Contents

List of Figures

1 Introduction

Ramp metering uses traffic signals to regulate vehicle flows from on-ramps to the mainline of the freeway. It alleviates the negative impacts of “capacity drop” resulting from massive merging behaviors and reduces the total time spent in the traffic system. Several field tests have demonstrated the effectiveness of ramp metering in terms of throughput, vehicle miles-traveled, vehicle-hours-traveled, and travel time reliability.

This project focuses on the application of Q-learning and Deep Q-learning algorithms to optimize ramp metering control on a highway through numerical simulations. The study considers a stretch of highway with variable parameters, such as the number of lanes and controlled entrance ramps equipped with traffic lights. The Simulation of Urban Mobility (SUMO) is utilized to simulate car-following and lane changes.

2 Objective

The primary goal of this project is to employ reinforcement learning algorithms, specifically Q-learning and Deep Q-learning, to enhance the control of ramp metering on highways. Through numerical simulations using SUMO, the team aims to develop an algorithm that optimizes the flow of traffic on both the highway stretch and the entering ramp.

3 Development tools

For the successful implementation of the project, a variety of development tools and libraries were employed. Each tool played a crucial role in different aspects of the project, ranging from traffic simulation to deep reinforcement learning.

3.1 Python

Description: Python served as the primary programming language for the project. Its versatility, ease of use, and extensive ecosystem of libraries made it an ideal choice for implementing the reinforcement learning algorithms, interfacing with SUMO, and handling various tasks within the project.

Purpose: General-purpose programming, algorithm implementation, and script execution.

3.2 SUMO (Simulation of Urban Mobility)

Description: SUMO is an open-source traffic simulation suite that provided a realistic environment for modeling car-following behavior, lane changes, and traffic controls. It played a central role in simulating highway traffic scenarios.
Purpose: Traffic simulation, modeling, and visualization.

3.3 TensorFlow and Keras

Description: TensorFlow, an open-source machine learning library, along with its high-level API Keras, were used for implementing and training deep reinforcement learning models. These tools streamlined the creation, training, and evaluation of neural networks.
Purpose: Deep reinforcement learning, neural network modeling, and training.

3.4 NumPy and Collections

Description: NumPy, a powerful numerical library, and the Collections module provided essential data manipulation and storage capabilities. These tools facilitated efficient handling of simulation data, particularly arrays, matrices, and specialized data structures.
Purpose: Numerical operations, data manipulation, and storage.

3.5 Traffic Control Interface (traci)

Description: The traci library acted as the interface between Python scripts and the SUMO traffic simulator. It enabled real-time control and monitoring of the simulation, allowing the reinforcement learning agent to interact with the simulated environment.
Purpose: Real-time simulation control and monitoring.

3.6 plot_model and load_model (Keras)

Description: plot_model was utilized for visualizing the architecture of the neural network models created using Keras. load_model facilitated the loading of pre-trained models, enhancing flexibility and reusability.
Purpose: Model visualization and pre-trained model loading.

3.7 sumolib

Description: The sumolib library, part of the SUMO simulation tools, provided essential functionalities. The *checkBinary* function from sumolib played a key role in checking the binary executables required for SUMO.
Purpose: SUMO-related binary checking.

4 Methodology

4.1 Environment setup

Configuration of the SUMO environment, defining the highway network, entrance ramps, and traffic light controls as illustrated in Fig.1

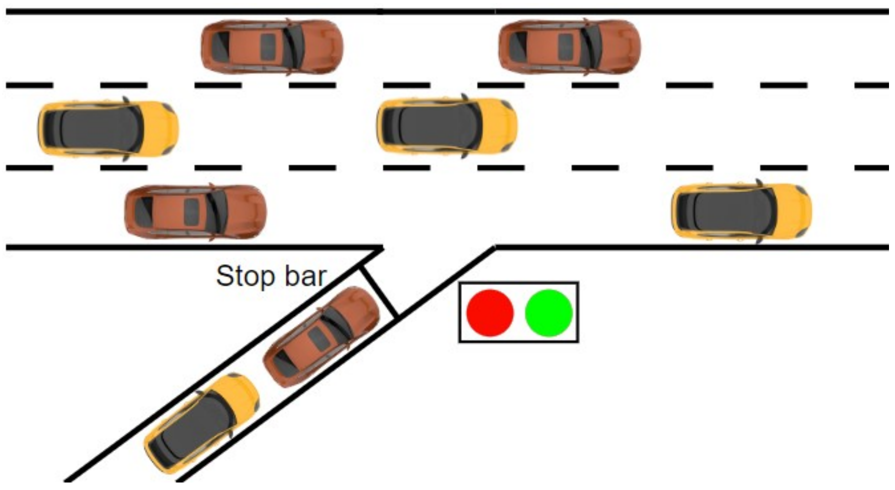


Figure 1: Two-phase control schema for ramp metering

Fig.2 illustrates the ramp metering problem in the general scheme of RL. The ramp meter acts as the agent to interact with the environment, namely the traffic system including the mainline and the on-ramp. The ramp meter takes certain control action a based on current traffic state s. Then the traffic system responds to the control action with the state transition from s to s'. The ramp meter obtains reward r that quantifies the effect of the control action a given state s. In the RL field, the interaction process is assumed to be a Markov Decision Process. That is, given current state s and action a, the following state s' and reward r are independent of previous states and actions.

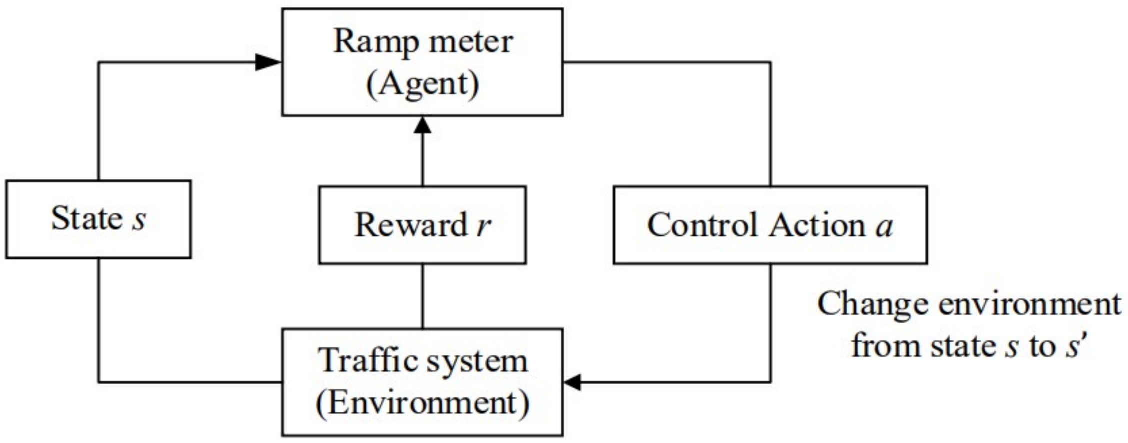


Figure 2: Ramp metering in the general schema of Reinforcement Learning

4.2 Algorithm implementation

In this section, we delve into the heart of the project, where the Q-learning and Deep Q-learning algorithms are seamlessly integrated into the simulation code. The objective is to dynamically control the traffic light at the entrance ramp, optimizing traffic conditions on both the highway stretch and the ramp. The algorithms play a pivotal role in enabling the traffic signal to adapt and learn from the simulated environment, iteratively refining its decision-making process.

4.2.1 sumoClass.py

The *sumoClass.py* file encapsulates essential functionalities for interfacing with the SUMO traffic simulator. This Python module serves as a bridge between the reinforcement learning algorithms and the simulation environment.

1. **Initialization (_init_method)**

   - The constructor initializes the simulation. It prints a message indicating that the simulation is being prepared.

2. **get_options method**

   - This method takes three parameters:

     - gui (a boolean indicating whether to use the SUMO GUI).
     - sumocfg_file_name (the SUMO configuration file name).
     - max_steps (maximum simulation steps).

- It checks if the 'SUMO_HOME' environment variable is set. If not, it exits with an error message.
- It sets the SUMO binary (command line or GUI) based on the gui parameter.
- It constructs and returns a list (sumo_cmd) representing the command to run SUMO with the specified configuration file and options.

3. **generate_routefile method**

   - This method generates a route file (route.rou.xml) for the cars in the simulation.
   - It uses a Weibull distribution to determine the timings for car generation.
   - Cars are generated based on random choices for straight or turn directions.
   - The generated routes and vehicle information are written to the route file.

4. **get_state method**

   - This method retrieves the current state of the simulation.
   - It initializes a state array with zeros and retrieves the list of active vehicles in the simulation.
   - For each vehicle, it calculates its position in terms of cells based on its lane position and ID.
   - The cell positions are specific to the simulation scenario and are used to represent the state of the intersection.
   - The method returns **the state** array, indicating the occupancy of cells by vehicles.

4.2.2 my_model.py

The *my_model.py* file represents the neural network model used for implementing the Q-learning and Deep Q-learning algorithms. This Python module leverages the TensorFlow and Keras libraries to construct a deep neural network that serves as the brain of our reinforcement learning agent.
This file has a class called TLSClass with the following components :

1. **Initialization (_init_method)**

- Initializes various parameters for the DQN, such as gamma (discount factor), learning rate, memory (replay memory for experience replay), action size, input and output dimensions, and the neural network model.
- The neural network model is built using the _build_model method.

2. **_build_model method**

   - Builds and compiles a fully connected deep neural network using the Keras library.
   - The network has a specified number of layers (num_layers) and width (width).
   - The output layer has a linear activation function, suitable for regression problems.
   - The model is compiled using mean squared error loss and the Adam optimizer.

3. **remember method**

   - Stores a sample (experience tuple) in the replay memory.
   - If the length of the memory exceeds a maximum size (_size_max), the oldest element is removed.

4. **get_samples method**

   - Retrieves a specified number (n) of samples randomly from the memory.
   - If the memory size is below a minimum size (_size_min), an empty list is returned.

5. **act method**

   - Chooses an action based on the epsilon-greedy policy.
   - With probability epsilon, a random action is chosen; otherwise, the action with the highest predicted Q-value is selected.

6. **load and save methods**

   - Load and save the weights of the neural network.

7. **_collect_waiting_times method**

   - Retrieves the waiting time of every car in the incoming roads.
   - It considers only the waiting times of cars in specified incoming roads ("E8", "E9").

8. **_simulate method**

   - Executes a specified number of simulation steps in SUMO while gathering statistics.
   - It updates the step counter, queue length, and waiting time during the simulation.

9. **_replay method**

   - Retrieves a group of samples from the memory and updates the learning equation for each of them.
   - It trains the neural network by fitting the Q-values using the Q-learning update rule.

These two modules, *sumoClass.py* and *my_ model.py*, collectively form the backbone of our algorithmic implementation. Their collaborative functionality facilitates a symbiotic interaction between the simulated traffic environment and the reinforcement learning agent, ultimately leading to an intelligent and adaptive traffic control system on the highway.

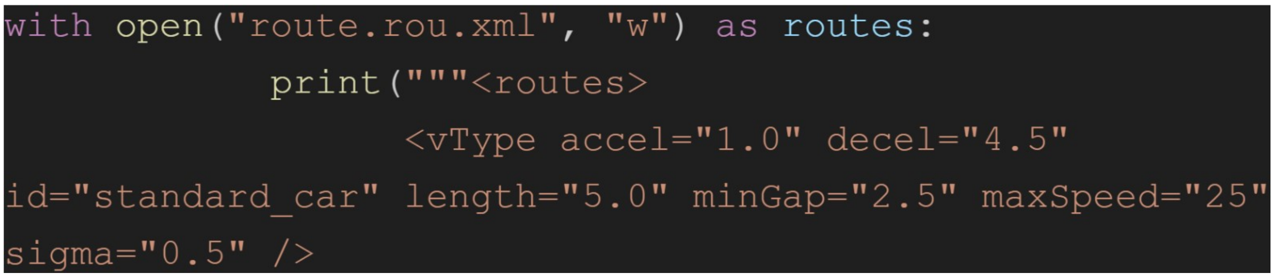
4.3 Scenario Variation

Implementation of scenarios to vary parameters, covering a wide range of situations:

- Length of the highway stretch : can be changed in the **TLSClass**

  self.model = self._build_model(6,400) # 400 is the length of the highway

- Speed limits on the highway and ramp : is defined when creating route.rou.xml file. the max speed limit can be changed from network.net.xml in this portion of code :



```
with open("route.rou.xml", "w") as routes:
            print("""<routes>
                  <vType accel="1.0" decel="4.5"
id="standard_car" length="5.0" minGap="2.5" maxSpeed="25"
sigma="0.5" />
```

Figure 3: network.net.xml's portion of code

- Initial car density on the highway and ramp : car density was defined by the weibull distribution.

- Car flow (vehicles per hour) on the highway and inflow from the entrance ramp : he generation of vehicles is controlled by the generate_routefile method, specifically by the use of a Weibull distribution to determine the timing of car arrivals. This timing information is then used to generate vehicles on either the "E8" or "E9" routes. The key variable related to car flow in this code is the timings array, which is generated as follows: timings = np.random.weibull(4, 1800)
  Here, *timings* is an array of 1800 values sampled from a Weibull distribution with a shape parameter of 4. These values represent the timing or arrival times of cars in the simulation.

- Number of lanes, and more.

5 Challenges and Considerations

The project team acknowledges potential challenges in fine-tuning the reinforcement learning model and ensuring its adaptability to diverse scenarios. Adjustments may be necessary to address unforeseen issues, and careful consideration will be given to the interpretability of the model's decisions.

6 Conclusion

This project represents a significant step towards harnessing reinforcement learning for efficient ramp metering on highways. Through the application of Q-learning and Deep Q-learning algorithms in a simulated environment, the team aims to contribute valuable insights to the field of traffic optimization and management.

7 Recommendations

Future work may involve exploring more advanced RL techniques, incorporating additional parameters into the state space, and considering real-world implementations. Collaboration with traffic engineering experts and urban planners could enhance the practical applicability of the developed model.

8 Bibliography

- https://github.com/romainducrocq/DQN-ITSCwPD/

- https://arxiv.org/pdf/2109.14337.pdf

- https://github.com/AndreaVidali/Deep-QLearning-Agent-for-Traffic-Signal-Control

- https://arxiv.org/pdf/2012.12104.pdf