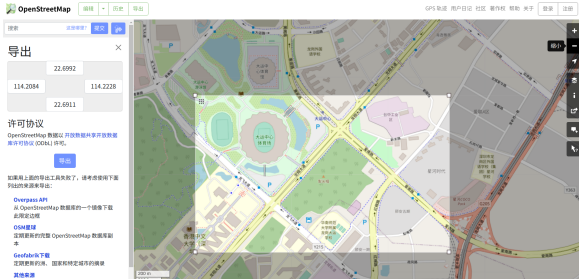
SUMO Basic Usage Summary

- SUMO 的安装（Python Package）；
- SUMO 仿真的三个核心文件
  - **.net.xml**，路网文件。包含路网的拓扑结构，道路的信息，信号灯的信息。可以使用netedit 进行编辑；
  - **.rou.xml**，车辆文件。包含车辆的信息，车辆出发时间，车辆的路径；
  - **.sumocfg**，配置文件。包含仿真中用到的所有信息。也可以通命令行参数的方式进行指定，例如“sumo -n x.net.xml -r x.rou.xml”；

Open Street Map

上面介绍了从头编辑路网，我们还可以直接对真实的路网进行编辑，生成需要的“.net.xml”。我们打开OpenStreetMap。

接着选中自己想要的区域，这里我们选择学校附近的道路，也就是“龙翔大道”，框选指定区域后点击导出，导出的格式为“.osm”：



OSM to SUMO Net

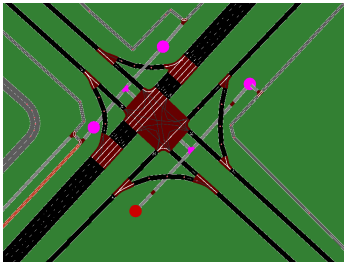
“SUMO” 提供了“netconvert” 的工具来将“OSM” 的路网转换为“SUMO Net”。

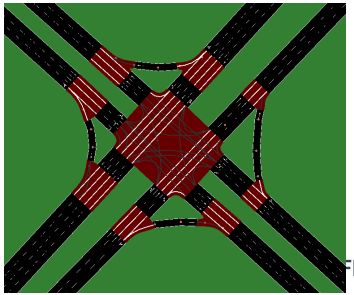
```
1 netconvert --osm map.osm --junctions.join True -o sumo.net.xml
```

Edit SUMO Net

直接转换得到的路网通常包含很多不需要的元素，因此我们需要手动对其进行修改。

- 删除多余的道路（人行道，铁轨，地铁等）；
- 修改车道数（可以去实地，或是通过卫星图片进行校对）；





Generate Route based on Flow

前面介绍车辆生成的时候，需要指定每一辆车的路径，这样是很麻烦的。“SUMO” 也支持直接定义起点和终点，和这个时间段内的车辆。

```
<routes>
    <interval begin="0" end="100">
        <flow id="00" from="a1" to="b1" number="50"/>
        <flow id="01" from="a2" to="b2" number="50"/>
    </interval>

    <interval begin="50" end="150">
        <flow id="11" from="a3" to="b3" number="50"/>
        <flow id="12" from="a4" to="b4" number="50"/>
    </interval>
</routes>
```

Flow Convert to SUMO Route

接着使用“duarouter” 进行转换，将上面创建的“.flow.xml” 文件转换为“.rou.xml” 文件。

```
duarouter --route-files=sumo.flow.xml --net-file=sumo.net.xml --
    output-file=sumo.rou.xml --human-readable-time --randomize-
    flows true --departlane random --seed 30
```

- –human-readable-time，将route 中的时间转换为day:hour:minute:second 的格式，而不是只有second；
- –randomize-flows，将车辆depart time 随机，而不是均匀出现；
- –departlane，车辆起始的车道，我们选择random；

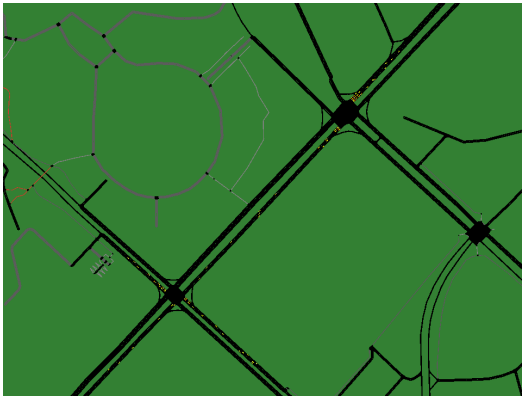
Generate Route based on Flow

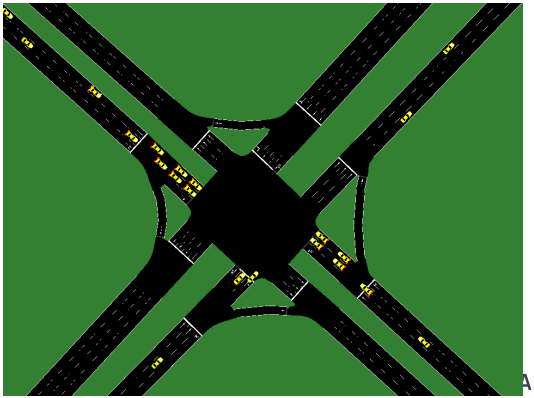
最后同样是定义“sumo.sumocfg”，指定路网和车流文件，和仿真开始和结束的时间即可（同样我们也可以直接指定而不需要定义配置文件）。

```
<configuration>
    <input>
        <net-file value="sumo.net.xml"/>
        <route-files value="sumo.rou.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <end value="200"/>
    </time>
</configuration>
```

Simulation on Real Network

最后就可以利用“sumo-gui” 命令进行仿真：





Introduction to Traci

TraCI is the short term for "**T**raffic **C**ontrol **I**nterface". Giving access to a running road traffic simulation, it allows to *retrieve values* of simulated objects and to *manipulate their behavior* "on-line".

```
1  import traci
2  import sumolib
3
4  sumoBinary = sumolib.checkBinary('sumo-gui')
5  traci.start(
6      [sumoBinary,
7       "-c", './sumo_net/sumo.sumocfg'
8      ],
9      label='0'
10 )
```

Traci Simulation Step

前面我们使用“Traci” 开启了仿真。接着使用“traci.simulationStep()” 来进行仿真。

在上面的文件中添加下面的内容，即可进行仿真，直到仿真中没有车辆就停止。

```
1 # Stop until no vehicles in net
2 while traci.simulation.getMinExpectedNumber() > 0:
3     traci.simulationStep()
4
5 traci.close()
```

Retrieval Vehicle Info

我们可以获得仿真中所有车辆的信息。例如下面的例子中，我们获得了车辆的（1）位置，（2）速度，（3）等待时间，（4）所在的edge：

```
while traci.simulation.getMinExpectedNumber() > 0:
    time = traci.simulation.getTime()
    for vehID in traci.vehicle.getIDList():
        pos = traci.vehicle.getPosition(vehID)
        speed = traci.vehicle.getSpeed(vehID)
        time_loss = traci.vehicle.getWaitingTime(vehID)
        print(time, vehID, pos, speed, time_loss)
    print('-')
    traci.simulationStep()
```

Retrieval Lane Info

上面是利用“Traci” 获得车辆的信息，当然我们也可以获得车道的信息。在下面的例子中，我们获得（1）平均车速，（2）碳排放：

```
while traci.simulation.getMinExpectedNumber() > 0:
    for edge_id in traci.edge.getIDList():
        veh_num = traci.edge.getLastStepVehicleNumber(edge_id)
        if veh_num > 0:
            avg_speed = traci.edge.getLastStepMeanSpeed(edge_id)
            avg_co2 = traci.edge.getCO2Emission(edge_id)
            print(edge_id, avg_speed, avg_co2)
    print('-')
    traci.simulationStep()
```

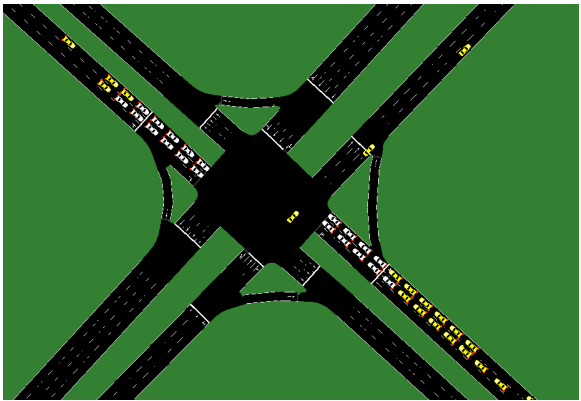
Change Vehicle Color

上面是利用“Traci” 获得信息，这里介绍利用“Traci” 对信息进行修改。在下面的例子中，我们修改车辆颜色，修改的条件为：当车辆等待时间超过10s 则修改颜色：

```
while traci.simulation.getMinExpectedNumber() > 0:
    time = traci.simulation.getTime()
    for vehID in traci.vehicle.getIDList():
        time_loss = traci.vehicle.getWaitingTime(vehID)
        if time_loss > 10:
            traci.vehicle.setColor(vehID, (255,255,255))
    traci.simulationStep()
```

Change Vehicle Color

我们运行上面的代码结果如下图所示，可以看到当车辆等到时间超过10s 后，颜色变为白色（除了颜色外，我们还可以修改速度，路径等，去适应不同的场景）。



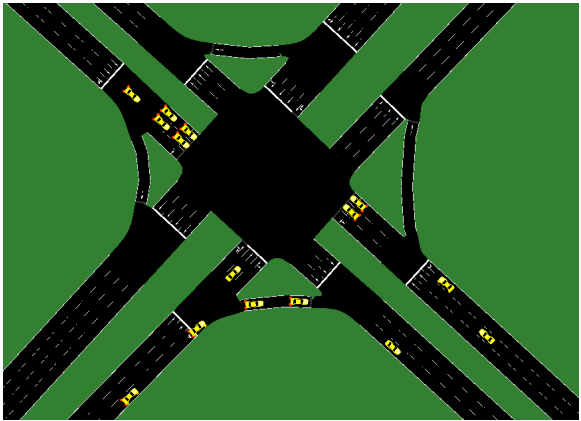
Add New Vehicle

除了修改车辆属性之外，我们还可以在仿真过程中插入额外的车辆。下面的例子中，我们插入了右转的车辆：

```
1 traci.route.add('r_turn', ['1125695391#0', '238823566', '
    219060439#2'])
2 for i in range(10):
3     traci.vehicle.add(str(i), 'r_turn', depart=i+20, departLane='
        free')
4
5 while traci.simulation.getMinExpectedNumber() > 0:
6     traci.simulationStep()
```

Add New Vehicle

我们运行上面的代码结果如下图所示，可以看到此时有车辆进行了右转。



A Toy Example

在前面的仿真中，我们发现由于路口没有信号灯，故东西方向的车辆会一直等待，直到南北方向没有车辆才会通行。

这里我们考虑对其设置一个策略，使得不会有一侧的车辆等到时间过长。策略如下：

- 每个时刻检查每个edge 的等待时间；
- 将等待时间大于平均等到时间的edge 上面车速加快；
- 将小于平均时长的edge 上面的车速变慢；

A Toy Example – Get Edge Avg Time Loss

于是根据上面的思路，我们首先获得每一个edge 上面的平均等待时间（需要注意这里车道有两段，为了方便我们取接近路口的一段）。

```
edge_ids = {
    '1125695390#0':0, '1125695391#1':0,
    '1125691753#2':0, '1125684496#2':0
}

while traci.simulation.getMinExpectedNumber() > 0:
    # Update Avg Time Loss for each Edge
    for edge_id in edge_ids.keys():
        veh_nums = traci.edge.getLastStepVehicleNumber(edge_id)
        time_loss = traci.edge.getWaitingTime(edge_id)
        avg_time_loss = time_loss/veh_nums if veh_nums!=0 else 0
        edge_ids[edge_id] = avg_time_loss
```

A Toy Example – Set Vehicle Speed

接下来我们写出对应的策略（有两段道路）。

```
avg_edge_time_loss = np.mean(list(edge_ids.values()))
for edge_id, avg_time_loss in edge_ids.items():
    for veh_id in traci.edge.getLastStepVehicleIDs(edge_id):
        traci.vehicle.setSpeed(veh_id, 14)
        traci.vehicle.setSpeedMode(veh_id, 5)

    if avg_time_loss >= avg_edge_time_loss:
        for veh_id in traci.edge.getLastStepVehicleIDs(
            upstream_edge[edge_id]):
            traci.vehicle.setSpeed(veh_id, 14)
    else:
        ...
```

Change Vehicle Color

最终的效果如下所示。注意我们这里的策略十分简单，例如没有考虑哪些方向一起通行是冲突的，这里只是作为演示。

Conclusion

- SUMO 仿真文件（.net.xml，.rou.xml，.sumocfg）；
  - “.net.xml” 记录路网的拓扑信息（车道数，长度，道路类型）；
  - “.rou.xml” 记录车辆的信息（出发时间，路径，路径，车辆类型）；
- Open Street Map 导出真实路网，使用Flow 生成流量；
  - 利用“netconvert” 将OSM 文件转换为SUMO Net；
  - 利用“duarouter” 将flow 转换为SUMO Route；
- Traci 获取和改变环境信息：
  - “traci.xxx.getxxx()” 获取指定的信息；
  - “traci.xxx.setxxx()” 设置指定的信息；

Q&A

Questions?

References I

[1] Pablo Alvarez Lopez et al. “Microscopic traffic simulation using sumo”. In: *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE. 2018, pp. 2575–2582.