Comparison of On-Policy vs Off-Policy Reinforcement Learning Algorithms for Traffic Signal Control

Ashley Danielle Aguilar[†], Brandon DeRosier[†], Richard Oh[†] and Matthew Williams[†]

Abstract
This paper presents a comparison of the performance of the Proximal Policy Optimization (policy) and Deep Q Learning (off-policy) algorithms in traffic signal control problems. The simulation environment was provided by CityFlow, an open-source traffic simulator, and OpenAI's Gym API to interface between the environment and the reinforcement learning model.

Keywords
reinforcement learning, traffic signal control, Proximal Policy Optimization (PPO), Deep Q-Learning

1. Introduction

Modern traffic lights use timers to control the flow of traffic, with some implementing sensors to detect the presence of vehicles. As the number of cars on city streets increases, this approach suffers from several issues. Severe traffic congestion has been found to be both harmful to health[1] and to local economies[2]. Efforts have been made to improve the throughput at signal-controlled intersections using reinforcement learning (RL) algorithms [3]. This investigation was designed to compare the differences in performance of two different classes of RL algorithm— off-policy and on-policy. A timed model was also created to serve as a baseline performance indicator in order to measure whether our RL models are improving on the current approach.

2. Background

This section seeks to provide background on the topics of reinforcement learning that will be used in the rest of the paper, including an explanation of the decision process, policy, observations, actions, and rewards.

2.1. Reinforcement Learning

Single-agent reinforcement learning problems can be modeled after the Markov Decision Process[4] (S, A, P, R) where:

- S is the set of states of the environment (*state space*, or *observation space* as the agent observes the environment's state $s \in S$)
- A is the set of actions that the agent can take (*action space*, where the agent can take the action $a \in A$)
- P represents state transition probabilities $P(s|a)$
- R represents the reward function $R(s, a)$

Some reinforcement learning algorithms also seek to optimize a policy $\pi(s|a)$ as described in the following section.

[†]These authors contributed equally.

2.2. Policy

The policy is the strategic map that the algorithm uses to determine the action to take based on what state it's in. The optimal policy is the one in which the reward is maximized. There are two things that the model can learn: the behavior policy $\beta(a|s)$ contains the information on the value of doing different actions, which the agent uses to select the action based on the current state, and the target policy is the policy that the agent is trying to learn.

An on-policy learning algorithm uses and updates the same policy as it takes actions and gets feedback from the environment. An off-policy algorithm uses a different policy to select actions than the one that is evaluated and optimized from data samples. Reasons that off-policy learning may be used include the freedom to explore other actions while the optimal policy is being learned.

2.3. Observations, Actions, and Rewards

Observations capture information about the state of the environment at a given time step. The set of all possible states is the *observation space* S and at time t, $s^t \in S$. *Actions* describe ways the agent can behave with respect to its environment, and the set of possible actions is the *action space* A. At the time t, the agent can take action $a^t \in A$ and receives a reward that is determined by the reward function $R(s^t, a^t)$.

3. Simulation Environment

In this section, we describe the setup of the traffic simulation environment. This includes the configuration of the environment along with the libraries and packages we used.

3.1. CityFlow & OpenAI Gym

CityFlow is an open-source reinforcement learning environment for traffic scenarios. It provides an extensive API for running the simulation, configuring traffic generation, and intersection mapping[5]. In conjunction with CityFlow, we used Gym which is an open-source Python library by OpenAI for interfacing between the environment and reinforcement learning algorithms. The library also allows for the creation of custom environments to represent specific tasks. For our purposes, we adapted a gym environment to work with CityFlow (gym-cityflow).

3.1.1. Traffic Generation

A vital part of the gym-cityflow environment is the flow.json file. This file determines how traffic is generated for the intersection and all foundational parts of the intersection. This includes the size of vehicles, their maximum speeds, acceleration rates, the spacing of traffic, and the rate at which cars are generated. A standardized intersection is used to get comparable results for each of our models and approaches. The flow file used in this project describes a single intersection with seven lanes traveling north, south, east, and west. Each direction has seven lanes with two left turn lanes, three straight traveling lanes, and two right turning lanes. Vehicles are generated using a semi-realistic constant traffic flow, meaning that cars will not be generated at a constant rate evenly over the given lanes in the intersection. Instead, we opted to weigh the generation of cars depending on which lanes they were traveling to more closely resemble a real-life intersection. This means that 60% of cars would travel straight and are generated at a rate of one car every three seconds. Cars turning right represent 30% of traffic and are generated at a rate of one car every six seconds. Finally, cars turning left represent 10% of traffic in the intersection and are generated at a rate of one car every eighteen seconds.

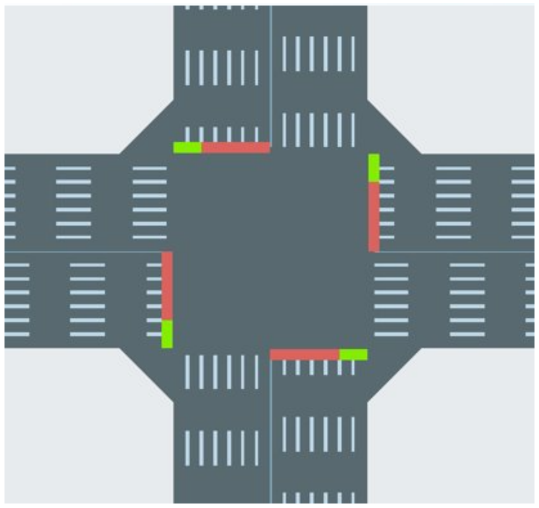

Figure 1: Layout of the standardized intersection with seven lanes going north to south, south to north, east to west, and west to east.

3.1.2. Simulation Step

For this project, a single "step" is equal to 1 second. This definition is crucial to several calculations (e.g. Cumulative Wait Time, a performance metric discussed in a later section).

3.2. Observation Space

For the traffic signal control problem, observations are discrete states describing the number of waiting vehicles in each lane. We defined the observation space as a gym dictionary space since the CityFlow function `'get_lane_waiting_vehicle_count()'` returns a dictionary where the *keys* are the names of lanes and the *values* are the number of vehicles in the given lane with speed less than 0.1 m/s. This produces a realistic input for the model since many real-world traffic signals have sensors that can detect the presence of vehicles waiting at the light. Additionally, since queue length can be correlated with the performance of an intersection, it also provides an accurate representation of the intersection's state[6].

To alleviate constantly changing phases, we expand the observation space to include the number of consecutive steps that the environment has been in the current phase. Without this, the model had no basis for correlating the relationship between the observation space and a reward based on phase times. Once the number of steps in the current phase was added to the observation space, developing a reward function to encourage models to stay in the same phase was straightforward.

3.3. Action Space

The action space was originally defined as a gym `MultiDiscrete` space where the actions are sets of values corresponding to the phases of the intersections to allow the gym-cityflow environment to work with both individual intersections and systems of intersections. However, as development continued, the focus shifted to individual intersections. This led to a change in the action space to a `Discrete` space, making our environment compatible with a greater amount of reinforcement learning approaches and allowing us to experiment with a larger pool of models.

3.4. Reward Function

The reward function was discovered to play a critical part in how well each model performed, so a large amount of time was dedicated to finding the optimal equation. Initial reward functions were based on functions defined and tested in an assessment conducted by researchers in the UK [7]. The first renderings of our models changed phases too frequently to be a feasible solution. There were many struggles in getting the algorithms to yield results that resembled a realistic intersection setting, thus our main goal became to limit the frequency of phase changes. A number of reward functions were tested, including:

- Queue Sum Reward Function: returns the total sum of vehicles waiting at the intersection
- Average Speed Reward Function: returns the sum of each vehicle's speed divided by the number of vehicles
- Phase Time Reward Function: returns a reward based on the number of steps the simulation has been in the current traffic light phase (reward grows the longer we have been in the current phase up to a set maximum amount)

3.4.1. Combined Reward Functions

Reward functions were also implemented that determined rewards based on a combination of factors. One implementation was a "Combo-Speed" function that used the Phase Time Reward Function in conjunction with the Average Speed Reward Function, with the goal being to reward the reinforcement model for staying in phases longer and maximizing the average speed of vehicles. However, this reward function did not lead to the desired results as the model would heavily favor a single phase over all the others leading to high average speeds for vehicles in those lanes and leaving the other lanes in gridlock.

The second combined reward function used a combination of the Phase Time Reward Function and Queue Sum Reward Function. The goal was to reward the model for staying in phases longer and reduce the queues in each lane. Although not quite as prevalent, we did notice the same issue occurring that was found in the previous function, as is shown in Figure 2.

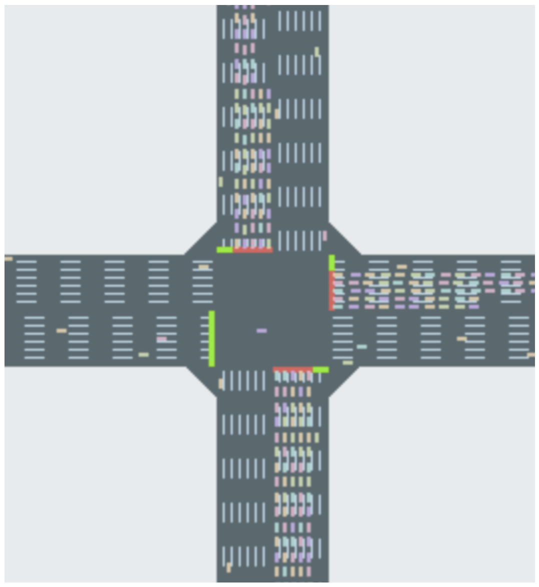

Figure 2: The Combo-Queue reward function demonstrates a preference for a certain phase.

4. Reinforcement Learning Algorithms

This section explains the difference between on-policy and off-policy RL algorithms, as well as the specifics for each particular algorithm used in this project. These include Proximal Policy Optimization (PPO) and Deep Q-Networks (DQN).

4.1. On-Policy Algorithms

When On-Policy algorithms are learning, the agent will use a behavior policy that determines what action it should take. The result of this learning will be a target policy that fully maps the actions with the given state space and a reward. On-policy algorithms use the most updated version of the target policy to determine the next action as its behavior policy. Rather than choosing its next action through an off-policy method such as randomly choosing the next action, it will use the most up-to-date policy to select the next action and continue to evaluate and improve on it.

4.1.1. Proximal Policy Optimization (PPO)

Proximal Policy Optimization is a policy gradient method, meaning that the algorithm's objective is to optimize a parameterized policy using stochastic gradient descent. The objective function, as described in the 2017 breakthrough paper "Proximal Policy Optimization Algorithms" by John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov [8], is as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

where ϵ describes a policy parameter, $\hat{\mathbb{E}}_t$ is an "empirical average" (i.e. arithmetic mean) over time steps t, $\hat{A}_t$ is an estimator for an advantage function at time step t, ϵ is a hyper-parameter, and $r_t(\theta)$ is a ratio of probabilities. The $CLIP$ denotes that the amount that the policy can update is scaled down from older algorithms.

4.2. Off-Policy Algorithms

Off-policy algorithms use a behavior policy that is entirely different from the target policy. This policy can be anything such as randomly choosing an action. Off-policy algorithms tend to be far more exploratory than on-policy algorithms since on-policy algorithms only use a single deterministic policy, and they can miss alternative behaviors that might have produced better outcomes.

4.2.1. Deep Q-Network (DQN)

Deep Q-Learning utilizes Q-learning, an algorithm that tries to learn a Q-function. A Q-function takes the action and state as inputs and outputs a reward. A Q-learning algorithm's goal is to maximize the rewards of this Q-function. The Q-function utilizes the Bellman equation as follows:

$$Q^{\pi}(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|s_t, a_t]$$

where s_t and a_t are the state and action respectively at time-step t, π is the policy, γ is the discount rate (a factor that determines how much the agent cares about rewards in the distant future relative to the ones in the immediate future), and $E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...]$ is the expected discounted cumulative reward. The action chosen for each new Q value is based on the behavioral policy. As the agent learns it will try to maximize the Q value[9]. As such, the behavioral policy used is a greedy policy.

DQN uses a neural network to approximate the Q-function. Once a Q-function is approximated, the network provides a function that outputs the action that provides the maximum reward for a given state. It is important to note that DQN does not work with continuous action and observation spaces as continuous functions are hard to maximize due to a large number of potential actions. For continuous spaces, PPO is much better suited[10]. Our implementation uses a two-layered neural network with a discount rate of 0.99 and a learning rate of .0001.

5. Performance Metrics

The ability to evaluate and quantify intersection metrics is a vital part of evaluating a model's performance. The chosen metrics are those that most effectively evaluated the efficiency of the reinforcement learning models[11]. Additionally, It is important to consider all of the metrics in conjunction to get the entire story for each intersection and how each model is handling the traffic.

5.1. Cumulative Wait Time

The cumulative wait time is equal to the combined wait time throughout the entire intersection. This metric totals all of the time vehicles spend waiting at a traffic signal. At each time step `'get_lane_waiting_vehicle_count()'` is called and added to a grand sum. This is an important metric to track because it shows how much global delay an intersection is causing.

5.2. Average Speed

Average speed averages the traveling speed of every vehicle in the intersection. This is calculated by finding the sum of speeds for every vehicle and then dividing it by the number of vehicles in the intersection. More efficient intersections will have a higher average speed.

5.3. Average Queue Length

Average queue length is the average number of cars waiting in each lane. We calculate the average queue length by taking the total number of cars waiting and dividing that by the number of lanes in the intersection. This is an important evaluator in indicating how well the lanes are being managed. With average speed and total wait time, it is possible to mask the light not changing phases but the average queue length can show us if the queues are really long in one intersection versus another.

6. Results

The following section presents the results for the traffic signal problem with each method. We begin with the baseline timed model and then evaluate the PPO and DQN model results. Finally, we overlay the results and compare them.

6.1. Actuated

The results for the actuated model which served as a baseline to compare our various reinforcement learning approaches are shown below. The model is configured to spend 30 seconds in each of four phases with five-second "transition phases" in between where only right turns are allowed.

In general, this model functions well despite its degrading performance over time and serves as a realistic comparison for our various models. Plots displaying the average queue, wait time, and average speed of the intersection over the course of the simulation are shown in Figure 3.

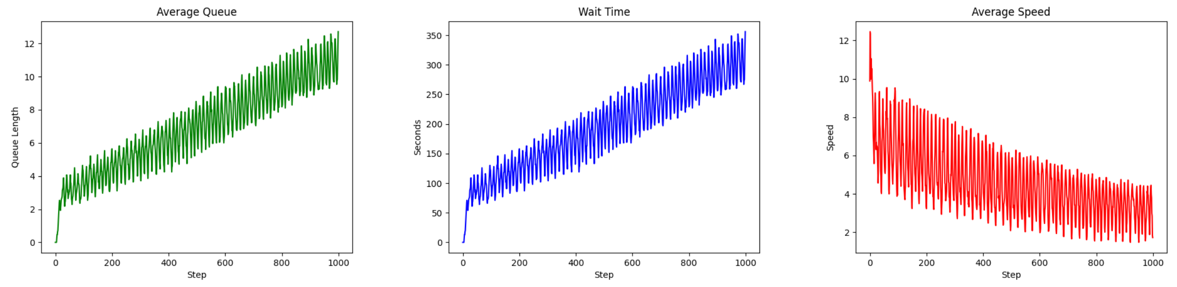

Figure 3: Average Queue Length, Average Speed, and Wait Time for an actuated single intersection with 30-second phase change time.

6.2. Deep Q-Learning

The performance of the DQN model was below that of the baseline. Rather than steadily increasing the average queue and wait times as seen within the baseline plots, the line for the DQN average queue and wait time increases at a higher rate than the baseline results before reaching a plateau. At this point, the simulated intersection has reached its maximum capacity. Thus, the average speed of the intersection decreases dramatically in comparison to the baseline. These patterns can be seen in the plots for each metric in Figure 4.

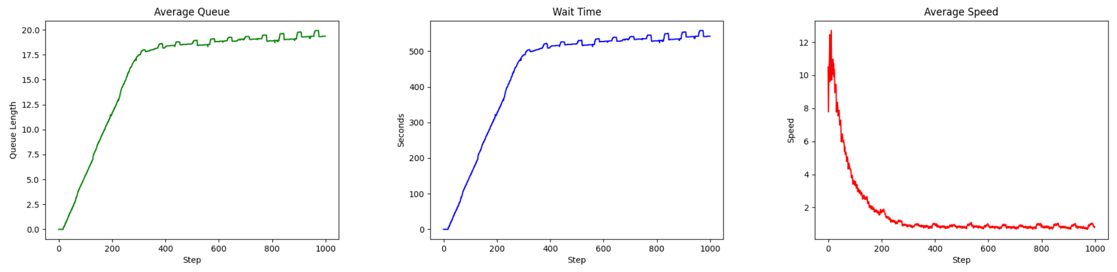

Figure 4: Average Queue Length, Average Speed, and Wait Time for a Deep Q-Learning model on a single intersection.

6.3. Proximal Policy Optimization

The PPO model results follow the same general trends as the DQN model. The queue and wait times steadily increase until they hit a plateau and the average speed sharply decreases before stabilizing around 1 m/s, as seen in Figure 5.

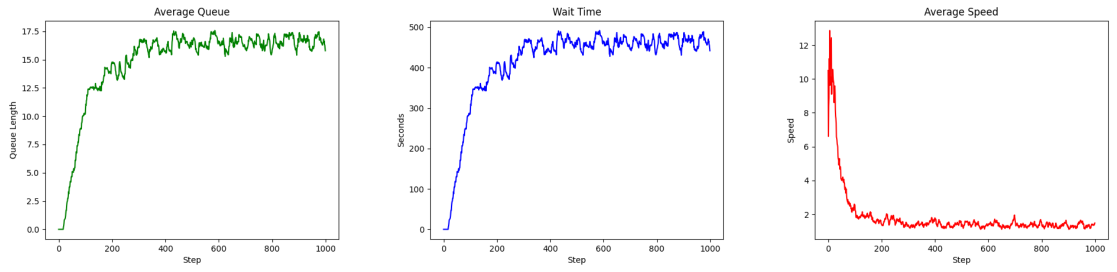

Figure 5: Average Queue Length, Average Speed, and Wait Time for a Proximal Policy model on a single intersection.

6.4. Comparisons

Figure 6 overlays the results from each model for comparison. This shows we are a ways off from providing a reinforcement learning solution that improves upon the standard approach. We feel these results are not an accurate representation of the performance of the RL algorithms. It is more likely a result of possible flaws with the reward functions, observation space, or action space.

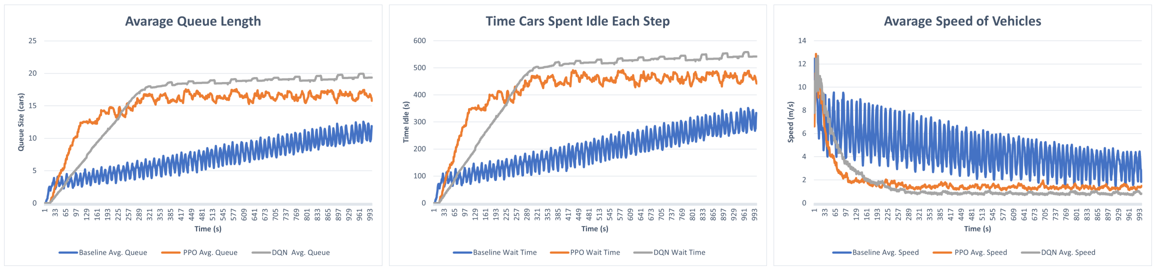

Figure 6: Average Queue Length, Average Speed, and Wait Time for the 3 approaches in the same plot.

7. Conclusion

In conclusion, our implementations of reinforcement models do not perform as well as the baseline timed model in increasing the efficiency of a single intersection. While PPO resulted in improved queue lengths, speed, and waiting time over DQN, both PPO and DQN developed the issue where one lane will tend to stay in a green phase while the others are stopped. The marginal improvement in the performance of on-policy RL over off-policy RL for traffic signal control has been found in other comparative studies for both single-agent [12] and multi-agent [13] systems.

8. Ideas for Project Extensions

Possible extensions for this project include:

- Modify the combination reward functions to add an additional incentive to change to phases that the model hasn't selected in a while. Ideally, this would help balance the time spent in each phase and avoid a situation where an intersection is spending most of its time in a single phase.
- Try different approaches for the gym-cityflow environment, such as expanding the action space to accept both the intersection phase and the time the environment will spend in that phase. This approach would allow us to avoid some of the issues we encountered with overly short phase times and models only using a few phases.
- Explore optimizing systems of intersections by implementing multi-agent reinforcement learning and modifying gym-cityflow environment to handle multiple intersections[14].

References

[1] J. I. Levy, J. J. Buonocore, K. von Stackelberg, Evaluation of the public health impacts of traffic congestion: a health risk assessment, Environmental Health 9 (2010).

[2] D. Schrank, B. Eisele, T. Lomax, 2019 Urban Mobility Report, Technical Report, 2019.

[3] H. Wei, G. Zheng, V. Gayah, Z. Li, Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation, SIGKDD Explor. Newsl. 22 (2021) 12–18. URL: https://doi.org/10.1145/3447556.3447565. doi:10.1145/3447556.3447565.

[4] H. Chaudhuri, V. Masti, V. Veerendranath, S. Natarajan, A comparative study of algorithms for intelligent traffic signal control, in: Machine Learning and Autonomous Systems, Springer Nature Singapore, 2022, pp. 271–287. URL: https://doi.org/10.1007%2F978-981-16-7996-4_19. doi:10.1007/978-981-16-7996-4_19.

[5] H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, Z. Li, CityFlow: A multi-agent reinforcement learning environment for large scale city traffic scenario, in: The World Wide Web Conference, ACM, 2019. URL: https://doi.org/10.1145%2F3308558.3314139. doi:10.1145/3308558.3314139.

[6] W. Genders, S. Razavi, Evaluating reinforcement learning state representations for adaptive traffic signal control, Procedia Computer Science 130 (2018) 26–33. URL: https://www.sciencedirect.com/science/article/pii/S1877050918303582. doi:https://doi.org/10.1016/j.procs.2018.04.008, the 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops.

[7] A. C. Egea, S. Howell, M. Knutins, C. Connaughton, Assessment of reward functions for reinforcement learning traffic signal control under real-world limitations, in: 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2020. URL: https://doi.org/10.1109%2Fsmc42975.2020.9283498. doi:10.1109/smc42975.2020.9283498.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017. URL: https://arxiv.org/abs/1707.06347. doi:10.48550/ARXIV.1707.06347.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, 2013. URL: https://arxiv.org/abs/1312.5602. doi:10.48550/ARXIV.1312.5602.

[10] T. N. Larsen, H. Ø. Teigen, T. Laache, D. Varagnolo, A. Rasheed, Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters, Front. Robot. AI 8 (2021) 738113.

[11] S. Touhbi, M. A. Babram, T. Nguyen-Huu, N. Marilleau, M. L. Hbid, C. Cambier, S. Stinckwich, Adaptive traffic signal control: Exploring reward definition for reinforcement learning, Procedia Computer Science 109 (2017) 513–520. URL: https://www.sciencedirect.com/science/article/pii/S1877050917309912. doi:https://doi.org/10.1016/j.procs.2017.05.327, 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal.

[12] Y. Zhu, M. Cai, C. Schwarz, J. Li, S. Xiao, Intelligent traffic light via policy-based deep reinforcement learning, 2021. URL: https://arxiv.org/abs/2112.13817. doi:10.48550/ARXIV.2112.13817.

[13] T. Cui, X. Liu, L. Zhang, Distributed optimization of regional traffic signals via deep reinforcement learning, in: 2021 40th Chinese Control Conference (CCC), 2021, pp. 6130–6135. doi:10.23919/CCC52363.2021.9550100.

[14] S. Kapoor, Multi-agent reinforcement learning: A report on challenges and approaches, 2018. URL: https://arxiv.org/abs/1807.09427. doi:10.48550/ARXIV.1807.09427.