SUMO, SUMO-tools e TraCI

Alessandro Cacco (ale.cacco@gmail.com / alessandro.cacco@studenti.unitn.it)

October 16, 2020

1. File di Esempio

L'archivio contiene vari file:

- Scenario: `test.net.xml`
- Pedoni: `pedestrians*.rou.xml`
- Traffico: `cars*.rou.xml`
  - N.B. i file di traffic con *"_ mod"* nel nome hanno i veicoli settati con tipo *"special"*, con la definizione che va come primo sub-item di `<routes>`. Li ho editati manualmente, ma si possono impostare in automatico durante la generazione del traffico randomico.

2. SUMO e tool annessi

Per l'installazione, basta seguire la guida a `https://sumo.dlr.de/docs/Installing.html`, in linux basta usare apt:

```
sudo add-apt-repository ppa:sumo/stable
sudo apt-get update
sudo apt-get sumo sumo-tools
```

Importante è da settare la variabile d'ambiente `SUMO_HOME`, che in caso di installazione con apt nel mio caso andava puntata a `/usr/share/sumo`. In caso non venga settata, il simulatore cerca online *ogni singola* definizione xml, diventando molto lento all'avvio della simulazione.

Tool utili:

- `sumo`/`sumo-gui`: simulatore CLI/GUI
- `netconvert`: convertitore formato scenario
- `netedit`: editor per creare/modificare scenari.
- `randomtrips.py`: script generatore di traffico.

Per lanciare SUMO da CLI, questi sono i parametri base:

- `-n <SCENARIO>`: lo scenario (network) da caricare
- `-r <ROUTEFILE1>,<ROUTEFILE2>,...`: carica il traffico (routes), separato da virgole in caso di file diversi. Nota che i pedoni sono considerati traffico a tutti gli effetti.
- `-start`: fa iniziare in automatico la simulazione con `sumo-gui` senza dover premere il tasto play
- `-Q`: fa chiudere automaticamente il programma alla fine della simulazione, o alla disconnesione di TraCI

- `-remote-port <PORT>`: lancia sumo in modalità server, in modo che resti in attesa di una connessione via TraCI (Traffic Control Interface)
- `-pedestrian.striping.dawdling <VALUE>`: valore per il modello dei pedoni, più informazioni in `https://sumo.dlr.de/docs/Simulation/Pedestrians.html`

3. Conversione degli scenari

Usando il tool netconvert si possono convertire scenari OSM nel formato XML di SUMO. La conversione è imperfetta, ci sono una serie di problemi con le junction (ovvero gli incroci). `https://sumo.dlr.de/docs/Networks/Import/OpenStreetMap.html` contiene alcune spiegazioni di come configurare la conversione, anche se in realtà dipende molto dalle caratteristiche della specifica mappa. In linea di massima:

```
netconvert --osm-files <MAPPA>.osm.xml -o <MAPPA>.net.xml
```

Interessante per poter editare facilmente lo scenario da script, il file `.net.xml` si può convertire in "plain XML files" in maniera lossless. Questi file sono molto più comprensibili per fare modifiche sullo scenario, è *caldamente* consiglato modificare questi e non il `.net.xml`. Più info a `https://sumo.dlr.de/docs/Networks/PlainXML.html`.

4. Traffico stradale e pedonale

Per generare traffico su una data mappa c'è un tool incluso, `DUAROUTER`, ma per facilitarne l'utilizzo c'è lo script `randomtrips.py` (`https://sumo.dlr.de/docs/Tools/Trip.html`, i parametri principali sono:

- `-p <VALUE>`: "repetition rate" dei viaggi generati, ovvero ogni quanti secondi parte un'auto. Si può settare a un valore inferiore a 1 per ottenere più auto.
- `-o <FILE>`: salva le route generate in questo file. ATTENZIONE a lanciare in parallelo più generazioni di traffico randomico, viene creato un file temporaneo con lo stesso nome di default, c'è un altro parametro per cambiarlo.
- `-persontrips`: genera traffico pedonale, quindi gestisce in automatico le restrizioni del tipo di veicolo per i pedoni.
- `-additional-file <FILE>`: carica info varie addizionali, tra cui i vehicletype se si vogliono settare in automatico. Più info qui `https://sumo.dlr.de/docs/sumo.html#format_of_additional_files`.
- `-vehicle-class <VEHICLE-TYPE>`: setta in automatico il tipo dei veicolo alle routes. Il modello va dichiarato in un "additionals" file.

5. TraCI

Per usare TraCI da python, le librerie nel mio caso sono state installate da apt in `/usr/share/sumo/tools`, quindi basta aggiungere questa dir alla variabile `$PYTHONPATH`. Un esempio di utilizzo è il seguente, dopo aver lanciato una simulazione in modalità "server":

```
import traci
t = traci.connect(<PORT>) #porta della simulazione
t.simulationStep() #per avanzare la simulazione di un tick
t.simulation.getCollidingVehiclesNumber()    #per ottenere il numero di incidenti
                                             #nel tick corrente
t.simulation.getCollidingVehiclesNumber()    #ritorna la lista dei veicoli
```

```
                                        #incidentati nel tick corrente
t.close() # per chiudere la connessione
          # con sumo-gui aggiungere -Q per farlo chiudere automaticamente
```

ATTENZIONE: in caso di scenari con molti veicoli in circolazione le richieste traci rallentano notevolmente la simulazione (`https://sumo.dlr.de/docs/TraCI.html#performance`).

6. Informazioni utili

Ridurre la prudenza agli incroci: `https://sumo.dlr.de/docs/Definition_of_Vehicles,_Vehicle_Types,_and_Routes.html#junction_model_parameters`
Causare incidenti:
. `https://sumo.dlr.de/docs/Simulation/Safety.html#deliberately_causing_collisions`
. `https://sumo.dlr.de/docs/Driver_State.html`
MODELLI di guida: `https://sumo.dlr.de/docs/Definition_of_Vehicles,_Vehicle_Types,_and_Routes.html`
Documentazione brutta di TraCI:`https://sumo.dlr.de/daily/pydoc`
Creare e modificare scenari: `https://sumo.dlr.de/docs/netedit.html`