Report

Github address:

# 1. Datasets

1. Public data benchmarks

We are using RESCO dataset for evaluation and experiments,

Paper: <https://people.engr.tamu.edu/guni/Papers/NeurIPS-signals.pdf>

Github: <https://github.com/LucasAlegre/sumo-rl>

<https://github.com/LucasAlegre/sumo-rl/blob/master/nets/RESCO/maps.png>

Check folder nets/RESCO. It contains real data with different complexities:

Diagram

Description automatically generated

1. Bucharest data and optimization for a real case

* Using OSMWizard and SUMO we can capture real data from Bucharest. Check folder MASSA\_Bucuresti for a few examples(see the other document in this folder to see how I did this).
* Problem: we don’t have real traffic data. We tried to take from cameras, it is possible but only 2-3 intersections cameras are working …
* How to real data:
  + O/D matrices into SUMO

<https://sumo.dlr.de/docs/Demand/Importing_O/D_Matrices.html#describing_the_taz>

* + Import Induction loop data <https://sumo.dlr.de/docs/TraCI/Induction_Loop_Value_Retrieval.html>
  + Bolt data (as promised): We can map the traffic O/D, traffic demand on lanes and intersections.

1. See the other research doc in this folder for other methods to get real kind of data from Paris and other locations in the world.

# 2. Current RL method implemented

## A. MDP definition

**Observation**

The default observation for each traffic signal agent is a vector:

**obs = [phase\_one\_hot, min\_green, lane\_1\_density,...,lane\_n\_density, lane\_1\_queue,...,lane\_n\_queue]**

**phase\_one\_hot** : one-hot encoded vector indicating the current active green phase

**min\_green** is a binary variable indicating whether min\_green seconds have already passed in the current phase

**lane\_i\_density** is the number of vehicles in incoming lane i dividided by the total capacity of the lane

**lane\_i\_queue** the number of queued (speed below 0.1 m/s) vehicles in incoming lane i divided by the total capacity of the lane

We can define own observation changing the method 'compute\_observation' of TrafficSignal.

**Actions**

The action space is discrete. Every 'delta\_time' seconds, each traffic signal agent can choose the next green phase configuration. Every time a phase change occurs, the next phase is preeceded by a yellow phase lasting yellow\_time seconds. E.g.: In the 2-way single intersection there are |A| = 4 discrete actions, corresponding to the following green phase configurations:

A screenshot of a computer

Description automatically generated with low confidence

**Rewards**

How much the total delay (sum of the waiting times of all approaching vehicles) changed in relation to the previous time-step.

A picture containing text, clock

Description automatically generated

We can customize reward function within method 'compute\_reward' of TrafficSignal.

## B. Software stack

A picture containing diagram

Description automatically generated

* Single agent means that there will be a single agent controlling all the traffic lights in the environment/ In this case we use Baselines3 to leverage existing RL core implementation. In this moment we use DQN only with a simple MLP to map states to a latent.
* Multi-agent means that there will be an individual agent assigned to each traffic light. In this case, we use RLLib for massive parallelization on clusters, processors, and GPUs.

This case also needs pre-processing operations such as normalizing the length of actions/observations, etc. We use *supersuit* package for this. As algorithm, now A3C is used.

## C. Experiments and Code discussion

* The code for both single and multi-agent is in experiments\massa\_\*.py
* To output the plots and get mean basic results according to a series of run, use plot.py.
* You can find for all these experiments the attached PyCharm configuration so if you install it, it is easy to understand the arguments and running setup.
* Just a result sample - total waiting time in simulation time range :
  + Single agent (DQN):

Chart

Description automatically generated

* + Multi agent (A3C):
  + Chart, histogram

    Description automatically generated

## Technical TODOs:

* 1. Fix the parallel updates. Currently in our repository it works only on sequential updates Check they work on another repository example: <https://github.com/Farama-Foundation/PettingZoo/blob/master/tutorials/rllib_pistonball.py>
  2. Check how other algorithms work, DDPG ? Seems like policy gradient methods are best

# 3. Future work and publishing

A. Beat state of the art multi-agent using different observations, rewards, deep architectures (see the other doc).

B. Use data from Bolt in Bucharest to publish how car passengers can collaborate together to improve real time traffic signals automatically, with an IoT based infrastructure in the city. Also Create a reusable tool to automatically extract map’s locations from OSM, link to a datasource and produce datasets.

C. Compare and review state of the art work and algorithms on real data from RESCO, Bucharest, simulations.

D. Niche apps: optimize traffic using RESCO/Bolt by: smart pedestrian crossing, cut-off restrictions for bicycle lanes or bus temporarily, speed restrictions, etc.

Some ideas: <https://flow-project.github.io/publications.html>