Traffic Control with Reinforcement Learning

Application of reinforcement learning to the management of a traffic light intersection

23/02/2024

by

Andrea Falbo
a.falbo7@campus.unimib.it

Ruben Tenderini
r.tenderini@campus.unimib.it

Under the supervision of
Dr. Giuseppe Vizzari

University of Milano-Bicocca

Contents

1 Introduction

1.1 Definition

This study aims to compare the effectiveness of different reinforcement learning strategies for managing the traffic of a signalized intersection, identified as BI (Big Intersection). Specifically, four different reinforcement learning techniques for traffic light management will be examined with fiexd cycle intersection:

1. **Fixed Cycle:** The traffic light phases follow a fixed cycle, repeating in a predictable manner.
2. **Q-Learning:** The operation of the traffic light is entrusted to an agent trained using the Q-Learning method.
3. **Deep Q-Learning:** The operation of the traffic light is entrusted to an agent trained using the Deep Q-Learning method.
4. **Sarsa Learning:** The operation of the traffic light is entrusted to an agent trained using the Sarsa Learning method.
5. **Sarsa Decay Learning:** The operation of the traffic light is entrusted to an agent trained using the Sarsa Learning method, which implements a decay in the exploration factor.

Each algorithm will be experimented with through 2 different configurations, varying the hyperparameters in order to explore the sensitivity of the algorithms and their adaptability to variable contexts, contributing to identifying optimal parameters and improving the generalization of performance in traffic control.

The training of the models will consider two different traffic situations: low traffic and high traffic. This choice is aimed at evaluating the models' ability to adapt to various traffic dynamics and to generalize effectively.

The trained models will subsequently be evaluated by running them both in the same traffic situation in which they were trained and in the previously unseen situation. This methodology aims to test the models' generalization ability, avoiding overfitting.

1.2 Objective

The primary objective of this study is to highlight whether a reinforcement learning approach can provide better performance than a traditional traffic light cycle. To do this, optimal configurations for each algorithm will be sought, and then the performance of these will be compared through different metrics and traffic situations.

2 Reinforcement Learning

2.1 Probabilistic Planning

In the context of Reinforcement Learning, the agent often faces the challenge of operating in environments where complete knowledge of the state is impractical. This problem is effectively addressed through Probabilistic Planning, a methodology that incorporates uncertainty into the agent's learning.

The heart of this approach lies in probability theory, which allows the agent to maintain probabilistic beliefs, called belief. These are dynamic and vary in response to new information obtained through sensors.

A key concept in this context is Bayes' Theorem, used to update beliefs based on sensory observations. This formula allows the agent to recalibrate its beliefs in a rational and data-driven manner:

$$P(x|y_1 \wedge \ldots \wedge y_n) = \frac{P(y_n|x)P(x|y_1 \wedge \ldots \wedge y_{n-1})}{P(y_1 \wedge \ldots \wedge y_n)}$$

Markov Localization is a particularly relevant technique in this context. Initially, the agent starts without knowledge of the current state. Through sensors, it perceives part of the current state, allowing the application of conditional probability and updating of beliefs through Bayes' Theorem. This process also occurs when the agent takes actions, perceiving the environment's change and accordingly updating its values.

A key concept associated with Markov is the Markov Decision Process (MDP). This model, characterized by a discrete set of states S, a set of actions A, a transition function $T(s, a, s')$, and a reward function $R(s, a, s')$, provides a formal structure for tackling decision-making problems in the presence of uncertainty.

The main goal in an MDP is to maximize rewards, differentiating itself from traditional search problems by its emphasis on maximizing rewards rather than minimizing costs. The probabilistic approach adopted in Probabilistic Planning enables effective modeling of uncertainty, developing planning strategies that dynamically adapt to the changing conditions of the environment.

2.2 Reinforcement Learning

Reinforcement Learning is a learning technique that involves an agent learning through interaction with a dynamic environment. The agent interacts with the environment in a sequential manner, taking actions and receiving rewards.

The goal of the agent is to maximize the cumulative reward provided by the environment in response to its actions. RL is based on a learning process of exploration and experimentation, where the agent must choose actions to maximize its reward. The agent learns from its experience by accumulating knowledge and developing increasingly effective strategies.

The goal of an agent is thus to maximize the sum of rewards over time:

$$\max \sum_{t=0}^{T} R(s_t, a_t)$$

where T is the maximum time step.

An agent aiming to maximize cumulative reward might face indecision in the case of multiple action sequences with equal total reward. To resolve this indecision, the discount factor γ is introduced, encouraging the agent to prefer quicker maximization of cumulative reward. The reward at time t is given by:

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$$

where r_i is the reward for the transition at time i. This summation is a geometric series and thus always converges to a finite value even for $T = \infty$.

2.3 Q-Learning

Q-Learning is a specific Reinforcement Learning algorithm based on constructing a table indicating the reward value for every possible state upon performing any possible action. The iterative procedure of building the table involves exploring the environment with more or less random actions.

At each step, the table is updated using the following formula:

$$Q[s][a] = Q[s][a] + \alpha \left(r + \gamma \cdot \max(Q[s']) - Q[s][a]\right)$$

Where: - $Q(s, a)$ is the action value estimate for state s and action a, - s is the current state, - a is the taken action, - r is the obtained reward, - s' is the subsequent state, - γ is the discount factor, - α is the learning rate, - $\max(Q[s'])$ returns the highest reward obtainable from state s'.

Initially, the action choice is random until sufficient exploration is deemed to have occurred. The commonly used policy for this purpose is the $\epsilon - greedy$ policy, where ϵ represents the probability of taking a random action, and is gradually reduced to a minimum.

Q-Learning is a powerful and effective technique, capable of learning optimal action strategies across various applications. However, it can be sensitive to noise, action indeterminacy, and managing continuous environments. Moreover, it requires a significant amount of memory to store the table, especially when the environment has a considerable state S and action A space: $(\theta(S \cdot A))$.

2.4 Deep Q-Learning

Deep Q-learning is an advanced version of Q-learning that uses a deep neural network for insights The Q function is rather than a table. This allows us to tackle problems with large state spaces and actions larger and more complex. The DQL is able to generalize better between similar states and can learn representations more useful representations of input data. Additionally, the use of a neural network allows for greater flexibility in learning complex strategies.

Thus, while Q-learning is limited to problems with relatively small state and action spaces and requires a table to store Q values, Deep Q-learning overcomes this limitation by using neutral networks deep rales to approximate the Q function, making it suitable for larger and more complex problems larger

2.5 SARSA Learning

SARSA (State-Action-Reward-State-Action) Learning is a learning algorithm for agent control in reinforcement learning environments. The algorithm maintains a Q function that estimates the action value in a specific state. During learning, the agent explores the environment, selects actions based on its current policy, and updates its Q estimate using the SARSA update equation:

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a)\right]$$

Where: - $Q(s, a)$ is the action value estimate for state s and action a, - α is the learning rate, - r is the reward obtained by performing action a in state s, - γ is the discount factor, - s' is the subsequent state, - a' is the subsequent action selected by the agent's policy.

This equation adjusts the action value estimate based on the obtained reward, the expected subsequent action value, and the learning rate, guiding the adaptation of the agent's policy during learning.

While Q-Learning uses an off-policy strategy, learning from the highest future action value estimate regardless

of the action actually selected, SARSA adopts an on-policy approach. This means that SARSA considers the actions actually chosen during exploration, incorporating the current policy into the action value estimates. This difference makes SARSA more sensitive to exploration policies, ensuring that the agent learns based on the actions it actually takes.

3 Environment

3.1 Introduction

The environment in which the agents are placed is characterized by an intersection with a single traffic light, known as big-intersection.net.xml in Sumo-RL [1], renamed BI.net.xml. This environment defines lanes, traffic lights, directions, and other aspects. There are four lanes on each side of the intersection: one for left turns, two for going straight, and one for going straight or turning right.

Subsequently, this configuration has been modeled into two distinct traffic scenarios, representing varying levels of traffic congestion: low traffic and high traffic. The traffic configurations have been renamed as follows:

1. Low Traffic: $BI\_50.rou.xml$
2. High Traffic: $BI\_150.rou.xml$

The number of vehicles for the traffic situations was determined through a series of iterative experiments to identify optimal conditions. In conclusion, the values that appeared to be most suitable for representing the two situations were:

1. Low Traffic: 50 vehicles per route
2. High Traffic: 150 vehicles per route

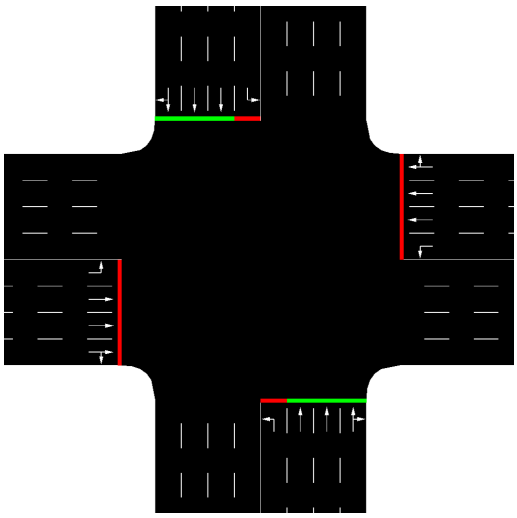

Figure 1: Representation of the environment

3.2 MDP: Observations, Actions, and Rewards

Our model, based on the Markov decision process, emphasizes the definitions provided by Sumo-RL:

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]

where:

- *phase_one_hot* is a vector encoding the current active green phase

- *min_green* is a boolean value indicating whether at least *min_green* seconds have passed during execution
- *lane_i_density* is the number of vehicles arriving in lane i divided by the total capacity of that lane
- *lane_i_queue* is the number of queued vehicles arriving in lane i divided by the total capacity of that lane

The action space is discrete. Every *delta_time* seconds, each traffic signal agent has the opportunity to select the next configuration of the green phase. Each time a phase ends, the next one will be preceded by a yellow phase lasting *yellow_time* seconds. In our case, we will have $|A| = 4$ discrete actions.

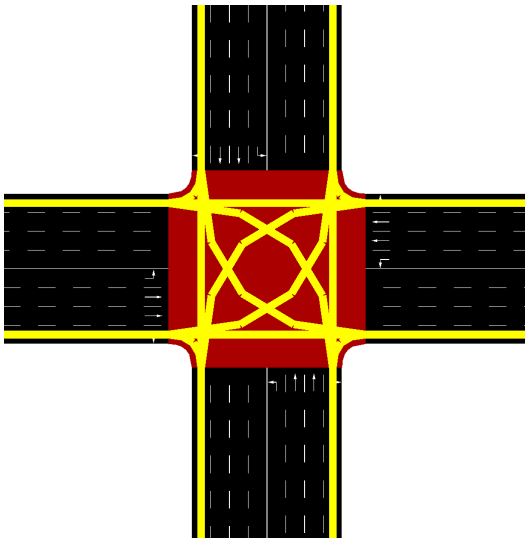

Figure 2: Representation of actions

The reward function, called Differential Waiting Time, is defined as the cumulative change in vehicle waiting times: $r_t = D_{\alpha_t} - D_{\alpha_{t+1}}$

This measure reflects how waiting time, in response to an action, has improved or worsened. The goal is to encourage the agent to take actions that lead to a decrease in total waiting time. If this decreases in the next step, the difference will be positive, indicating a favorable outcome.

3.3 Configurations

The SUMO [2] environment was configured through parameters defined in Sumo-RL:

- Total simulation duration: 100,000 seconds
- Interval between agent actions: 5 seconds
- Duration of yellow phase: 2 seconds
- Minimum duration for a green phase: 5 seconds
- Maximum duration for a green phase: 50 seconds

To evaluate agent performance, four system metrics provided by Sumo-RL were selected:

- *system_total_stopped*: represents the total number of vehicles stopped in the current step.

- *system_total_waiting_time*: indicates the sum of simulation time in seconds that vehicles have been stopped since the last time.
- *system_mean_waiting_time*: represents the arithmetic mean of all vehicle waiting times.
- *system_mean_speed*: indicates the arithmetic mean of vehicle speeds.

4 Study

To assess whether reinforcement learning algorithms perform better in traffic management compared to a fixed-cycle traffic light, models were trained under both light and heavy traffic conditions, and then tested in both the same training conditions and different conditions.

Evaluation utilized the four metrics provided by SUMO-RL. Only plots related to average waiting time will be presented. The results obtained are as follows.

For a better understanding of the images or to compare other metrics, it is recommended to refer to the repository RL-Traffic-Control.

4.1 Low Traffic - Low Traffic

In this experiment, agents were trained in a low traffic environment and then tested in the same environment.

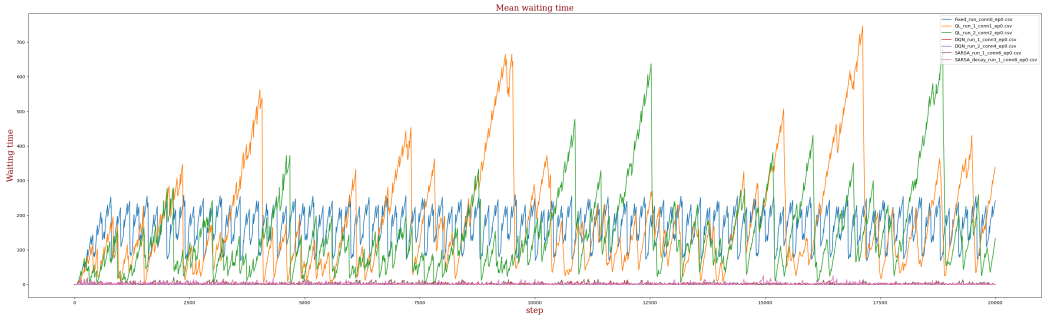

Figure 3: Mean Waiting Time: low traffic - low traffic

It is highlighted that Q-Learning agents perform worse and show greater oscillation compared to the fixed-cycle traffic light, while Sarsa and DQN agents offer significantly better performance than the fixed cycle.

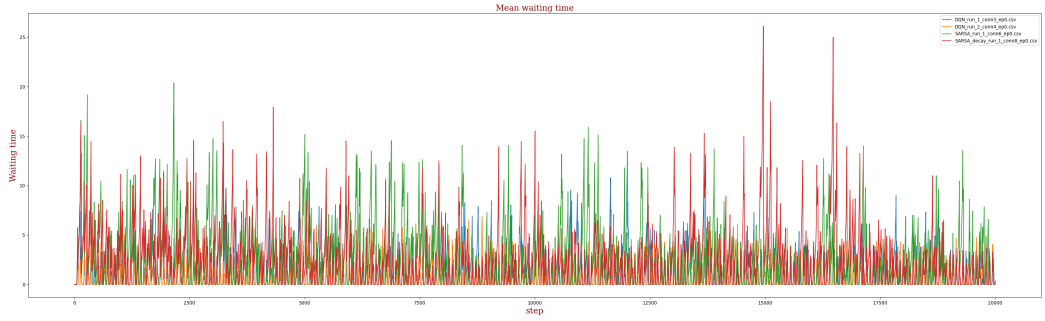

Figure 4: Mean Waiting Time DQN and SARSA: low traffic - low traffic

In this visualization, QL and FC agents have been removed for better observation. It is noted that DQN agents offer relatively better performance compared to agents with the SARSA strategy.

4.2 Low Traffic - High Traffic

In this experiment, agents were trained in a low traffic environment and then tested in a high traffic environment.

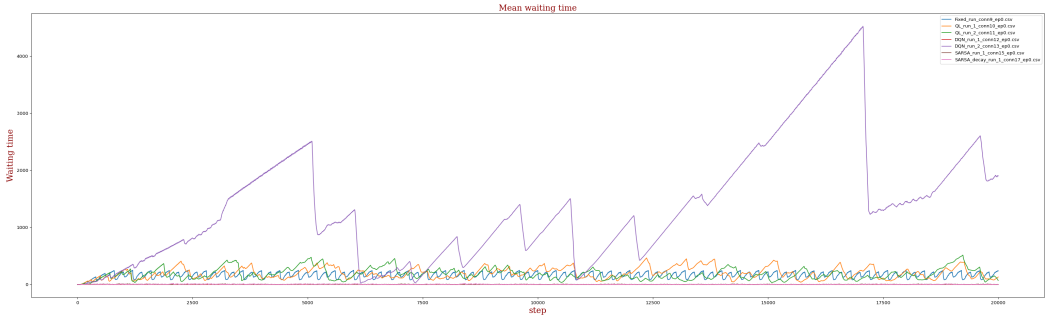

Figure 5: Mean Waiting Time: low traffic - high traffic

It is quickly observed that a DQN agent surprisingly offers significantly worse performance compared to the fixed cycle, which behaves similarly to Q-Learning agents. Below is another image with better visualization, excluding Q-Learning agents, the fixed cycle agent, and the DQN agent with poor performance.

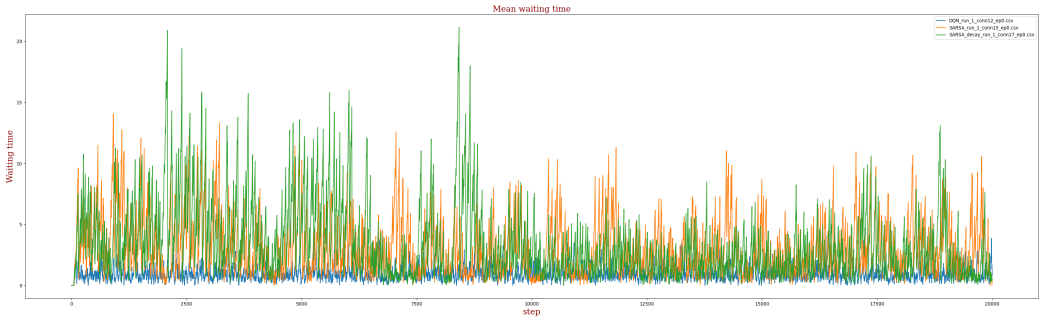

Figure 6: Mean Waiting Time Sarsa and DQN: low traffic - high traffic

With this new visualization, it is evident that the DQN agent offers significantly better performance than SARSA.

4.3 High Traffic - Low Traffic

In this experiment, agents were trained in a high traffic environment and then tested in a low traffic environment.

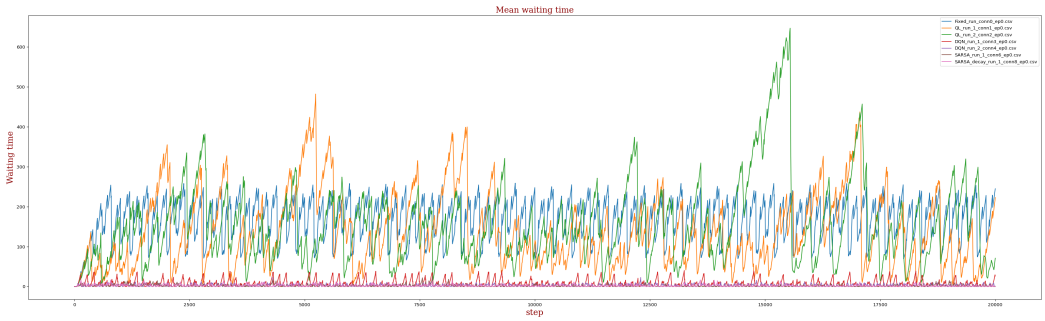

Figure 7: Mean Waiting Time: high traffic - low traffic

Similar to the "low traffic - low traffic" case, Q-Learning agents perform worse and show greater oscillation compared to the fixed-cycle traffic light, while Sarsa and DQN agents offer significantly better performance than the fixed cycle.

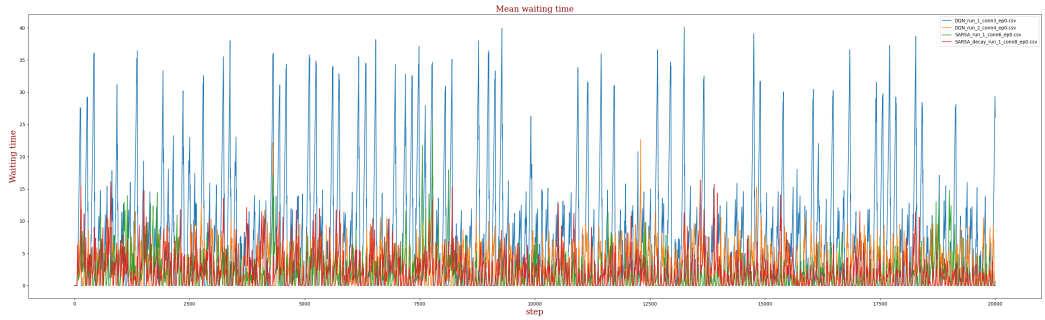

Figure 8: Mean Waiting Time DQN and SARSA: high traffic - low traffic

Unlike the visualization on DQN and SARSA with training on the same traffic scenario, here SARSA agents perform relatively better compared to DQN.

4.4 High Traffic - High Traffic

In this experiment, agents were trained in a high traffic environment and then tested in the same environment.

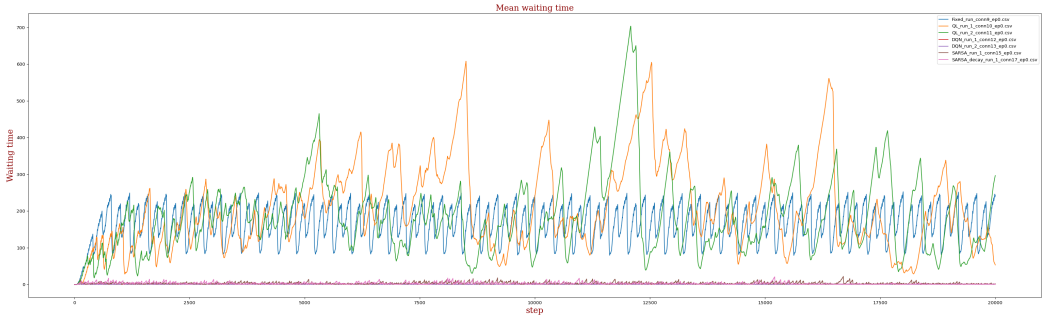

Figure 9: Mean Waiting Time: high traffic - high traffic

Similar to the "low traffic - high traffic" case, Q-Learning agents perform worse than the fixed cycle, while SARSA and DQN are better, although it is not clear who is better between them. Below is a better visualization.

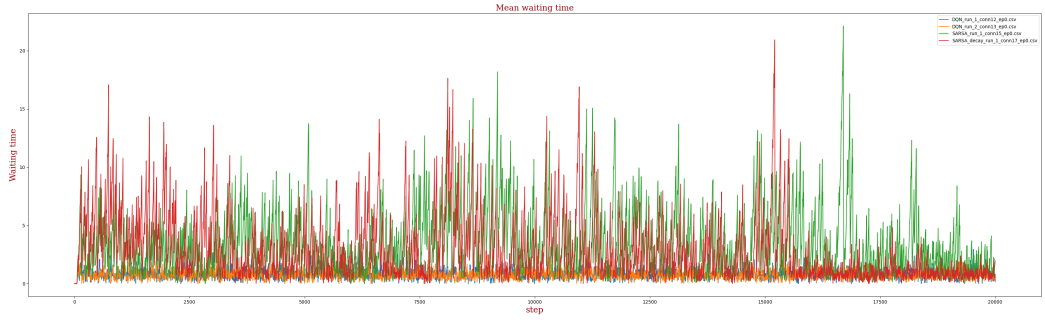

Figure 10: Mean Waiting Time Sarsa and DQN: high traffic - high traffic

In this case, both a DQN agent and a SARSA agent offer high performance. (orange and blue, not very visible)

5 Conclusions

In conclusion, the results of this study clearly demonstrate that reinforcement learning algorithms, such as Deep Q-Learning and SARSA, are able to significantly improve traffic light management performance compared to the traditional fixed-cycle strategy for the intersection used.

The Q-Learning algorithm, on the other hand, often showed a deterioration in performance compared to the baseline strategy.

6 Conclusion

Reinforcement Learning, with its various algorithms like Q-Learning, Deep Q-Learning, and SARSA, presents a comprehensive framework for teaching agents to make decisions in dynamic environments. By navigating the trade-offs between exploration and exploitation, these algorithms enable agents to learn optimal policies for complex problems. Despite the challenges of large state-action spaces and the requirement for significant computational resources, advancements in RL continue to push the boundaries of what's possible in autonomous decision-making and adaptive systems.

References

[1] Lucas N. Alegre. SUMO-RL. `https://github.com/LucasAlegre/sumo-rl`, 2019.

[2] DRL. Eclipse SUMO - Simulation of Urban MObility. `https://github.com/eclipse-sumo/sumo`, 2023.