SUMO User Conference 2022
https://doi.org/10.52825/scp.v3i.116

Published: 29 Sept. 2022

A Comparison of Reinforcement Learning Agents Applied to Traffic Signal Optimisation *

Jacobus Louw[1], Louwrens Labuschange[1], and Tiffany Woodley[1][https://orcid.org/0000-0002-3651-6426]

[1]ByteFuse AI, Stellenbosch, South Africa

cobus.louw@bytefuse.ai, louwrens.labuschange@bytefuse.ai, tiffany.woodley@bytefuse.ai

Abstract

Traditional methods for traffic signal control at an urban intersection are not effective in controlling traffic flow for dynamic traffic demand which leads to negative environmental, psychological and financial impacts for all parties involved. Urban traffic management is a complex problem with multiple factors affecting the control of traffic flow. With recent advancements in *machine learning* (ML), especially *reinforcement learning* (RL), there is potential to solve this problem. The idea is to allow an agent to learn optimal behaviour to maximise specific metrics through trial and error. In this paper we apply two RL algorithms, one policy-based, the other value-based, to solve this problem in simulation. For the simulation, we use an open-source traffic simulator, *Simulation of Urban MObility* (SUMO), packaged as an OpenAI Gym environment [3, 9]. We trained the agents on different traffic patterns on a simulated intersection [24]. We compare the performance of the resultant policies to traditional approaches such as the Webster and *vehicle actuated* (VA) methods. We also examine and contrast the policies learned by the RL agents and evaluate how well they generalise to different traffic patterns.

Contents

*Other people who contributed to this document include Greg Newman, Etienne Barnard, Ruan van der Merwe and Tessa Smit.

Accepted at SUMO conference 2022. This paper contains no competing interests. All authors made equal contributions.

1 Introduction

Traffic light intersections are responsible for directing large volumes of traffic flow, making their optimal control critical. Negative environmental, psychological, and financial impacts are associated with traffic build up, resulting from inefficient control [14]. Currently, traffic lights predominantly use control methods which require resource intensive configuration[18, 15]. This configuration requires a large amount of engineering hours, making these systems costly to implement and maintain [5].

To improve financial viability, we propose a low-cost control system that can obtain similar or improved functionality in an automated manner. This system is implemented using an RL agent that can be trained in a simulated environment, thereby removing the need for costly configurations. Practically, implemented systems predominantly control a single intersection. Our approach is trained and tested on a single intersection, providing a low maintenance alternative to current solutions.

The promise of using RL for the application of traffic light control has been demonstrated in previous research predominantly utilising Deep Q-learning [24, 1, 8]. There has been emergent research into using *proximal policy optimisation* (PPO) due to its ease of implementation and encouraging results [12]. The main focus of this paper is to directly compare Deep Q-learning and PPO on the traffic light optimisation application. A comprehensive comparison is completed by evaluating the method's policy performance, not only in terms of average performance, but also the robustness of the chosen strategy.

For direct comparison to current implementations, our achieved results are compared to traditionally used methods on a simulated intersection. These traditional methods include a gap-based actuated setup, a time-loss actuated setup, and a fixed-time approach which makes use of the Webster equation [17, 2].

2 Related work

Traffic signal timing optimisation is a complex task which necessitates a high level of expertise in the industry. Many methods varying in complexity have been developed for this application. Despite much work on the development of these methods, even more sophisticated methods

struggle to accurately predict traffic operation at intersections. This difficulty is due to a wide variety of influencing factors.

A commonality across methods is the evaluation of the model's performance. The performance can be quantified by using either a performance index or a measure of the level of service. These are generally calculated using metrics such as number of vehicle stops at the intersection, vehicle delay etc.

Traditionally used methods can be split into two types: actuated and fixed-time approaches. Both methods have limitations: actuated approaches perform best at isolated intersections and fixed-time approaches are unable to dynamically respond to unexpected increases in traffic. These limitations have opened up new research branches into adaptive methods and RL which, whilst still new, show much promise for this application.

2.1 Fixed-time control

There are several methods for controlling the timing of traffic signals which require manual configuration. Fixed-time methods fall into this category where cycle stages of the different traffic light phases are set ahead of time and once set their operation will not deviate. In order to set the cycle stages different evaluations can be used, the most common one is based on an equation developed by Webster [22]. His formula can be used to calculate the optimal cycle length that would minimise the total delay at an intersection

$$C_o = \frac{1.5 \cdot L + 5}{1 - \sum Y_i}, \tag{1}$$

where C_o is the optimum cycle length in seconds, L is the total lost time per cycle in seconds, and Y_i is the volume/saturation flow ratio per critical movement in stage i.

Webster further elaborates on the fact that the equation's proposed cycle lengths have some leeway. Cycle lengths chosen within the range of $0.75C_o$ to $1.50C_o$ should not significantly increase the delay. Although robust to small deviations in the cycle length, this equation is very sensitive to the accuracy of the metrics of lost time and saturation flow given to the equation. It further is unable to account for pedestrian traffic.

2.2 Vehicle-actuated control

Vehicle-actuated approaches allow for deviation in their response based on how many vehicles are detected as well as pedestrian buttons pushed [20]. This deviation in cycle times allows for a more tailored approach to current traffic patterns. When at an isolated intersection where traffic follows a more sporadic pattern, vehicle-actuated control can provide considerable reductions in delay when compared to fixed-time approaches [13]. This is due to the fact that fixed-time approaches cannot readily adapt to the randomness in vehicle arrivals.

To implement such an approach, first a fixed-time approach is established; the actuated control strategy can then extend the set cycle times if there are queues at the receiving green lanes. If these queues were cut off too soon by a red light, the overflow of vehicles would have to be discharged at the next green light, causing delays. The queues can be established by measuring the gap between vehicles, if there is a long gap between vehicles it is an indication that the queue has been discharged.

2.3 Reinforcement learning

Adaptive controllers, for example *split cycle offset optimisation technique* (SCOOT) and *Sydney coordinated adaptive traffic* (SCATS), have been shown to improve upon actuated methods, however the difficulty of the initial manual tuning process still remains resource consuming and costly [6, 10, 18]. Both the advancements in computational methods and amount of traffic data stored have made *reinforcement learning* (RL) a feasible solution to the problem. Since RL can be trained in a simulated environment, it removes the initial costly barrier.

RL has the ability to learn optimal control in dynamic and uncertain environments [19]. This ability makes it well suited to the traffic light optimisation problem which needs to dynamically react to changing traffic patterns. Across emergent research into this approach the traffic light simulation tool, SUMO, has been used to train and evaluate implemented RL systems [24].

A relatively simple approach is to apply tabular RL methods to this problem [21]. Although effective on smaller case studies such as single intersections, these methods do not scale well to problems with larger state spaces. When we apply RL to our application, the state space of more complex intersections becomes far too large to represent with a table [23]. The solution to addressing the large state space lies in interpolation and approximation – being able to use a small subset of states to generalise and make decisions over a much larger subset. This can be achieved by using function approximation to approximate the value function, allowing problems with high-dimensional state spaces to be solved. Various function approximation techniques can be used in combination with RL, but recently *artificial neural networks* (ANNs) have become the commonly method used to represent value functions and policies. The combination of ANNs and RL is known as *deep reinforcement learning* (DRL).

Deep Q-learning is a popular DRL algorithm that utilises an ANN in conjunction with Q-learning [11]. The resultant model is known as a *deep Q-network* (DQN). The benefits of using a DQN are highlighted by Wei et al [24]. These authors develop a DQN for the purpose of optimal traffic control, the DQN is improved by implementing state-of-the-art techniques such as *experience replay* (ER) and a target network. Their paper further investigates how robust the DQN is to different traffic patterns. They suggest that further improvement can be made to the performance of the DQN in terms of convergence and stability.

Another popular RL approach is using policy-based methods. Traditional policy-based methods tend to destabilise if not constrained. *Trust region policy optimisation* (TRPO) methods were developed with the aim of solving this problem by restricting the magnitude of the changes allowed to be made to the policy. PPO is an emergent method which has all the same benefits as TRPO methods, with the added advantages of easy implementation and better sample complexity. This method is also data efficient, as it is able to run multiple epochs on a single sample and has been shown to outperform other policy gradient methods on applications such as Atari [16]. Mousavi has demonstrated that PPO can achieve comparable results to a DQN in the context of traffic light optimisation, making it a promising area for further research [12]. This paper makes a direct comparison between the two methods in terms of robustness and strategy implementation to help formulate the direction of future research.

3 Experimental setup

The traffic light intersection is simulated using *Simulation of Urban MObility* (SUMO), an open source traffic simulation package [9]. The intersection geometry and traffic patterns used to simulate the simple intersection used for our experiments are described in Sections 3.1 and 3.2. Phase changes can be made in the simulation to directly control the generated traffic; these

phase changes are detailed in Section 3.3.

3.1 Intersection layout

SUMO was used to simulate the intersection used in Wei et al's paper which can be visualised in Figure 1 [24]. This simulated intersection was used to both train and test the RL agents, as well as test the traditional methods used as benchmarks.

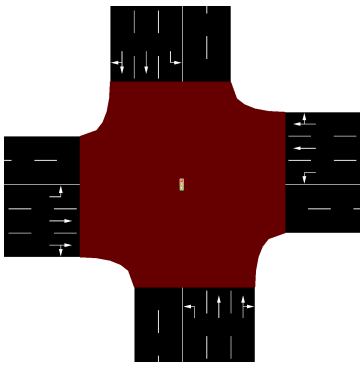

Figure 1: The intersection geometry used throughout this paper [24].

3.2 Traffic patterns

Four different synthetic traffic patterns are employed by Wei et al. to capture the spatial-temporal aspects of urban traffic [24]. The four traffic patterns were simulated in the SUMO environment, and include:

- A traffic distribution which simulates higher traffic flow patterns on the major roads over the minor roads (P1).
- A traffic pattern which simulates higher traffic flow on the left-turn lanes over the through lanes (P2).
- A tidal traffic pattern, where two perpendicular lanes have higher traffic flow (P3).
- Time based varying traffic patterns, where one of the major lanes has time varying traffic patterns, where the other lanes kept a steady traffic flow (P4).

These synthetic traffic patterns were generated using a binomial distribution with the arrival rates specified in Table 4. Since all these are feasible traffic patterns, the methods were tested on all four patterns. Testing on the different traffic patterns gives us a more complete idea of how the agent operates.

3.3 Traffic light phase definitions

Phase changes in the simulation can be made to directly control the generated traffic; these phase changes are detailed in Section 2. The links between lanes represent the possible directions a vehicle can take from a given lane. If the link is green the vehicle can move through the green link and leave the intersection. If the link is red the vehicle will have to wait for a phase change

where the corresponding link is green. Purple links indicate a stop for right turning vehicles. These vehicles may turn right when it is safe to do so.

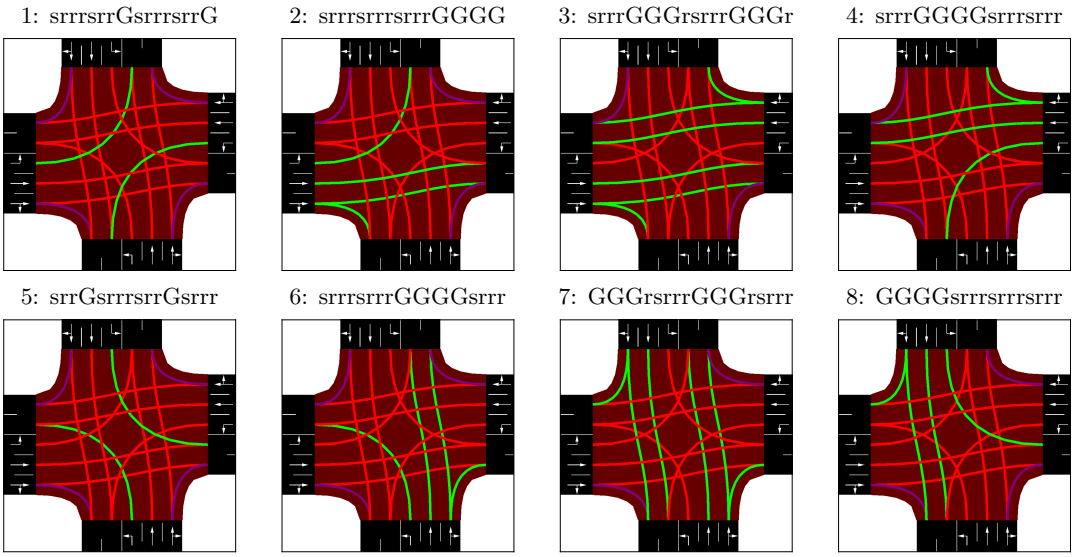

Figure 2: The phase definitions of the intersection [24].

A yellow time between consecutive green phases is implemented. This entails turning all green links yellow for a specified time before switching to the next phase. In order to prevent the agent from making rapid changes between phases a minimum green time is implemented where the agent is forced to wait a predefined duration before being able to make another phase change. The parameters used for the simulation are shown in Table 3.

4 Methodology

One goal of this study is to develop a policy for dynamically controlling the traffic light that optimises flow at the intersection described in Section 3 . To control the traffic light optimally requires efficient decision making regarding phase changes. This requires careful consideration of how the problem should be structured in order to apply RL techniques such as Deep Q-learning and PPO.

4.1 Reinforcement learning approach

In order to apply RL, the traffic light optimisation problem should be framed as a *Markov decision process* (MDP). The MDP can be referred to as the environment, where the state of the environment consists of the current traffic conditions and phase of the traffic light. Furthermore an agent can interact with the MDP by performing actions in the environment. In this case, the set of actions available is phase changes of the intersection (as detailed in Section 3.3). By taking actions which change the phase of the traffic light the agent is able to directly affect the traffic flow. After an action (phase change) has been implemented a reward

is returned to the agent. The reward signal can be set up based on the exact objectives of the intersection and serves as indication of how well the agent is performing. The goal of the agent is to accumulate the largest possible return (sum of rewards) over time. Figure 3 illustrates the interaction between the agent and the environment. In this case the environment is a SUMO simulation wrapped in a Gym wrapper. The wrapper allows the agent to treat the environment like any other Gym RL environment, enabling quick changes to both agent and intersection.

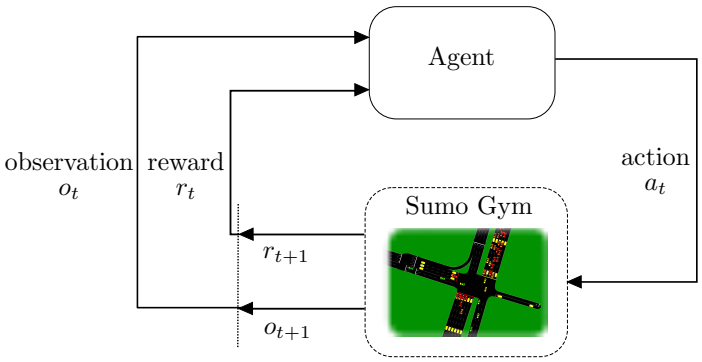

Figure 3: The interaction between the agent and Sumo Gym environment. At each time step the agent receives a reward r_t and observation o_t. The agent performs an action a_t in the environment based on the observation received. In return the environment returns a new observation o_{t+1} and reward r_{t+1}. The interaction between the agent and environment is recurrent.

With the problem framed as an MDP, different RL techniques can be used to solve the problem of finding an optimal decision path through the environment. Q-learning is a popular value-based RL method that learns the value function of an optimal policy. By recurrently alternating between estimating the Q-values of the current policy and improving the policy by acting greedily with respect to the estimated Q-values, the agent indirectly moves towards an optimal policy. The greedy policy can be followed by choosing the action with the highest Q-value in each state. Q-learning is classified as a value-based method since it requires a value function to be estimated.

In contrast to Q-learning, *policy gradient* (PG) methods aim to improve upon a policy directly. Instead of predicting the value of an action, the agent directly predicts the action(s) that will yield the most return (the sum of the rewards). Schulman et al. [16] state that improvements are made to the policy by updating the policy in the direction of a calculated gradient

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla_\theta log \pi_\theta(a_t|s_t)\hat{A}_t]. \tag{2}$$

The advantage $\hat{A}_t$ is an indication of how much better the true value of being in a state s_t given the current policy π_θ is compared to an estimated value. If $\hat{A}_t$ is positive it means the value was underestimated and the probability of taking that action in the policy is increased.

Making large steps based on these calculated gradients can lead to the policy becoming unstable and diverging. To mitigate against this, PPO aims to keep deviations from the previous policy small by clipping them to ensure stability. Since the updated policy will not differ too significantly from the previous policy there will be lower variance when training.

4.2 Agent observations

The observations returned by the environment have a significant effect on the agent's performance. This is because the agent's actions are based on the observations it receives. For example if the environment is only partially observable, it cannot be classified as an MDP and presents new challenges to the agent.

The environment used in this study can be easily modified by using wrapper functions, by allowing for the information returned by the environment to be adapted to suit the application's specific requirements. In our experiments we made use of an observation wrapper which returned a vector of queue lengths for each lane [24]. The queue lengths are determined by the number of halting vehicles. SUMO defines a halting vehicle as a vehicle moving at a speed ≤ 0.1 m.s^{-1} [9]. After the wrapper has been applied the observations returned to the agent take the following form

$$\boldsymbol{o}_t = \left[q_t^1 q_t^2 \cdots q_t^{12}\right]^T, \tag{3}$$

where q_t^i is the queue length (or number of halting vehicles) in lane i at time step t. Furthermore, the cardinality of the observation space is $|\boldsymbol{o}_t| = 12$, since the intersection in Figure 1 has 12 incoming lanes.

4.3 Reward function

The reward function is a crucial component when solving an RL problem. It specifies the metric that the RL agent will strive to maximise in the long run. It is therefore critical to design the reward function to align with the goal the designer wants the agent to achieve. Sutton et al. advise against implicitly encoding domain knowledge into the reward function, as this may cause the agent to learn behaviour that results in a large reward without achieving the desired goal [19].

For this reason, we kept the reward function as simple as possible. Our reward function gave the agent a penalty (negative reward) for the aggregate of the waiting time of vehicles at the intersection [4]. The reward at a time step can be calculated using

$$R_t = -\sum_i^n w_t^i, \tag{4}$$

where n is the number of vehicles halting at the traffic light and w_t^i is the waiting time of vehicle i at time step t. The waiting time of a vehicle only starts when it begins to halt i.e. when its speed is below 0.1 m.s^{-1}. Since a vehicle's waiting time is correlated with delay, by maximising this reward we direct the agent to learn how to reduce delays.

4.4 DQN implementation

Since traffic light optimisation is complex, the state space of the environment becomes extremely large and calculating a value for every possible action-value pair becomes infeasible [24]. To account for this an ANN can be used to approximate the action-value function resulting in a DQN. When using an ANN in conjunction with the off p olicy Q -learning a lgorithm, some instabilities occur. RL works in a sequential manner, therefore observations are highly correlated. This conflicts w ith t raditional s upervised l earning w here d ata s amples a re *independent and identically distributed* (IID). In order to address this issue an *experience replay* (ER) buffer

is utilised. As the agent steps through the environment it stores its experience in the replay buffer. The agent then samples random mini batches from the buffer to train the DQN. The second problem is that the targets used to update the DQN are not stationary. This is because the DQN self is used to compute its targets. As a result, as the DQN's weights change, so do the new target values. To address this, a frozen copy of the DQN is created, known as the target network. The target network is now rather used to compute the agent's targets. The parameters of the target network are held constant and only updated to the new parameters after a set period [11]. This further helps to stabilise training.

The ANN was set up as follows, the network was made up of two hidden layers with 64 neurons in the first layer and 32 in the second. The activation function used was *rectified linear unit* (ReLU) after each hidden layer. The networks were trained using an *adaptive moment estimation* (Adam) optimiser, an extension of *stochastic gradient descent* (SGD), which has been shown to work well for problems which are noisy [7]. The hyperparameters used can be found in Table 1.

Hyperparameter	**Value**
Replay memory size (M)	10000
Mini-batch Size (B)	128
Starting (ε)	0.9
Ending (ε)	0.05
Target network update interval (ΔT)	1800
Discount factor (γ)	0.999
Learning rate (α)	0.01

Table 1: Hyperparameters used for DQN

4.5 PPO implementation

To adapt PPO to the complex environment, two separate ANNs where created. The first ANN was used to capture the policy by outputting a probability distribution over the agents' actions. The second network was used to estimate the state action value function. The same architecture was used for both of the networks. The networks were setup with an input layer with the number of neurons equal to the number of lanes in the observation, two hidden layers containing 128 neurons and an output layer where each neuron represents a possible action. The hyperparameters used are outlined in Table 2.

Hyperparameter	**Value**
Clipping parameter (ϵ)	0.2
Mini-batch size (B)	32
GAE parameter (λ)	0.95
Num epochs	4
Horizon (T)	128
Learning rate actor (α_a)	2.5×10^{-4}
Learning rate critic (α_c)	2.5×10^{-5}

Table 2: Hyperparameters used for PPO

5 Experiments and results

In this section we evaluate the performance of the PPO and DQN agents across the four traffic patterns defined by Wei et al. [24]. Furthermore, we compare the performance of the RL approaches discussed above to three benchmarks: gap-based actuated, delay-based actuated, and Webster. The parameters for the gap-based and delay-based traffic lights are set to the default values found in SUMO, as shown in Table 7 and Table 8, respectively. The configuration for the Webster traffic light using the parameters in Table 5 results in the timings shown in Table 6.

The gap-based actuated traffic light is implemented by SUMO [9]. This method entails to prolong the green time of any phase whenever a continuous stream of traffic is detected. The gap-based implementation of SUMO also supports dynamic phase selection. This entails assigning priorities to different phases of the traffic light depending on various factors. The phase with the highest priority is then selected as the next phase of the traffic light.

The delay-based actuated traffic light prolongs the current green phase if there are vehicles with accumulated time loss. A vehicle's time loss begins as soon as it enters the detector range. If the accumulated time loss exceeds the minimum time loss value, the corresponding green phase is requested to be extended if it is active. The instantaneous time loss of a vehicle is defined as

$$1 - \frac{v}{v_{\max}}, \tag{5}$$

where v is its current velocity and $v_{\max}$ the allowed maximal velocity.

We trained each RL agent for 200 episodes on each of the four traffic distributions P1 through P4 (see Table 4). Each episode consists of 1800 simulation time steps, where each time step is equal to one real-time second. All experiments are seeded with the same seed to ensure that all agents are presented with the same traffic and initial policies. After training, we evaluate the eight trained agents (4 PPO, 4 DQN) on each of the four traffic distributions using a different seed than used during training. This results in 32 total agents being evaluated (16 PPO, 16 DQN).

For the benchmarks, there is no training involved and as such we only evaluate them once using each of the four traffic distribution resulting in 12 benchmarks - 4 actuated, 4 delayed and 4 Webster.

SUMO, [9], offers a plethora of output statics which we generated at the end of each testing episode in order to gain an understanding into the workings and efficiencies of the benchmarks versus the DQN and PPO RL agents.

5.1 Performance during training

We present three metrics: vehicle waiting time, mean queue length and mean speed to evaluate the performance of each agent over consecutive training episodes. These metrics are defined by SUMO [9], as:

- Waiting time: The time in seconds which the vehicle speed was below or equal 0.1 m.s^{-1} (scheduled stops do not count).
- Speed: The mean speed of all vehicles in the network in m.s^{-1}.
- Queue length: The mean queue length of all lanes in meters.

To ensure that our agents do indeed converge, we ran the training process 10 times, each time seeding the traffic and agents with a different seed. The results is the learning curves of in

Figure 4, Figure 5 and Figure 6, with the line showing the mean across the 10 runs, and the shaded area indicates a 95% confidence interval.

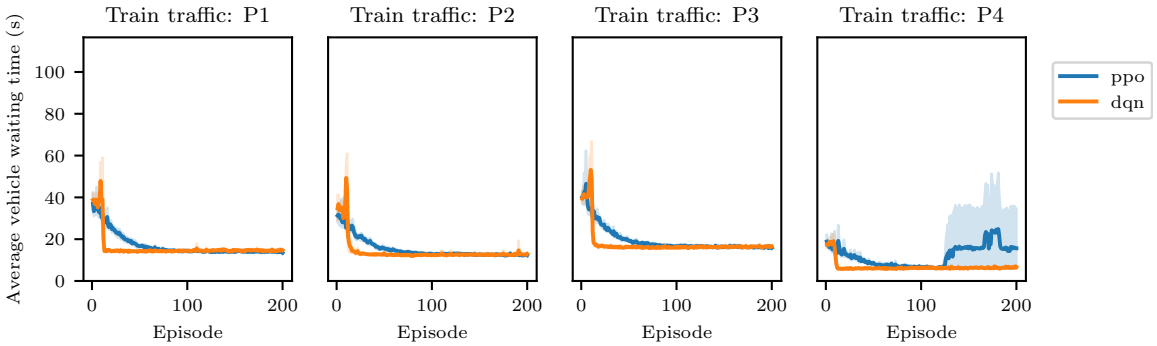

Figure 4: The average vehicle waiting time of DQN compared to PPO over training episodes. Since the goal was to minimise waiting time, this also serves as a proxy for the learning curve.

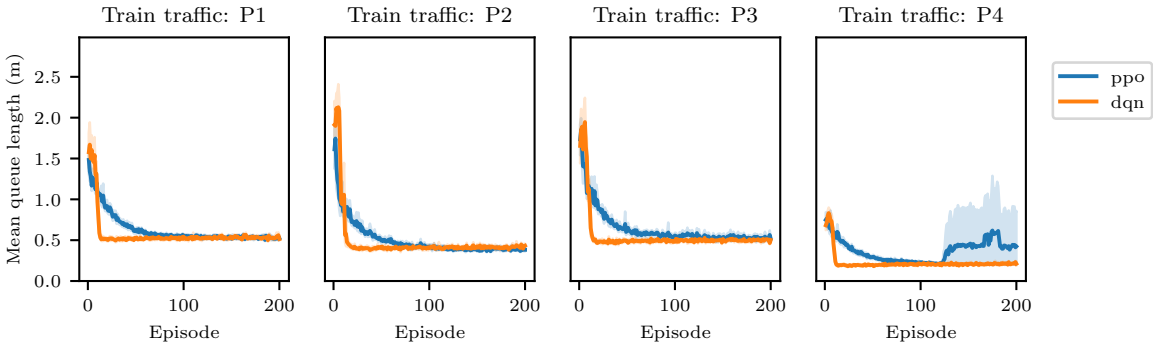

Figure 5: The mean queue length of DQN compared to PPO over training episodes.

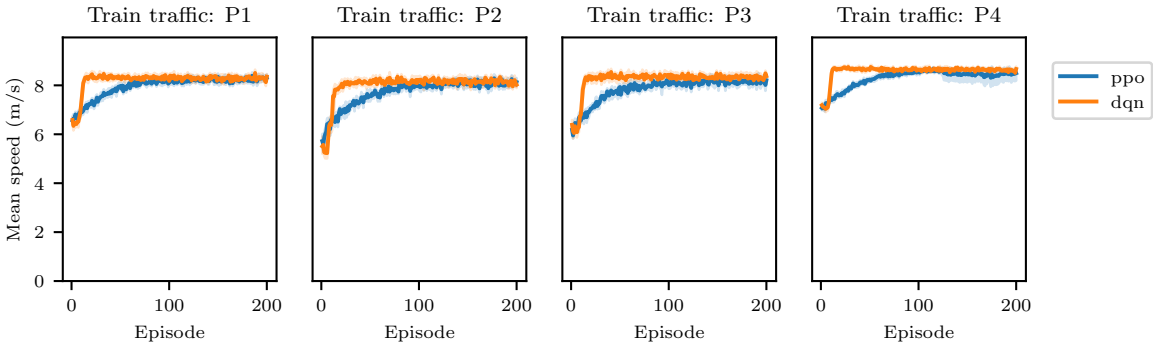

Figure 6: The mean vehicle speed of DQN compared to PPO over training episodes.

For all three metrics it is clear that the DQN agent convergences much faster to a stable solution compared to the PPO agent. Further, the PPO agent seems to have a few runs that become unstable after the 130 episodes mark on traffic pattern P4. Why this is we leave for future work and for the remainder of this paper we use both a DQN and PPO agent that remained stable during the entire training process.

In addition to policy performance, we measured the computational cost of training the various agents. We present similar figures to those shown above, but this time they are plotted over relative training time. All agents were trained on AWS using ml.m5.large instances. As illustrated in Figure 7, both the DQN and PPO agents learn a high-performing policy in a matter of minutes.

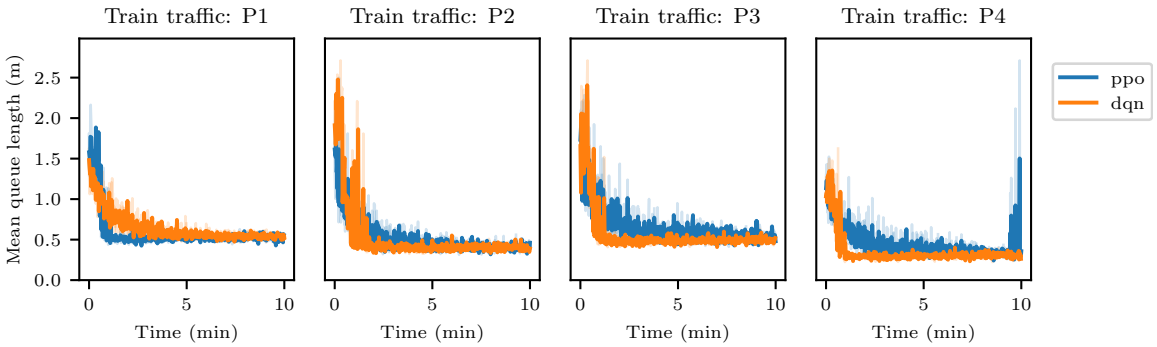

Figure 7: The mean queue length of DQN compared to PPO over relative training time. Both agents converge to optimal policies in a few minutes of training.

5.2 Performance comparisons

To evaluate the two RL agents against the three benchmark agents, we zoom in on the distribution of the waiting time, speed and queue lengths. Using SUMO's output functionality, we are able to export the waiting time and speed for each vehicle and the queue length for each edge at each second during the simulation.

5.2.1 Waiting time at the intersection

Figure 8 shows the distribution of speed for each of the three benchmarks. From the figure it is clear that the Webster algorithm performs the worst of the three benchmarks against all four test traffic patterns which is expected as it is unable to adapt to any dynamic traffic patterns. The delay-based method outperforms the gap-based method on all the traffic patterns except P4. This can be explained by the varying nature of the P4 traffic which the actuated method can handle effectively in a manner which reduces the tail (max waiting) of the distribution.

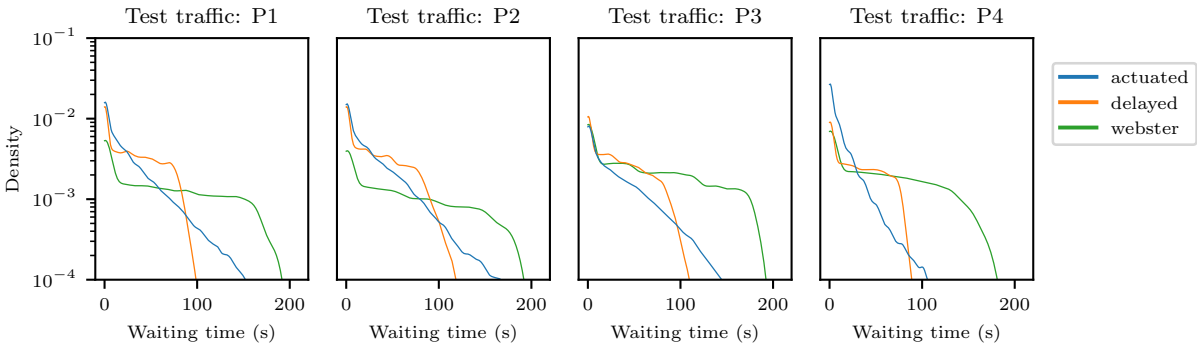

Figure 8: Vehicle waiting time distribution for 25 episodes across 4 traffic distributions for 3 benchmarks: Webster, delay and actuated.

Figure 9 shows the waiting time distribution of the DQN RL agent for all 16 combination of train-test traffic pairs. The waiting time distribution appears to be invariant to the traffic which the DQN agent is trained on, except for a slight benefit when training and testing on the same traffic for P3 and P4 respectively. This is expected for the more varying traffic as the agent has seen both regimes of traffic during training.

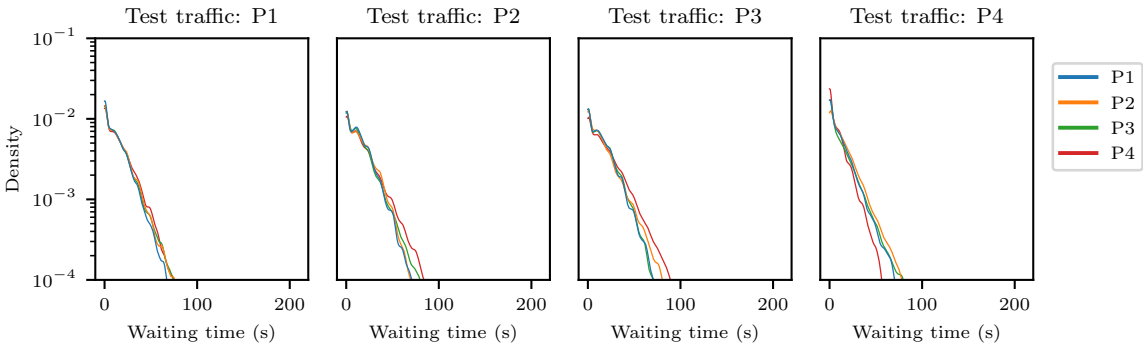

Figure 9: Vehicle waiting time distribution for 25 episodes for 4 test traffic patterns and 4 train traffic patterns for DQN.

Figure 10 shows the waiting time distribution of the PPO RL agent and tells a similar story to the DQN RL agent with the agents tested on the same network they were trained on performing slightly better compared to the other trained traffic patterns, but overall, all agents perform well and no cars have a waiting time greater than 100 seconds.

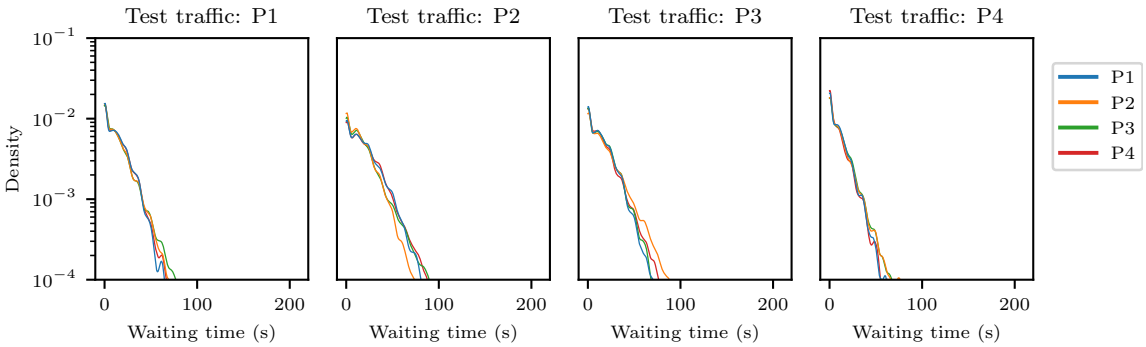

Figure 10: Vehicle waiting time distribution for 25 episodes for 4 test traffic patterns and 4 train traffic patterns for PPO.

Figure 11 compares the 3 benchmarks to the DQN and PPO agents trained and tested on the same traffic pattern. From the graph it is apparent the benefit of using the RL agents. They are adapting to the traffic demands, bringing the tail end of the waiting time distribution in to around 50 seconds. In other words, if the RL agents are used, there are no cars waiting at the intersection for longer than 50 seconds. According to Figure 11, the RL agents perform very similarly.

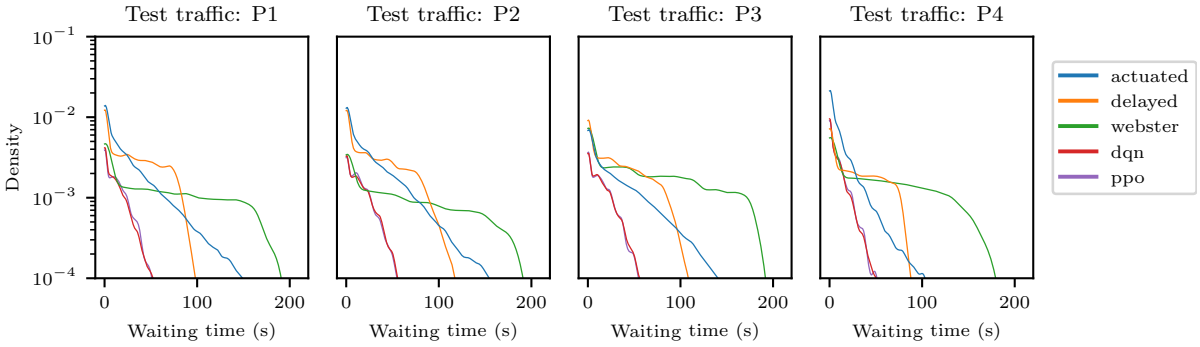

Figure 11: Vehicle waiting time distribution for 25 episodes across four test traffic patterns comparing benchmarks with the two RL agents.

The speed and queue lengths distributions show similar results to the waiting time distribution with the RL agents outperforming all the benchmarks. Figure 12 and Figure 13 show the two best RL agents (trained and tested on the same traffic) against the three benchmarks.

Figure 12 shows that for the benchmark agents there are more vehicles reaching higher speeds compared to the RL agents. However, there is a significant amount of cars that have speed less than 3 m.s^{-1}, whereas the RL agents' speed distributions are more uniformly distributed serving all cars more equally.

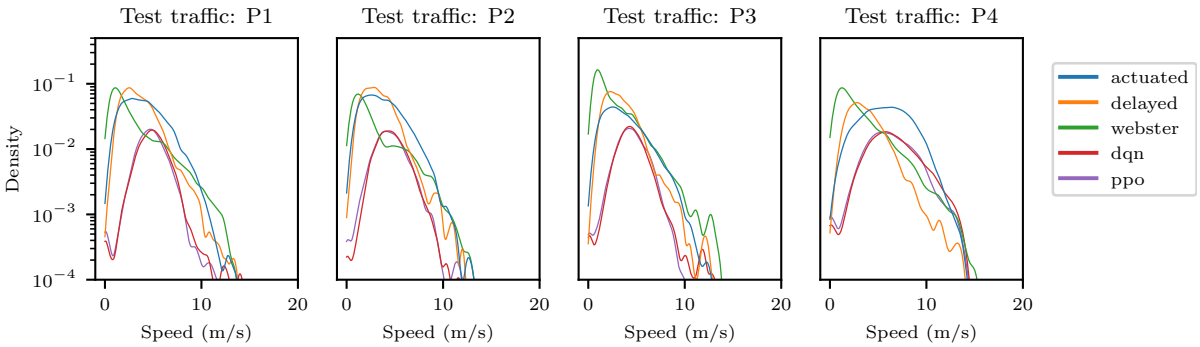

Figure 12: Vehicle speed distribution for 25 episodes across 4 test traffic patterns comparing benchmarks with RL agents.

Figure 13 provides another lens on the performance of the RL agents, showing the distribution of queue lengths for the three benchmarks and the two RL agents (trained and tested on the same traffic). From the graph it is clear that the RL agents are outperforming the benchmark agents as the RL agents are able to clear the lanes well before they fill up to their maximal capacity of 150m.

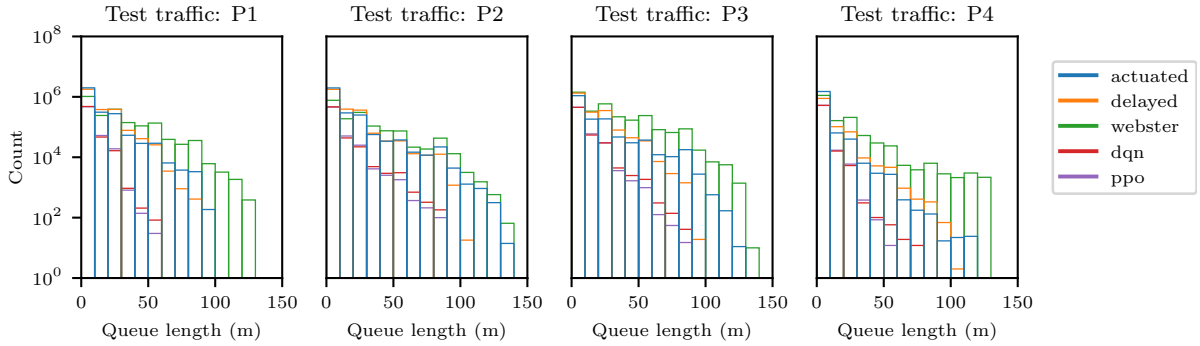

Figure 13: Lane queue length distribution for 25 episodes across four test traffic patterns comparing benchmarks to RL agents.

5.3 Trained policies

In this section we attempt to analyse the policy each of the benchmark and RL agents followed when being presented with a particular traffic pattern. In the case of the RL agents, how the policies are affected when trained on the various traffic patterns P1 through P4. We gain insights into the trained policies by looking at two different metrics namely the percentage time spent in a phase over the 25 episodes and the number of times an agent switched from one phase to another over the 25 episodes (phase transition matrix).

5.3.1 Percentage time spent in a phase

In order to condense the eight dimensional action space of the agents into a graph we us polar plots shown in Figure 15 with their legend shown in Figure 14. The polar plots compare the five agents against each other using the defined polar plot legend. The percentage time an agent spends in the different phases during testing determines the shape of the polygon. The further the vertex is from the centre, the longer the agent spent in that respective phase. This allows us to see insights such as if certain phases were favoured or ignored in the policy.

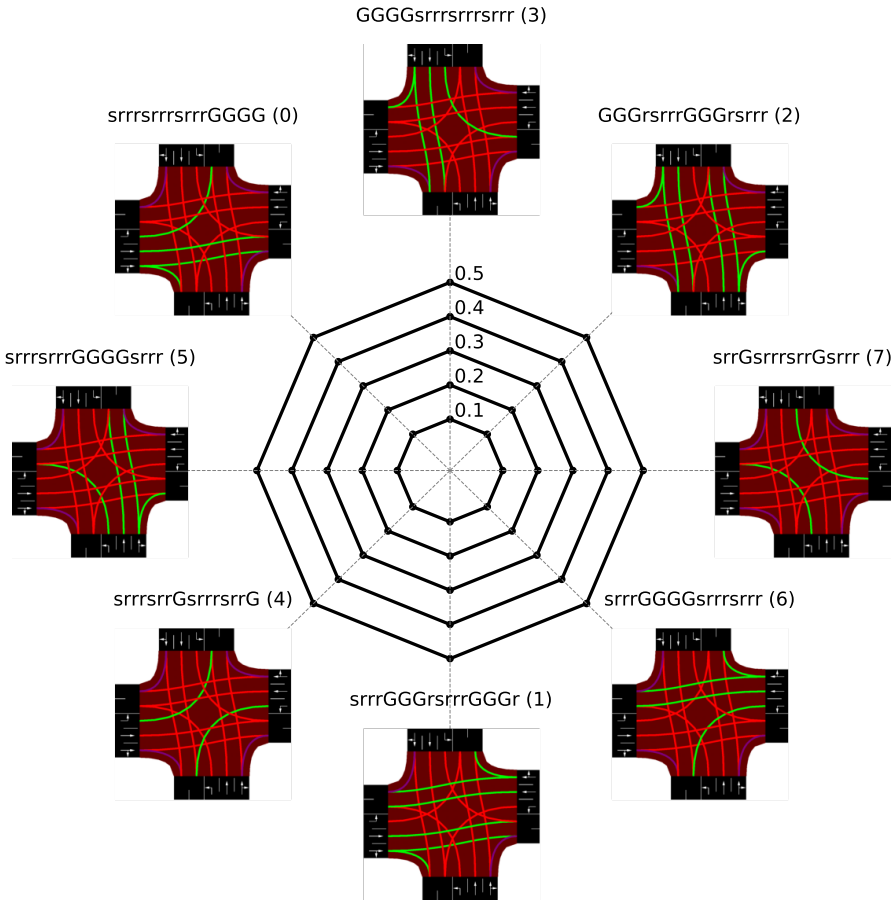

Figure 14: Policy polar plot legend.

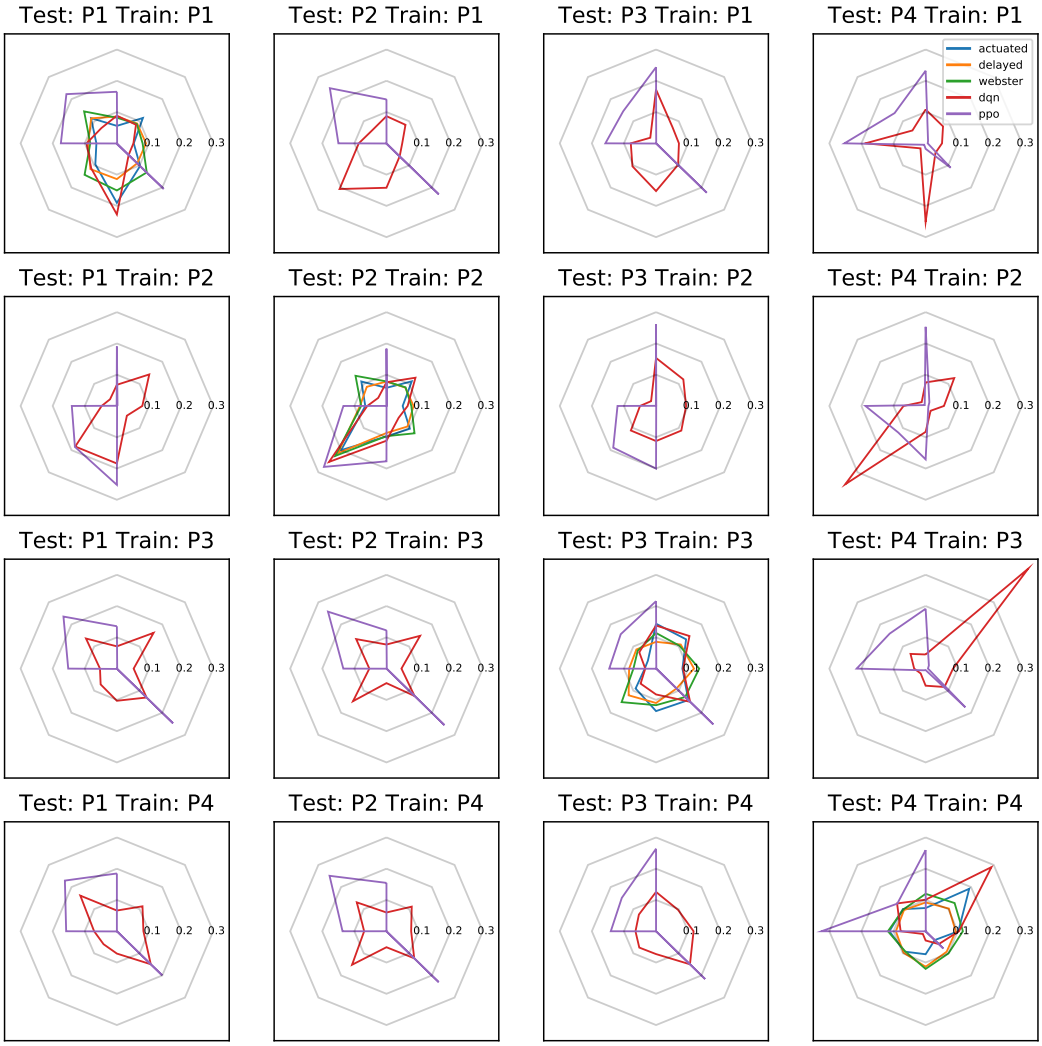

Figure 15: Percentage time spent in each state per agent per test-train traffic over 25 testing episodes.

From the emergent patterns in Figure 15 it is clear that the DQN and PPO agents have very different policies when it comes to optimising a given traffic set. This is evident by the different shaped polygons indicating the different policies favoured different states. A few note worthy patterns from Figure 15 are:

- The bird-like pattern the DQN policy has when trained on traffic P3 and P4 and tested on traffic P2.
- The kite-like pattern the PPO policy has when trained on traffic P1, P3 and P4 and tested on traffic P2, as well as when trained on traffic P3 and P4 and tested on traffic P1.

- The similarities in policies between the DQN agent and the benchmarks when the test and train traffic is P2.

All three of these cases are tested on the P2 traffic pattern which includes higher demand turning traffic. This highlights that increased turning traffic demands creates a significant change in the policy.

The bird-like pattern the DQN has when trained on traffic P3 and P4 and tested on traffic P2 can be explained by looking at the four phases the DQN agent visited most, namely: 0, 2, 4, and 6. Referencing Figure 14 we can see that these four phases allow the agent to switch for traffic on the main N-S route (phase 2), the E-W route (phase 6), the W-E route (phase 0) as well as having a dedicated turning phase (phase 4) for the E-W/W-E routes. Objectively, the bird-like policy of the DQN makes sense, as favour using these phases should allow an agent to address most traffic patterns.

The bird-like policy of the DQN is in stark contrast to the kite-like policy the PPO agent learnt. If we take a closer look at which phases the PPO agent favoured: 0, 3, 5 and 6, we can see that two of these overlap. Both agents require phase 0 and 6 to clear the E-W/W-E traffic. However, the difference between the PPO and DQN agents is evident in the way which they clear the N-S/S-N traffic, the PPO instead utilises phase 3 and 5 instead of phase 2 the DQN uses. Again, considering that both these RL agents performed equally well for our metrics and outperformed the benchmarks, it is clear that there are multiple optimum policies for a given intersection and traffic pattern.

Lastly, looking at the similarities between the DQN agent and benchmark agents when the train and test traffic both equal P2, we can see that the RL agents estimate the benchmark's policies under certain conditions. However, the DQN agent is able to put less of a dependency on phase 0 and 6 and clear the turning traffic demand more efficiently using phase 4. Whereas, the PPO agent resorts to using phase 3 and 5 for the N-S/S-N demand and in contrast to the DQN and benchmarks, uses phase 1 for the E-W/W-E demand instead of phase 0 and 6.

The waiting time, speed and queue length graphs from Section 5.2 showed comparable results across all policies. It is interesting to see that similar results can be achieved with such varied policies. It also highlights the significance of using such visualisations to evaluate the policies.

5.4 Phase transition matrices

The policy polar plots introduced in the previous subsection, are a useful insight into the policies in terms of where the agents spent their time. To gain further intuition into the policies we look into how the policies transition between the different phases. This can be visualised in Figures 16 through 20. These figures show the number of times each agent moved from one phase (on the x-axes) to another (y-axes) during the 25 episodes during testing.

We begin by introducing our baseline algorithms which are used as comparison for the RL agents. The phase transition matrices are shown for the three baseline methods namely Webster, delay-based and vehicle-actuated. The matrices can be visualised in Figures 16, 17 and 18 respectively.

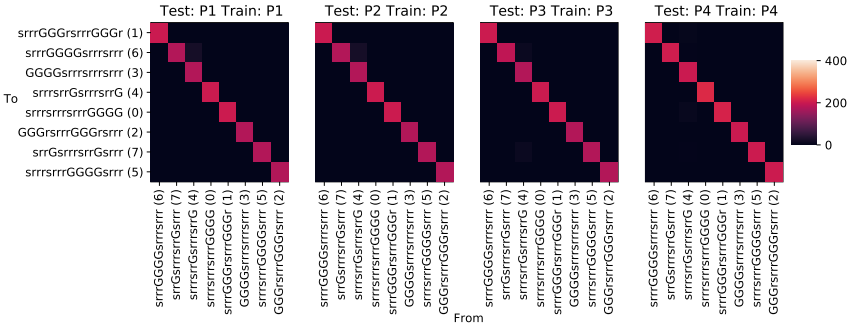

Figure 16: Number of times the traffic light switch from one phase (x-axis) to another (y-axis) over 25 episodes for the Webster algorithm.

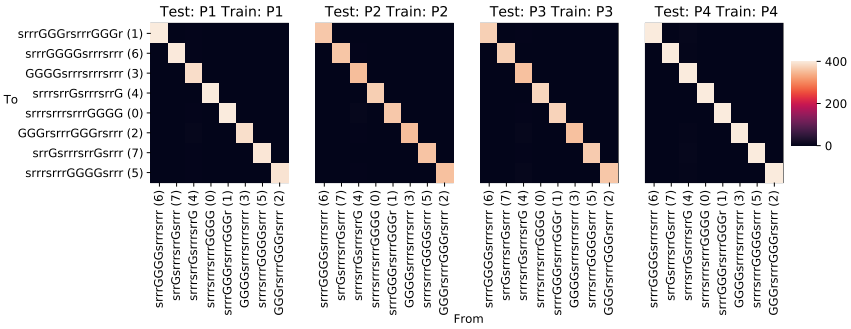

Figure 17: Number of times the traffic light switch from one phase (x-axis) to another (y-axis) over 25 episodes for the delay-based algorithm.

The first thing to note is the is the diagonals formed for the Webster (Figure 16) and delay-based (Figure 17) agents. These diagonals form as both these agents have a fixed sequence which they are allowed to use, so each phase is always followed by a particular other phase.

The delay-based method uses shorter minimum green times which can be extended dynamically as traffic patterns change. Since the Webster method has static phase timings not allowing for such extensions it's green time is set to a longer time period. This difference is illustrated in Figure 17 where the delay-based benchmark has higher counts (demonstrated by the brighter squares) than the Webster benchmark.

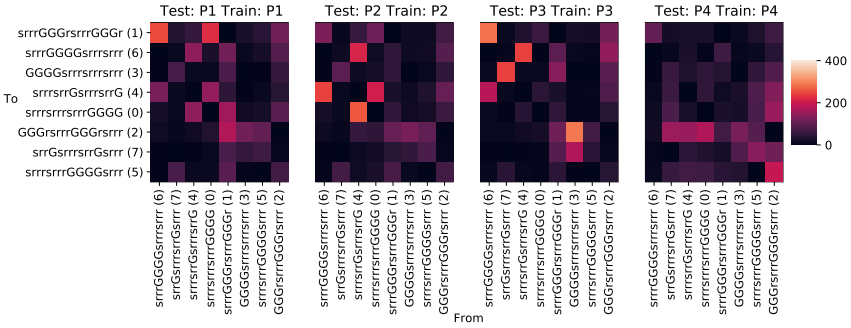

Figure 18: Number of times the traffic light switch from one phase (x-axis) to another (y-axis) over 25 episodes for the gap-based actuated algorithm.

The phase transition matrix for the gap-based actuated benchmark shown in Figure 18 shows that this benchmark is indeed able to dynamically select the next phase and in our configuration we've allowed it to choose any phase. There is little similarity between the the phase transition matrices of the gap-based actuated benchmark and the phase transition matrices of the DQN and PPO agents. This implies that the RL agents learned their own policies and did not learn a strategy similar to the conventional gap-based benchmark.

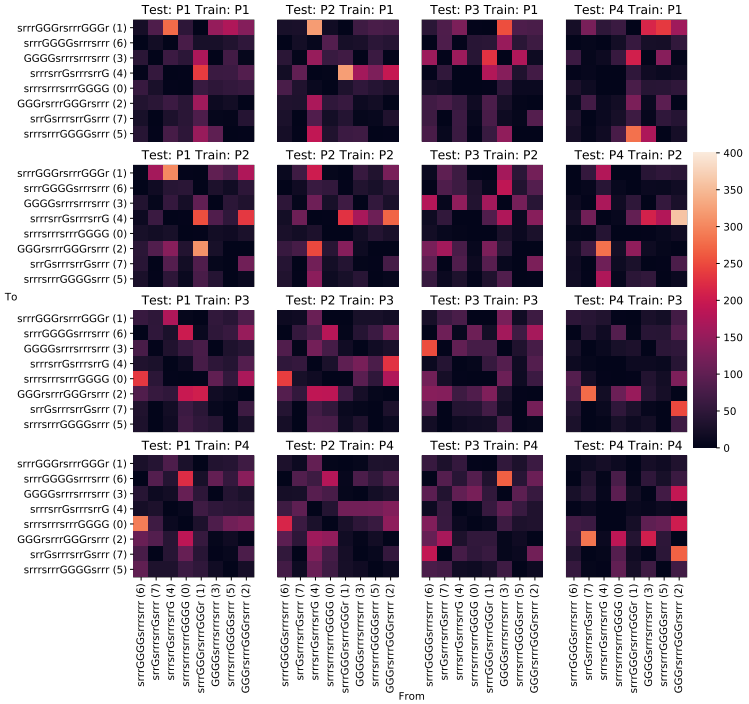

Figure 19: Number of times the traffic light switch from one phase (x-axis) to another (y-axis) over 25 episodes for the DQN agent.

Finally, we compare the phase transition matrices of the DQN and PPO agents, shown in Figure 19 and Figure 20 respectively. We can see that these agents approach the task at hand in vastly different ways. The DQN agent is much more "soft" with its transition, utilising all phases, whereas the PPO agent transitions between a subset of all phases and completely ignores some phases. This comes as a surprise as the PPO agent has the more probabilistic architecture, sampling from its sample space instead of making a hard prediction like the DQN. However, it appears from the phase transition matrices that these sampling distributions of the PPO quickly converge with very low variability around the sampling distribution.

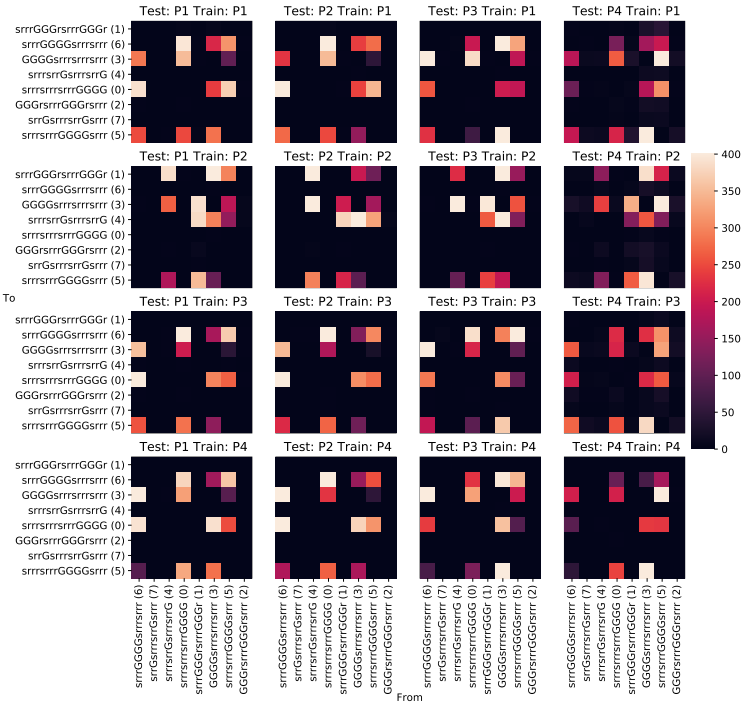

Figure 20: Number of times the traffic light switch from one phase (x-axis) to another (y-axis) over 25 episodes for the PPO agent.

It is also worth noting the patterns that emerge from the PPO's phase transition matrices across trained traffic patterns (same rows have same patterns). They suggest that the PPO agents overfit on the traffic pattern it has seen in training and struggles to generalise well to unseen traffic, only utilising the transitions it learnt during training. Whereas the DQN agent appears to generalise better across both training and testing, not favouring any transitions, but rather using all transitions to achieve the desired phase. For example when agents trained on traffic pattern P2 (which included a larger volume of left turning traffic) were tested on other traffic patterns, the agents tended to favour phases which allowed for the flow of left turning traffic. Even when tested on other traffic patterns, this agent is trained to optimise flow for this specific traffic pattern and tends to favour the phases that serve the left turning vehicles. The policies learnt by these agents approach the problem in vastly different ways, and when considering a real-world implementation of these agents, additional work would be required to ensure that the policies these agents learn transfer to the real world.

6 Conclusion

The aim of this paper was to provide an in depth comparison between Deep Q-learning and *proximal policy optimisation* (PPO) on a traffic light optimisation problem. Both the PPO and DQN methods drastically improved upon traditionally implemented methods namely, gap-based actuated, delay-based actuated and a Webster fixed-time approach. On a high level comparison of the agents, we see performance improve as they maximise their return during training.

Training time and computational efficiency can be a concern, since *deep reinforcement learning* (DRL) is known to require large amounts of data and many training iterations to converge to optimal policies. The experiments conducted in this study show that training times for this specific intersection are extremely short. In future research, it will be interesting to see how training times increase with more complex networks and intersections. Furthermore, no dataset is required to train an RL agent because the data required for training is generated by the simulation environment (SUMO). However, the simulation must be accurately parameterised, which usually necessitates the use of real-world data.

In addition to demonstrating that the policies are efficiently and effectively trained, the goal was to gain a deeper understanding of the strategies learned through the use of the various methodologies, namely PPO and DQN. To visualise the difference between the agents' policies, we introduce several plots, including the polar policy plot and phase transition matrices. These visualisations suggest that the strategies learned by these agents differ significantly. PPO agents typically use only a subset of all phases, whereas DQN agents spread their policy across all phases.

Another finding was that the agents tend to overfit on the traffic used to train a policy. An agent who has been trained on traffic with a high volume of vehicles turning left learns a policy that accommodates this specific traffic flow. This training artefact can be seen when the agent is tested on other traffic patterns, as discussed in the previous section. A future research avenue could be to train a more general policy that mitigates these training artefacts.

Despite the agents overfitting, this study shows the promise about using RL as an approach to traffic light optimisation. The agents performed well on the traffic they are trained on showing their ability to learn different types of traffic distributions and with a more encompassing training set can become a robust and efficient solution.

References

[1] Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 129(3):278–285, 2003.

[2] Richard E Allsop. Delay-minimizing settings for fixed-time traffic signals at a single road junction. *IMA Journal of Applied Mathematics*, 8(2):164–185, 1971.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Alvaro Cabrejas Egea and Colm Connaughton. Assessment of reward functions in reinforcement learning for multi-modal urban traffic control under real-world limitations. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2095–2102. IEEE, 2021.

[5] Mengyu Guo, Pin Wang, Ching-Yao Chan, and Sid Askary. A reinforcement learning approach for intelligent traffic signal control at urban intersections. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 4242–4247. IEEE, 2019.

[6] PB Hunt, DI Robertson, RD Bretherton, and RI Winton. Scoot-a traffic responsive method of coordinating signals. Technical report, 1981.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[8] Li Li, Yisheng Lv, and Fei-Yue Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 3(3):247–254, 2016.

[9] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[10] PR Lowrie. Scats, sydney co-ordinated adaptive traffic system: A traffic responsive method of controlling urban traffic. 1990.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[12] Sajad Mousavi, Michael Schukat, Peter Corcoran, and Enda Howley. Traffic light control using deep policy-gradient and value-function based reinforcement learning. *IET Intelligent Transport Systems*, 11, 04 2017.

[13] Robert Oertel and Peter Wagner. Delay-time actuated traffic signal control for an isolated intersection. In *Proceedings 90th Annual Meeting Transportation Research Board (TRB)*, 2011.

[14] Syed Shah Sultan Mohiuddin Qadri, Mahmut Ali Gökçe, and Erdinç Öner. State-of-art review of traffic signal control methods: challenges and opportunities. *European transport research review*, 12(1):1–23, 2020.

[15] Sampson, van As, Joubert, Dazeley, Labuschagne, and Swanepoel. South african road traffic signs manual. *Civil Engineering= Siviele Ingenieurswese*, 3(3):101, 2012.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[17] Li Song, Wei Fan, and Pengfei Liu. Exploring the effects of connected and automated vehicles at fixed and actuated signalized intersections with different market penetration rates. *Transportation Planning and Technology*, 44(6):577–593, 2021.

[18] Aleksandar Stevanovic, Cameron Kergaye, and Peter Martin. Scoot and scats: A closer look into their operations. 01 2009.

[19] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[20] Tomer Toledo, Tamir Balasha, and Mahmud Keblawi. Optimization of actuated traffic signal plans using a mesoscopic traffic simulation. *Journal of Transportation Engineering, Part A: Systems*, 146(6):04020041, 2020.

[21] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[22] F.V. Webster. Traffic signal settings. Technical report, 1958.

[23] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. *ACM SIGKDD Explorations Newsletter*, 22(2):12–18, 2021.

[24] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2496–2505, 2018.

A Simulation configuration

Parameter	**Value**
Lane length	150 meters
Vehicle length	5 meters
Minimal gap between vehicles	2.5 meters
Car-following model	Krauss following model [9]
Max vehicle speed	13.42 $m.s^{-1}$
Acceleration ability of vehicles	2.6 $m.s^{-2}$
Deceleration ability of vehicles	4.5 $m.s^{-2}$
Duration of yellow signal	3 seconds
Minimum green time	10 seconds
Number of episodes	200
Episode length	1800 seconds

Table 3: Simulation parameters [24]

Traffic pattern	Direction	Arrival rate		
		Through	Left-turn	Right-turn
P1:Major/Minor road	N-S	0.05	0.025	0.01
	S-N	0.05	0.025	0.01
	E-W	0.1	0.05	0.01
	W-E	0.1	0.05	0.01
P2:Through/left-turn lane	N-S	0.05	0.025	0.01
	S-N	0.05	0.025	0.01
	E-W	0.05	0.1	0.01
	W-E	0.05	0.1	0.01
P3:Tidal traffic	N-S	0.1	0.08	0.01
	S-N	0.05	0.025	0.01
	E-W	0.1	0.08	0.01
	W-E	0.05	0.025	0.01
P4:Varying demand traffic	N-S	0.05	0.025	0.01
	S-N	0.05	0.025	0.01
	E-W	0.05 (0-1200 steps) 0.15 (1200-1800 steps)	0.025	0.01
	W-E	0.15 (0-600 steps) 0.05 (600-1800 steps)	0.025	0.01

Table 4: Arrival rates for different traffic patterns as outlined by Wei et al. [24]

B Benchmark configuration

Parameter	**Value**
Lost time	12
Max cycle	240
Min cycle	20
Sat headway	3.2

Table 5: Configuration used for Webster

P1	P2	P3	P4	State
32	50	34	23	srrrsrrGsrrrsrrG (4)
32	30	19	21	srrrsrrrsrrrGGGG (0)
34	22	27	26	srrrGGGrsrrrGGGr (1)
33	30	29	22	srrrGGGGsrrrsrrr (6)
21	21	33	25	srrGsrrrsrrGsrrr (7)
21	21	18	24	srrrsrrrGGGGsrrr (5)
21	21	26	26	GGGrsrrrGGGrsrrr (2)
20	21	29	26	GGGGsrrrsrrrsrrr (3)

Table 6: Webster phase timings for each traffic pattern

Parameter	**Value**
Detector gap	1
Minimum duration (all phases)	10
Maximum duration (all phases)	60
Frequency	300
Max gap	3
Nexts	All phases included current
Passing time	10

Table 7: Configuration for gap-based actuated

Parameter	**Value**
Detector range	100
Minimum duration (all phases)	10
Maximum duration (all phases)	60
Frequency	300
Minimum time loss	1

Table 8: Configuration for time-loss actuated

C Additional results

C.1 Speed

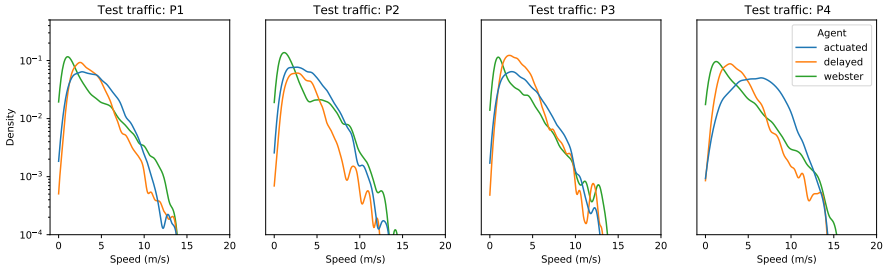

Figure 21: Benchmarks

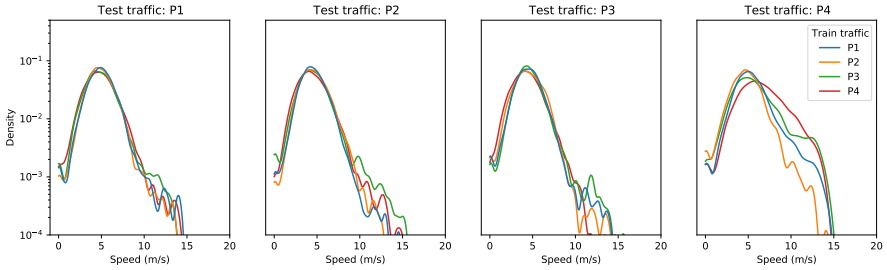

Figure 22: DQN

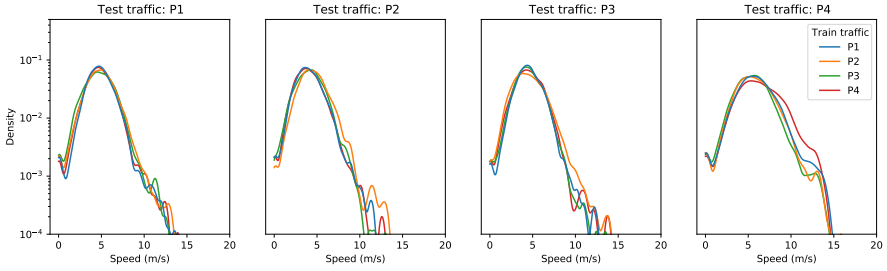

Figure 23: PPO

C.2 Queue lengths

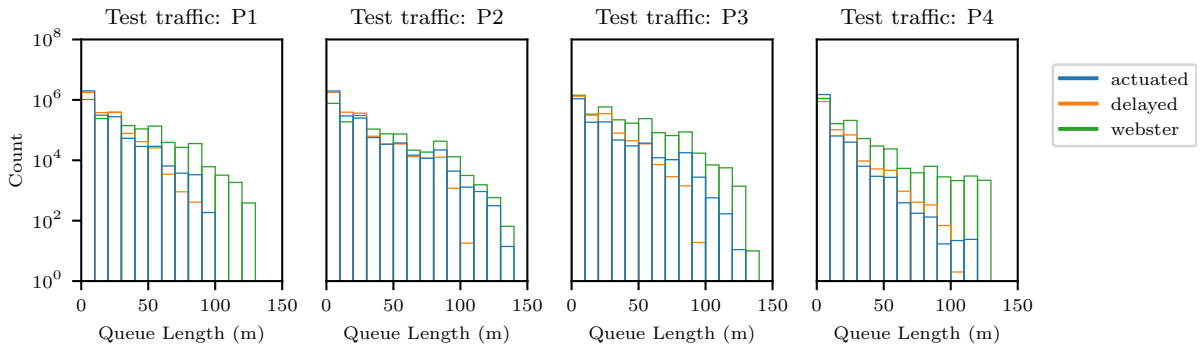

Figure 24: Benchmarks

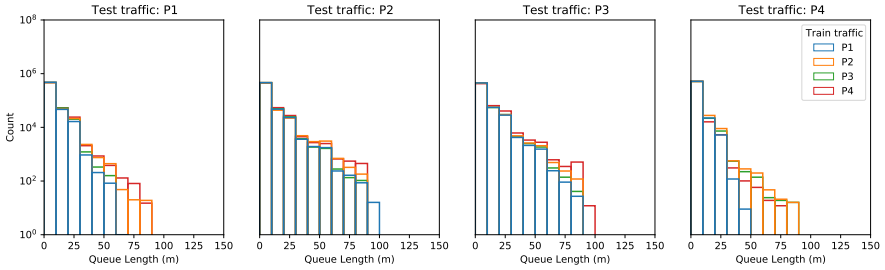

Figure 25: DQN

Figure 26: PPO

C.3 Percentage time spent in each phase

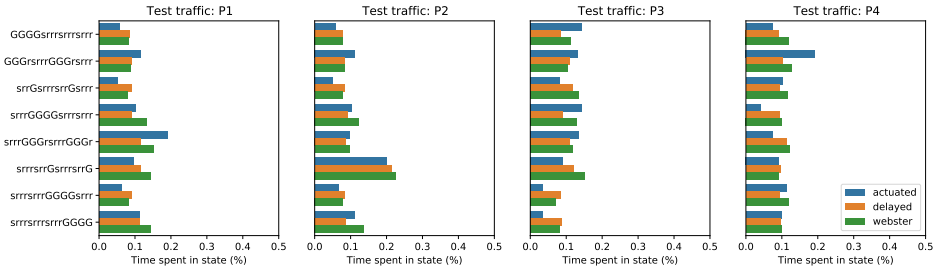

Figure 27: Percentage time spent in phase for the benchmarks agent for each test traffic .

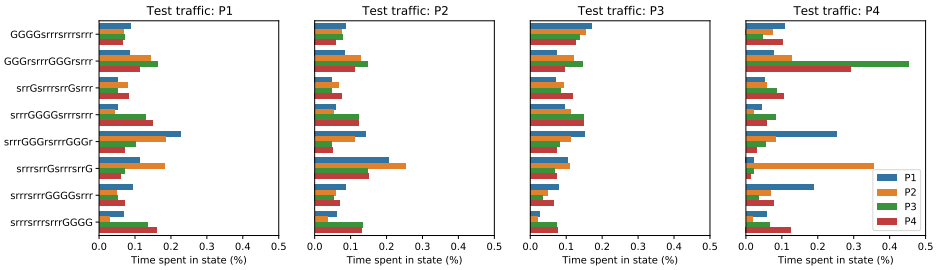

Figure 28: Percentage time spent in phase for the DQN agent for each test train traffic pair.

Figure 29: Percentage time spent in phase for the PPO agent for each test train traffic pair.