university of groningen / faculty of science and engineering

Deep Reinforcement Learning for Traffic Signal Control Optimization

Bachelor's Project Thesis

Bogdan Palfi, s3984451, b.palfi@student.rug.nl,
Supervisor: Dr. Matthia Sabatelli, m.sabatelli@rug.nl

Abstract: The increase in the number of personal vehicles per capita often leads to traffic congestion in highly urbanized areas. A solution to this problem consists of optimizing traffic signal control (TSC) policies using deep reinforcement learning (DRL). For this reason, the current thesis analyzed the TSC performance of two DRL algorithms introduced by Sabatelli et al. (2020), namely Deep Quality-Value (DQV) and Deep Quality-Value-Max (DQV-Max). The two algorithms were compared to classic DRL algorithms such as Deep Q-Network (DQN) and Double Deep Q-Network (DDQN), as well as to heuristic strategies such as Longest-Queue-First or Fixed-Timings. All algorithms were trained and tested in a simulated environment, created using Eclipse SUMO, which involved a single cross intersection with 3 lanes. The DRL agents were tasked with reducing the waiting time at the intersection by selecting which of the lanes were granted a green light. The results showed that all DRL agents achieved lower average waiting times than the heuristic strategies. Overall, the DQN and DQV-Max algorithms were more stable and kept waiting times lower than DDQN and DQV. The latter algorithms presented some instability, indicated by occasional waiting time spikes. While the environment was not complex enough to provide incentives for real-world deployment, the thesis acted as a proof of concept for the DQV and DQV-Max algorithms in the context of TSC.

1 Introduction

The past decades saw a rapid growth in the number of personal vehicles per capita, a tendency especially visible in highly urbanized areas. Because of this, the demand for traffic infrastructure has increased significantly, leading to traffic congestion, which presents serious economic, environmental and public health issues (Samal et al., 2020). A solution to alleviate this problem consists of improving traffic signal control (TSC) policies, which manage how traffic lights change in an intersection. Sub-optimal TSC policies lead to what is known as *green idling*, which occurs when the traffic light for a specific lane is green yet there is no vehicle on that lane (Rasheed et al., 2020). Therefore, an optimized TSC should not allow green idling to occur.

However, manually optimizing TSC policies can be an arduous task due to traffic's stochastic nature and the multitude of possible intersection configurations. To this end, model-free *Reinforcement learning* (RL) has shown promising results in dynamically adapting TSC policies to the needs of specific intersections (Rasheed et al., 2020). The principle behind model-free RL is to learn certain *value functions*, named *state-value* and *state-action* by using a trial-and-error approach when interacting with the environment. The state-value function, denoted as $V(s)$, is employed to estimate how good state s is with respect to the expected return of the RL agent (Sutton et al., 2018). Similarly, the state-action function, denoted as $Q(s, a)$, allows the agent to estimate the expected return when taking action a in state s (Sutton et al., 2018). Therefore, the two functions allow the agent to optimize its behavior in order to increase this expected return, which represents the cumulative reward that the agent receives from its environment. Model-free *Deep Reinforcement Learning* (DRL), the combination between deep learning and model-free reinforcement learning, functions on the same principles by employing artificial neural networks to estimate such value functions. However, the classic DRL algorithms present certain limitations since they only estimate the Q function (Sabatelli et al., 2020). In addition, this Q function is often overestimated, which can lead to sub-optimal policies since

overestimation errors can be propagated through learning (van Hasselt et al., 2015).

For these reasons, two model-free DRL algorithms, *Deep Quality-Value* (DQV) and *Deep Quality-Value-Max* (DQV-Max), which approximate both the V and Q functions, were introduced by Sabatelli et al. (2020). The aim of this thesis is to test the performance of the DQV and DQV-Max algorithms in the context of TSC optimization. The thesis should thus act as a proof of concept for the two algorithms, taking them closer to real-world deployment. To this end, the thesis will first cover the current developments in the field of reinforcement learning (Section 2.1) and deep reinforcement learning (Section 2.3). Afterwards, the environment (Section 3.1) as well as the heuristic strategies (Section 3.2) and the DRL agents (Section 3.3) will be discussed. In addition, an extra employed method, called *Prioritized Experience Replay* (Section 3.5), will be explained, followed by the experimental setup (Section 3.6). Finally, the results (Section 4) along with a discussion (Section 5), covering both limitations and future work will be presented.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) involves an agent which explores and interacts with its *environment* in order to maximize the *reward* it receives. Therefore, at each specific time step denoted as t, the agent can find itself in a finite set of states $\mathcal{S} = \{s^1, s^2, ..., s_n\}$, in which it can perform a finite set of actions $\mathcal{A}$ (Sutton et al., 2018). Performing an action $a_t \in \mathcal{A}(s_t)$ at time step t, while in state $s_t \in \mathcal{S}$ makes the agent transition to state s_{t+1}, according to the transition probability distribution $p(s_{t+1}|s_t, a_t)$. For each transition, the agent also receives a reward r_t, determined by the reward function $\mathcal{R}(s_t, a_t)$. However, since RL tasks are often continuous, an additional term, the *discount factor*, is used. This term is denoted as $\gamma \in [0, 1)$ and controls the current value of future rewards (Sutton et al., 2018).

A RL agent acts according to a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which indicates what action is appropriate in a specific state. With this in mind, the definition of the *state-value* function V^π can be seen in Equation 2.1, while the *state-action* Q^π function is defined in Equation 2.2 (Sutton et al., 2018). The two equations calculate the expected cumulative discounted reward received under policy π, when starting in state s (2.1) or when taking action a in state s (2.2). Therefore, the goal of the RL agent is to optimize its policy so that the two value functions are maximized for all states or state-action pairs.

$$V^\pi(s) = \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big| s_t = s, \pi\Big] \tag{2.1}$$

$$Q^\pi(s, a) = \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big| s_t = s, a_t = a, \pi\Big] \tag{2.2}$$

2.2 Q-Learning

One of the most popular RL algorithms, designed to learn the Q function, is known as *Q-Learning*, defined in Equation 2.3. α denotes the learning rate.

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)\big] \tag{2.3}$$

This algorithm uses Temporal Difference (TD) Learning (Sutton, 1988) since it samples experiences from the environment in order to create future estimations, with which it updates the value of the Q function. More specifically, TD-Learning involves creating a TD-Error δ, defined in Equation 2.4, and using this error to update the Q function. The first two elements of this TD-Error are denoted as the TD-Target, which is defined as the sum between the reward received at the current time step, r_t, and the discounted maximum expected Q value of the next time step, $\gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$. Q-Learning is thus an off-policy algorithm since the estimated Q value of the next time step is greedily chosen using the max operator.

$$\delta = r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \tag{2.4}$$

2.3 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is an improvement over traditional RL techniques since it employs artificial neural networks, considered universal function approximators (Hornik et al., 1989), to approximate value functions. The main advantage of this approach is that it allows for much larger state-action spaces than classic, tabular RL methods do. The most well known DRL algorithm is called Deep Q-Network (DQN) and was first introduced as a solution to the high dimensional inputs of Atari games (Mnih et al., 2013). The original DQN algorithm employed a convolutional neural network, specialized in analyzing visual imagery, in order to approximate the Q function. This was accomplished by turning the TD-Error from Q-Learning into a differentiable loss function (see Equation 2.5) which the neural network, denoted as θ, must minimise (Mnih et al., 2013). However, the addition of neural networks also introduced more instability in the algorithm. For this reason, the loss function employed two extra methods, named Target-Network and Experience-Replay, which increase stability during training.

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\Big[\big(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)\big)^2 \Big] \quad (2.5)$$

Target Network This method involves using a second neural network, denoted as the target network θ^-, to calculate the TD-Target (Mnih et al., 2013). This target network has the same architecture as the main online network θ but a different update frequency. While the online network is updated at every training step, the target network's weights are frozen and the update is done periodically by completely copying the weights from the online network.

Experience-Replay The second method involves storing the agent's experiences in a memory buffer and then creating training batches by uniformly sampling from this buffer (Lin, 1992; Mnih et al., 2013). The memory buffer functions as a queue and the experiences are stored as tuples $\langle s_t, a_t, r_t, s_{t+1}\rangle$. This method breaks the autocorrelation between consecutive experiences and thus further increases stability during training.

Having defined the two methods, the loss function for DRL agents can be defined as:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\big[\delta_\theta^2\big]$$

where D is the Experience-Replay memory buffer and δ_θ^2 is defined as the quadratic loss of the TD-Error parameterized by the network θ. Equation 2.5 also follows the same structure.

3 System description

3.1 Environment

The agent's environment is created using the open-source, multi-modal traffic simulation package, Eclipse SUMO (Simulation of Urban MObility) (Lopez et al., 2018). Environments created in this package are controllable using the traffic control interface library, TraCI, programmed in Python, a high-level programming language (Van Rossum & Drake, 2009). This specific package was chosen because it seems to be the most popular tool for implementing RL and DRL algorithms for TSC (Haydari & Yılmaz, 2022).

The environment itself consists of a single cross-intersection, controlled using traffic lights (see Figure 3.1). Each of the four intersection legs has three lanes and a length of 200 meters. According to real world conventions, the rightmost, middle and leftmost lanes, seen from the driver's perspective, will be considered the first, second and third lane, respectively.

The intersection's TSC has in total 8 traffic light phases, 4 responsible for green light signals and 4 responsible for yellow light signals. Each green phase is followed by a 3 second yellow phase in which the intersection gets cleared, thus acting as a safety mechanism. The vehicle's behavior for yellow light signals is to pass through the intersection if a safe stop is not possible, otherwise to decelerate. As for the green light signals, vehicles decelerate on approach until reaching the *visibility distance* of 4.5 meters away from the intersection (Lopez et al., 2018). Afterwards, if the intersection is clear, they pass through. For the red light signal, vehicles must come to a complete stop.

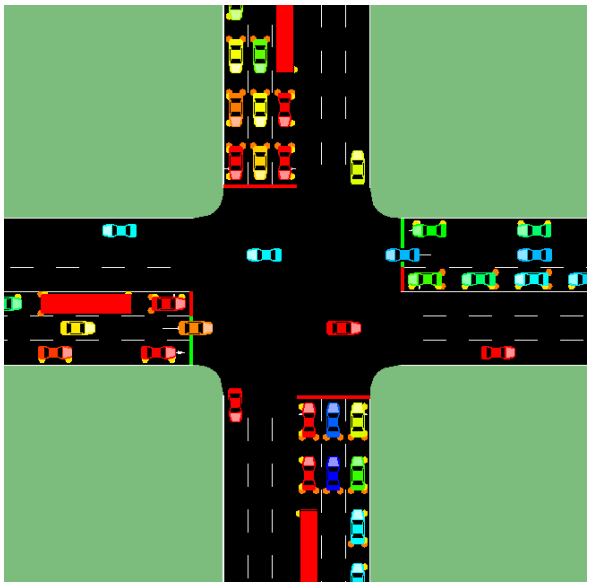
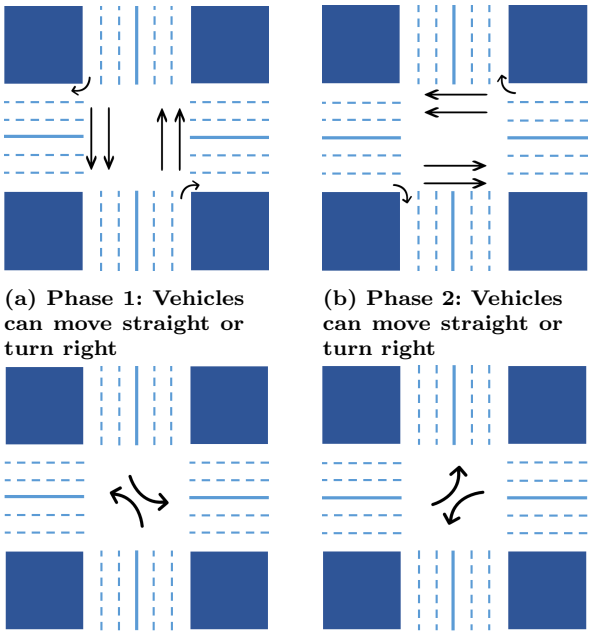


Figure 3.1: The agent's environment: a single cross-intersection with 3 lanes, seen in phase 2



(a) Phase 1: Vehicles can move straight or turn right

(b) Phase 2: Vehicles can move straight or turn right

(c) Phase 3: Vehicles can turn left

(d) Phase 4: Vehicles can turn left

Figure 3.2: The 4 green light phases

The 4 green phases can be observed in Figure 3.2. In the first two phases, vehicles can go straight using the first and second lanes or turn right via the first lane (see 3.2a, 3.2b). In the other two phases, vehicles can only turn left from the third lane (see 3.2c, 3.2d). The agent's actions involve choosing one of the four green phases to be applied to the traffic lights. When choosing a new green phase, the simulation automatically switches to the 3 seconds yellow phase corresponding to the current green phase. After the 3 seconds pass, the intersection will switch to the new chosen green phase.

As for the traffic, vehicles are generated at the end of the approach lanes and are removed from the simulation at the end of the depart lanes. The route of each vehicle is created using the *random trips* tool provided by SUMO. Two different fixed scenarios were randomly generated using this tool, representing medium and high traffic. These scenarios contain the route of each vehicle as well as the simulation time at which they are generated. Therefore, vehicles are generated 1.15 seconds apart for medium traffic or each second for high traffic. Finally, there are two types of vehicles generated in the simulation, passenger vehicles and buses. Passenger vehicles have a length of 4.5 meters and a maximum speed of 35 meters/second and are generated with a 90% probability. The buses have a length of 14 meters and a maximum speed of 30 meters/second and are generated with a probability of 10%.

3.2 Heuristic Strategies

In order to have a baseline comparison for the DRL agents, three other agents that act according to heuristic strategies were implemented. The first agent follows a Random policy by always choosing the next green phase randomly. The second agent implements a Longest-Queue-First policy, which involves choosing the green light phase corresponding to the lane with the most vehicles. Finally, the third agent follows a Fixed-Timings policy by selecting consecutive green light phases that last 45 seconds each. For all three agents, a 3 second yellow phase is automatically introduced to make the transition between the current green phase and the next. For the random and longest-queue-first agents, green light phases last for at least 10 seconds. More specifically, the agents are able to choose a new green light phase only after the 10 seconds have passed.

3.3 Deep Reinforcement Learning Agents

In total, four DRL agents are implemented to optimize the TSC policy for the simulated intersection. The first two agents represent classic DRL algorithms, denoted as Deep Q-Network (DQN) (Mnih et al., 2013) and Double Deep Q-Network (DDQN) (van Hasselt et al., 2015). The other two agents are the Deep Quality-Value (DQV) and Deep Quality-Value-Max (DQV-Max) algorithms, introduced by Sabatelli et al. (2020).

All DRL agents used the same state-action space, received the same rewards and had the same goal. The state was defined as the number of vehicles per lane. However, since the first and second lanes share green light phases, they were merged into a single lane. Therefore, the number of vehicles for the two lanes was summed, creating a state size of 8, each intersection leg only providing 2 lanes instead of 3. In addition, a special case was defined for vehicles which wanted to turn left but the third lane was already full, thus forcing them to wait on the first or second lanes. This behavior was a limitation of the simulation environment, which continued to generate vehicles despite the fact that the desired lane was full. In such cases, vehicles were counted as being found in the third lane to ensure that agents learn the proper policy in the first training steps, when actions were often taken randomly.

Furthermore, the agents had 4 possible actions, which involved choosing one of the 4 green phases to be used in the intersection. However, the agents were forced to maintain the green light phase for at least 10 seconds before being able to choose a new phase. This was implemented to ensure traffic lights are not changed too often or sudden, which would be unsafe in real-world scenarios.

Finally, the reward received at each time step was the negative accumulated waiting time, averaged for all vehicles in the simulation. More specifically, the accumulated waiting time represents the entire waiting time for each vehicle since it entered the simulation. Therefore, the goal of each agent was to decrease this average accumulated waiting time.

3.3.1 DQN

The first algorithm, also discussed in Section 2.3, is DQN (Mnih et al., 2013), the DRL implementation of Q-Learning. DQN aims to minimise the loss function shown in Equation 2.3, where the TD-Error δ_θ, parameterized by the network θ, is defined as:

$$\delta_\theta \;=\; r_t \;+\; \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) \;-\; Q(s_t, a_t; \theta)$$

As mentioned before, DQN uses the Experience-Replay memory buffer D to sample training batches and the Target-Network θ^-, to calculate the TD-Target.

3.3.2 DDQN

The DDQN (van Hasselt et al., 2015) algorithm, inspired by the Double Q-Learning algorithm (Hasselt, 2010), comes as a solution to DQN's Q value overestimation problem. This problem is caused by Q-Learning's method of calculating the TD-Target by using the same estimated Q value for action selection and evaluation (Hasselt, 2010). Double Q-Learning sought to reduce the overestimation of the classic Q-Learning algorithm by separating the action selection and evaluation steps. Therefore, the main idea behind DDQN is to use the online network θ to select the action and the target network θ^- to evaluate it. Similarly to DQN, the target network θ^- is updated periodically by copying the weights of the online network θ. Equation 3.1 shows the formula for the DDQN's TD-Error δ_θ, used in the loss function from Equation 2.3.

$$\delta_\theta = r_t + \gamma Q(s_{t+1}, \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a, \theta); \theta^-) - Q(s_t, a_t; \theta) \quad (3.1)$$

As it can be seen in Equation 3.1, the $\arg\max_{a \in \mathcal{A}}$ operator first selects the action that maximizes the Q value estimated by θ. This action is then used to evaluate the Q value of the next state, predicted by θ^-.

3.3.3 DQV

DQV is an on-policy algorithm which aims to learn both the V and Q value functions, by using two different neural networks, denoted as Φ and θ, respectively (Sabatelli et al., 2020). The loss function, parameterized by θ, is defined in Equation 2.3, while

the one parameterized by Φ can be seen in Equation 3.2. The corresponding TD-Errors δ_Φ and δ_θ of each loss function are then defined in Equations 3.3 and 3.4, respectively.

$$L(\Phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\left[\delta_\Phi^2\right] \quad (3.2)$$

The main idea behind DQV is to learn the V function via TD-Learning and then use V's corresponding target network Φ^- to calculate the TD-Target for the Q function. This can be seen in Equations 3.3 and 3.4, used to learn V and Q.

$$\delta_\Phi = r_t + \gamma V(s_{t+1}; \Phi^-) - V(s_t; \Phi) \quad (3.3)$$

$$\delta_\theta = r_t + \gamma V(s_{t+1}; \Phi^-) - Q(s_t, a_t; \theta) \quad (3.4)$$

As it can be seen, both TD-Error equations use the same TD-Target parameterized by the target network Φ^-, corresponding to the V function.

3.3.4 DQV-Max

DQV-Max is similar to DQV in the sense that it uses two different neural networks Φ and θ to learn both the V and Q functions, respectively. However, the TD-Targets are different since the V function is approximated by using the $\max_{a \in \mathcal{A}}$ operator, which involves off-policy learning (Sabatelli et al., 2020). More specifically, the TD-Target used for updating V is the same as the TD-Target used in DQN, also including the target network θ^- that corresponds to the Q network θ. This can be seen in Equation 3.5, which shows the TD-Error δ_Φ of the V function.

$$\delta_\Phi = r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - V(s_t; \Phi) \quad (3.5)$$

As for the Q function, the TD-Error δ_θ can be seen in Equation 3.6, which calculates the TD-Target by using the V value estimated by the online network Φ. The TD-Errors for the Q function are similar for both DQV and DQV-Max, with the exception that DQV uses the target network Φ^- while DQV-Max uses the online network Φ.

$$\delta_\theta = r_t + \gamma V(s_{t+1}; \Phi) - Q(s_t, a_t; \theta)] \quad (3.6)$$

3.4 Neural Network Architectures

The neural network models for all DRL agents were created using the application programming interface (API) Keras (Chollet et al., 2015). This API runs on top of the end-to-end, open-source platform for machine learning, Tensorflow (Abadi et al., 2015).

All DRL agents employed an online neural network θ to approximate the Q function. The architecture of θ involves 3 fully connected hidden layers, each with 32 nodes. All hidden layers used the Rectified Linear Unit (ReLU) activation function while the output layer used a linear activation function. The input layer receives a state of the agent containing 8 positive integers, while the output layer produces the 4 estimated Q values, one for each action. This neural network was trained using the loss function shown in Equation 2.3. Finally, DQN and DDQN employed the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 0.001, while DQV and DQV-Max used the RMSProp optimizer, as introduced in the Coursera course given by Hinton et al. (2016). The RMSProp optimizer used a learning rate of 0.001, a discounting factor of 0.95 and $\epsilon = 0.01$, the constant ensuring numerical stability. In addition to θ, DQV and DQV-Max used a second neural network Φ to approximate the V function. Φ is almost identical to θ with the sole exception that the output layer has a size of 1, producing the estimated V value for the given state. Φ minimized the loss function defined in Equation 3.2. Finally, the target networks θ^- and Φ^- had the same architecture as the corresponding θ and Φ, respectively.

3.5 Prioritized Experience Replay

Prioritized Experience-Replay (PER) is a type of Experience-Replay which was employed for all implemented DRL agents. The reason for using this method was that previous studies showed that PER can significantly increase the performance of DRL algorithms tasked with TSC (Fang et al., 2019). The main idea behind PER is to replace the uniform sampling of traditional Experience-Replay with prioritized sampling (Schaul et al., 2015). More specifically, the priority p_i of experience $i = \langle s_t, a_t, r_t, s_{t+1}\rangle$ is dictated by its TD-Error δ_i, as seen in the following equation:

$$p_i = |\delta_i| + \epsilon \tag{3.7}$$

where $\epsilon = 0.01$ represents a constant that ensures $p_i > 0$ and thus a non-zero probability for sampling each experience. The reasoning behind this non-zero probability is that prioritizing experiences with high TD-Errors can reduce diversity. This effect can be alleviated by using stochastic prioritization (Schaul et al., 2015), which defines the probability $P(i)$ of sampling experience i as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{3.8}$$

where α determines the prioritization level. For instance, when $\alpha = 0$, all probabilities are equal, meaning that the sampling is uniform. In this case, α was set to 0.8. Furthermore, k indicates that the denominator is the sum of all priorities.

In addition, prioritized experiences tend to be over-sampled since a bias for high TD-Errors is introduced. This bias can be alleviated by using importance-sampling (IS) weights (Schaul et al., 2015), as defined in Equation 3.9. The equation defines the weight w_i corresponding to experience i, which has the role of down-weighting experiences with a high probability $P(i)$ of being sampled and up-weighting those with a low probability. This weight is then included in the priority calculation by replacing δ_i with $w_i\delta_i$ in Equation 3.7. N represents the number of experiences stored in the experience buffer, while β controls how much importance-sampling correction is applied to each TD-Error. Initially, β was set to 0.3, however, this value was annealed to 1 by an increment of 0.0005 every time a batch is sampled. This was done to correct the bias more aggressively towards the end of the training, when the algorithm converges and thus requires updates to be unbiased (Schaul et al., 2015).

$$w_i = \left(\frac{1}{N} \times \frac{1}{P(i)}\right)^\beta \tag{3.9}$$

3.6 Experimental Setup

3.6.1 Training procedure and parameters

All DRL agents were trained on the medium traffic scenario and then tested on both the medium and the high traffic scenarios. The reason for this approach was to test the adaptability of the DRL agents compared to the heuristic strategies. Training involved using the agent's experiences to optimize the weights of the neural networks Φ and θ, tasked with learning V and Q. During testing, the pre-trained network θ was employed for action selection, without any update in the network's weights. Each agent was trained for 30000 simulation steps, using the PER technique. The memory buffer had a maximum capacity of 2000 experiences and training started when there were at least 300 experiences in the buffer. Once this minimum threshold was reached, the online networks were updated at each simulation step by sampling with priority training batches of size 32. The target networks, on the other hand, had an update frequency of 500 training steps. More specifically, once training started, target networks were updated every 500 steps. Finally, network updates used a discount factor $\gamma = 0.99$ and a learning rate of 0.001. These hyperparameters were chosen empirically.

In addition, each DRL agent used the ϵ-greedy exploration strategy, which balances the exploration and exploitation trade-off. The main idea behind this strategy is to explore the environment with a probability of ϵ and exploit the learned policy with a probability of $1-\epsilon$. Exploration is accomplished by having the agent take random actions, while exploitation involves choosing the action that maximizes the expected Q value for the next state. In this experiment, ϵ had an initial value of 0.7 and was annealed to 0.1 by a 3×10^{-5} increment every simulation step.

3.6.2 Agent evaluation

All agents were evaluated based on the accumulated waiting time averaged for all vehicles in the simulation. Therefore, an optimal agent would keep waiting times as low as possible while also being stable enough to ensure fairness between lanes. The main idea behind the latter statement is that a high fluctuation in waiting times indicates a lack of fairness between lanes, meaning that some lanes are not given enough priority.

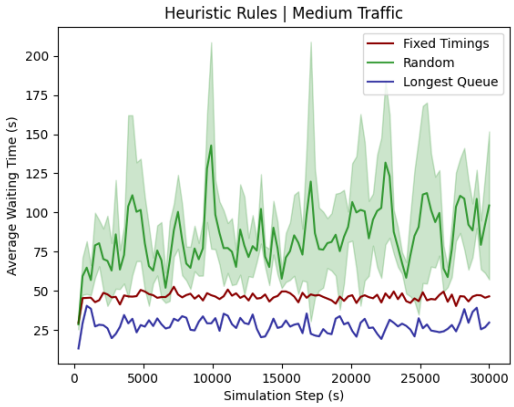


Figure 4.1: The average accumulated waiting times achieved by the heuristic strategies for medium traffic.

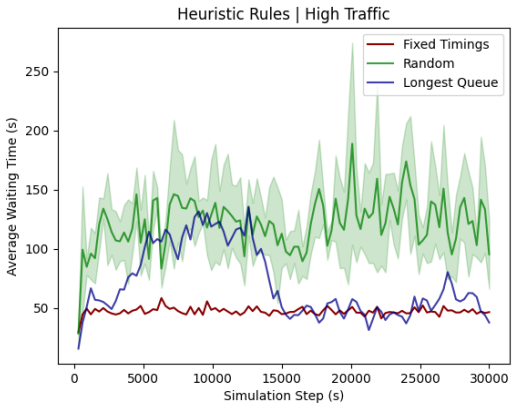


Figure 4.2: The average accumulated waiting times achieved by the heuristic strategies for high traffic.

4 Results

In order to have a baseline comparison, the average accumulated waiting times achieved by the heuristic strategies can be seen in Figures 4.1 and 4.2. Figure 4.1 shows that the Longest-Queue-First policy manages to reduce waiting times the most for medium traffic. On the other hand, Figure 4.2 shows that the Fixed-Timings policy achieved lower and more stable waiting times for high traffic. This trend can also be seen in Table 4.1, which shows the accumulated waiting times averaged over the 30000

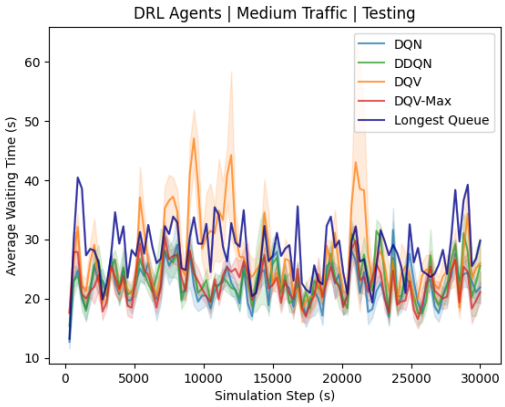


Figure 4.3: The average accumulated waiting times achieved by the trained DRL agents at each simulation step for medium traffic. The longest-queue-first policy was also included as a baseline.

simulation steps. For medium traffic, the Longest-Queue-First policy achieved an average waiting time of 28.11 seconds, almost 20 seconds less than the 46.09 seconds given by the Fixed-Timings policy. The trend is reversed for high traffic, in which Fixed-Timings achieves an average waiting time of 47.36 seconds, compared to 70.77 seconds given by the Longest-Queue-First policy. As expected, randomly choosing green light phases leads to longer waiting times, regardless of the traffic situation.

Furthermore, Figures 4.3 and 4.4 present the average accumulated waiting times achieved by the trained DRL agents at each simulation step, compared to the optimal heuristic for the specific traf-

Table 4.1: The accumulated waiting time (s) for each agent, averaged over 30000 simulation steps | Medium and High Traffic scenarios

Agent	Medium	High
Random	84.62s	122.35s
Fixed-Timings	46.09s	**47.36s**
Longest-Queue-First	**28.11s**	70.77s
DQN	**22.89s**	**35.27s**
DDQN	23.25s	45.60s
DQV	26.80s	43.80s
DQV-Max	**21.53s**	**34.02s**

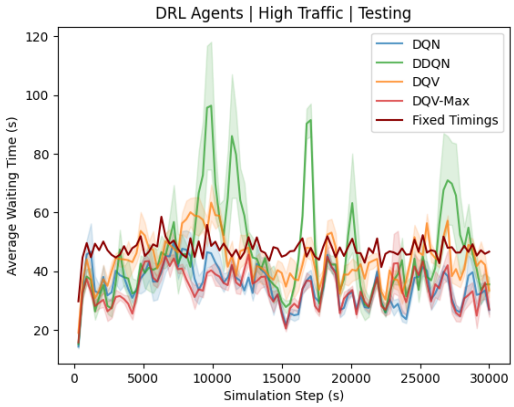


Figure 4.4: The average accumulated waiting times achieved by the trained DRL agents at each simulation step for high traffic. The Fixed-Timings policy was also included as a baseline.

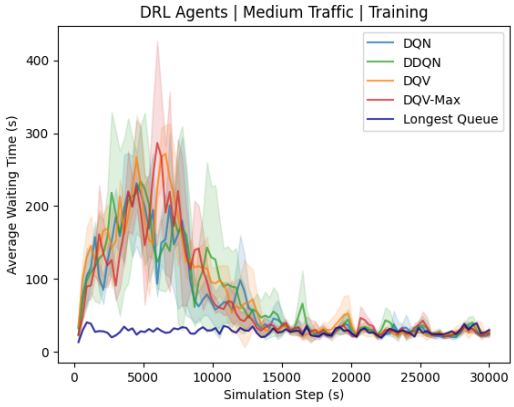


Figure 4.5: The average accumulated waiting times achieved by the DRL agents at each simulation step when training on medium traffic. The longest-queue-first policy was also included as a baseline.

Table 4.2: The average required training time (minutes) for each agent to complete the 30000 simulation steps| Medium Traffic scenario

Agent	Training time
DQN	15.26 min
DDQN	15.53 min
DQV	19.51 min
DQV-Max	20.76 min

fic scenario. It can be seen that the DRL agents are, on average, an improvement over the heuristic strategies, achieving lower waiting times for both traffic scenarios. This is confirmed by analyzing Table 4.1, which shows that all DRL agents achieve lower average waiting times compared to the heuristic strategies agents. However, DQN and DQV-Max seem to produce lower and more stable waiting times compared to DDQN or DQV, which show a noticeable instability for high and medium traffic, respectively. This instability is also visible in Table 4.1, where DDQN and DQV have on average longer waiting times than DQN and DQV-Max.

Finally, Table 4.2 contains the number of minutes required for each DRL agent to complete the 30000 training simulation steps. It can be seen that both DQN and DDQN take around 15 minutes, while DQV and DQV-Max around 20 minutes, which is an approximated 33% increase. Despite this, Figure 4.5, which contains the average accumulated waiting times achieved by the DRL agents at each simulation step during training, shows that all DRL agents converge in about the same number of simulation steps.

5 Discussion

All in all, the DRL agents show, on average, shorter waiting times than what was achieved by the widely used heuristic strategies, Longest-Queue-First or Fixed-Timings. However, DDQN and DQV presented some instability in the form of waiting-time spikes, allowing waiting times to sometimes rise above the heuristic baseline. In the simulation, such scenarios were often linked to lanes which were forced to wait longer than the rest, which increased the averaged waiting time for the entire intersection. The two algorithms that were more stable, DQN and DQV-Max, have a common element in the sense that they both employed the $\max_{a \in \mathcal{A}}$ operator to calculate the TD-Error. More specifically, DQN and DQV-Max select the maximum expected Q value for the next state when updating the Q and V function, respectively. For DQV-Max, this V function is then used to update the Q function as well, which is responsible for the action selection. With these results in mind, it can be stated that DQV-Max shows similar or better performance compared to classic DRL algorithms

when it comes to TSC. Moreover, despite its instability, DQV has on average shorter waiting times than all heuristic strategies tested. However, its performance for TSC seems to be at most similar or slightly lower than DQN and DDQN.

Furthermore, it must be noted that the environment was less complex than other environments usually employed for testing DRL agents, having a small state-action space. More specifically, the agent was able to perform only 4 actions while the state itself consisted of 8 positive integers, also limited in scale by how many vehicles could fit in the intersection. Therefore, it might be the case that more complex environments could favour the DQV and DQV-Max algorithms, allowing them to distance themselves from the classic DRL algorithms. This occurred in Sabatelli et al. (2020), where DQV and DQV-Max were tested on the more complex Atari-2600 benchmark (Bellemare et al., 2013) and showed a better performance than DQN and DDQN. In addition, the simple environment meant training times were short, around 15-20 minutes. In this thesis, the DQV and DQV-Max algorithms had a longer training time but converged in the same amount of simulation steps as DQN and DDQN. The longer training time was to be expected since both algorithms train two different neural networks simultaneously, compared to DQN and DDQN which only train a single neural network. However, it might be the case that in more complex environments, DQV and DQV-Max would require less steps to converge, which could balance the higher training time. This was the case in Sabatelli et al. (2020), where the two algorithms converged significantly faster than DQN and DDQN when deployed in the Atari-2600 benchmark (Bellemare et al., 2013). The reason for this could be that the V function, which is used to train the Q function, converges using a lower number of parameters (Sabatelli et al., 2020).

Regarding the real-world deployment of DRL algorithms in the context of TSC, there are some issues which must be addressed. Firstly, training on real-world traffic would be unfeasible because of the trial-and-error approach of RL algorithms, which would cause significant traffic congestion (Rasheed et al., 2020). The alternative, which involves using simulated environments, has the drawback that such environments are often not realistic enough to account for unexpected driver or pedestrian behavior. In addition, simulated environments allow DRL agents to use traffic data which is impractical to gather in real-world scenarios. For instance, one common approach for DRL optimised TSC is to gather information about the traffic from aerial images or videos given by the simulation (Rasheed et al., 2020). However, in real-world scenarios, employing cameras to gather aerial images is impractical, expensive and can also be affected by poor weather conditions. While some research found that higher-dimensional state representations are beneficial for DRL optimized TSC (Zhang et al., 2018), it has to be investigated whether the extra performance can out-weight the practical matters of data gathering. On the contrary, Genders & Razavi (2018) found no difference in performance between lower and higher-dimensional state representations. This could be an additional argument for employing simpler states determined using sensors, rather than the more complex states that require cameras, radars or even vehicles connected to the same network.

For these reasons, the current thesis opted to train the DRL agents on much simpler states, namely the number of vehicles on each lane, which can be computed using simple sensors. To be noted that the number of vehicles per lane is not equivalent to the queue length, which would imply that vehicles are stationary. The limitation of queue lengths comes from the fact that slow moving vehicles are not considered to be in a queue and would thus be ignored by the TSC. In order to count the number of vehicles for each lane, a threshold which determines where the counting begins should be set in place. However, this was not required in the current thesis since a single intersection was implemented, whose lanes did not feed into other intersections.

Overall, the current literature presents a major obstacle in the way of real-world deployment of DRL agents for TSC. More specifically, there seems to be no widely adopted benchmark for realistic TSC simulations. In this way, algorithms are tested on different intersection layouts and traffic scenarios, each measuring different metrics with regards to success. This makes comparing algorithms difficult, which would dispel potential early adopters of DRL optimized TSC. Furthermore, many algorithms are tested in unrealistic environments, which was also the case for the current thesis. While such

approaches can act as a proof of concept for different algorithms, they do not meet the criteria for deployment. The solution for this would be the adoption of realistic TSC benchmark tools such as *LemgoRL*, a realistic simulation of an intersection found in Lemgo, Germany (Muller et al., 2021). Such tools, which employed real-world data and ensured safety regulations compliance, could accelerate the real-world deployment of DRL agents in the context of TSC. As a final note, it should be stated that heuristic strategies are already a good enough solution for intersections that do not present a high risk of congestion. Therefore, deploying DRL agents for TSC in such cases would often not be worthwhile.

5.1 Limitations

The fact that vehicles were generated each second or each 1.15 seconds did not cover the true stochastic nature of traffic. Moreover, in the real-world, some lanes and routes are in higher demand than others, which would change the performance of some heuristic strategies. Therefore, it cannot be estimated exactly how well the DRL algorithms would perform in a truly stochastic and realistic environment. Finally, hyperparameter tuning in the context of DRL is often a highly demanding task, time and processing power wise. For this reason, the four DRL agents implemented used the same hyperparameters in most situations. However, it might be the case that some hyperparameter values are not optimal for all DRL agents analyzed in this thesis.

5.2 Conclusion and future work

Finally, DQV and DQV-Max present an alternative to classic algorithms currently used in the DRL field. The current thesis showed that their performance is similar or surpasses classic DRL algorithms and heuristic strategies in the context of simulated TSC. Future studies should analyze whether the algorithms would benefit from a soft update of the target networks. More specifically, rather than completely copying the weights from the online network, a soft update would imply more frequent, partial updates of the target network (Kobayashi & Ilboudo, 2021). Moreover, it should be inspected whether DQV and DQV-Max overestimate the Q value when used for TSC. Sabatelli et al. (2020) showed that the two algorithms do not suffer from Q overestimation like DQN does. However, this was only tested for the Atari-2600 benchmark, not for TSC. A final aspect which could be analyzed would be the removal of the 10 seconds minimum green time. This would give the DRL agent more freedom when learning, which might improve performance. However, safety concerns should be taken into considerations to ensure dangerous traffic light switches do not occur.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013, jun). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, *47*, 253–279. doi: 10.1613/jair.3912

Chollet, F., et al. (2015). *Keras.* GitHub. Retrieved from `https://github.com/fchollet/keras`

Fang, S., Chen, F., & Liu, H. (2019, oct). Dueling Double Deep Q-Network for Adaptive Traffic Signal Control with Low Exhaust Emissions in A Single Intersection. *IOP Conference Series: Materials Science and Engineering*, *612*(5), 052039. doi: 10.1088/1757-899x/612/5/052039

Genders, W., & Razavi, S. (2018). Evaluating reinforcement learning state representations for adaptive traffic signal control. *Procedia Computer Science*, *130*, 26–33. doi: https://doi.org/10.1016/j.procs.2018.04.008

Hasselt, H. (2010). Double Q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, & A. Culotta (Eds.), *Advances in neural information processing systems* (Vol. 23). Curran Associates, Inc.

Haydari, A., & Yılmaz, Y. (2022). Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey. *IEEE Transactions on*

Intelligent Transportation Systems, *23*(1), 11–32. doi: 10.1109/TITS.2020.3008612

Hinton, G., Srivastava, N., & Swersky, K. (2016). *Neural Networks for Machine Learning.* Retrieved from `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\_slides\_lec6.pdf`

Hornik, K., Stinchcombe, M., & White, H. (1989, jul). Multilayer Feedforward Networks Are Universal Approximators. *Neural Netw.*, *2*(5), 359–366.

Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization.* arXiv. doi: 10.48550/ARXIV.1412.6980

Kobayashi, T., & Ilboudo, W. E. L. (2021, apr). t-soft update of target network for deep reinforcement learning. *Neural Networks*, *136*, 63–71. doi: 10.1016/j.neunet.2020.12.023

Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*, 293–321. doi: https://doi.org/10.1007/BF00992699

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., ... Wießner, E. (2018). Microscopic Traffic Simulation using SUMO. In *The 21st IEEE International Conference on Intelligent Transportation Systems.* IEEE.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning.* arXiv. doi: 10.48550/ARXIV.1312.5602

Muller, A., Rangras, V., Ferfers, T., Hufen, F., Schreckenberg, L., Jasperneite, J., ... et al. (2021). Towards real-world deployment of reinforcement learning for Traffic Signal Control. *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. doi: 10.1109/icmla52953.2021.00085

Rasheed, F., Yau, K.-L. A., Noor, R. M., Wu, C., & Low, Y.-C. (2020). Deep Reinforcement Learning for Traffic Signal Control: A Review. *IEEE Access*, *8*, 208016–208044. doi: 10.1109/access.2020.3034141

Sabatelli, M., Louppe, G., Geurts, P., & Wiering, M. A. (2020). The Deep Quality-Value Family of Deep Reinforcement Learning Algorithms. *2020 International Joint Conference on Neural Networks (IJCNN)*. doi: 10.1109/ijcnn48605.2020.9206677

Samal, S. R., Kumar, P. G., Santhosh, J. C., & Santhakumar, M. (2020, dec). Analysis of Traffic Congestion Impacts of Urban Road Network under Indian Condition. *IOP Conference Series: Materials Science and Engineering*, *1006*(1), 012002. doi: 10.1088/1757-899x/1006/1/012002

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). *Prioritized Experience Replay.* arXiv. doi: 10.48550/ARXIV.1511.05952

Sutton, R. (1988, 08). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44. doi: 10.1007/BF00115009

Sutton, R., Bach, F., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT Press Ltd.

van Hasselt, H., Guez, A., & Silver, D. (2015). *Deep Reinforcement Learning with Double Q-learning.* arXiv. doi: 10.48550/ARXIV.1509.06461

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace.

Zhang, R., Ishikawa, A., Wang, W., Striner, B., & Tonguz, O. (2018). *Using Reinforcement Learning with Partial Vehicle Detection for Intelligent Traffic Signal Control.* arXiv. doi: 10.48550/ARXIV.1807.01628