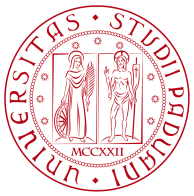


University of Padova

Department of Mathematics

Master Thesis in Data Science

Adaptive Traffic Light Control Using Double Deep Q-Networks: Balancing Efficiency and Fairness for Urban Mobility

Supervisor
Gian Antonio Susto
University of Padova

Co-supervisor
Matteo Cederle, PhD
University of Padova

Master Candidate
Giacomo Scatto

Academic Year
2023-2024

I'm deeply grateful to everyone who contributed
their time and support throughout this entire experience.

Abstract

One of the most critical challenges in modern urban society is related to traffic management, where too many inefficiencies are leading to unacceptable levels of road congestion, pollution and delays, both for vehicles and pedestrians.

This thesis explores a novel approach to optimize traffic light control, exploiting Deep Reinforcement Learning (RL) techniques. The goal is to build an RL agent able to dynamically select the optimal traffic light phase and determine the appropriate duration for which to maintain it, eventually reducing traffic congestion and enhancing the overall traffic flow.

The proposed RL agent has been trained to adapt to varying levels of traffic, ranging from light to moderate and eventually heavy levels of congestion, ensuring a stable and robust behavior under different scenarios. Additionally, this study analyzes the consequences of using different time intervals for the agent's action, investigating how this affects the overall system performance. Finally, this study is distinguished from most of the works in the literature for the focus on vulnerable road users, specifically on pedestrians. In this context, during the decision-making process the model takes into consideration both vehicle and pedestrian flows, balancing their needs based on the relative assigned weight. An analysis was conducted on three different weight levels, aiming at finding a trade-off strategy able to ensure fairness of service both to drivers and pedestrians.

The findings will highlight how RL – and specifically Deep RL techniques – provides a promising solution to traffic management, significantly enhancing urban mobility by reducing traffic jams and thus improving the overall experience for all the members involved.

Contents

Abstract v

List of figures ix

1 Introduction 1
- 1.1 The Modern Scenario . . . 1
- 1.2 Integrating Technology into Urban Traffic Management . . . 2
- 1.3 Leveraging Reinforcement Learning to solve Traffic Challenges . . . 2
- 1.4 Related Works . . . 3
  - 1.4.1 Conventional Traffic Light Control . . . 3
  - 1.4.2 RL Traffic Light Control . . . 4
- 1.5 Method . . . 5

2 Foundations of Reinforcement Learning 7
- 2.1 RL principles and MDPs . . . 7
  - 2.1.1 Expected Discounted Return . . . 8
  - 2.1.2 Value Function . . . 9
  - 2.1.3 Policy . . . 10
- 2.2 Bellman Equations . . . 10
- 2.3 Model-Free Methods . . . 11
  - 2.3.1 Monte Carlo Methods . . . 12
  - 2.3.2 Temporal Difference Methods . . . 12
- 2.4 Deep Reinforcement Learning . . . 14
  - 2.4.1 Deep Q-Learning . . . 14
  - 2.4.2 Deep Q-Networks . . . 16
  - 2.4.3 Double DQN . . . 16

3 Proposed Method 19
- 3.1 Network building . . . 19
- 3.2 Environment definition . . . 20
  - 3.2.1 State Space . . . 20
  - 3.2.2 Action Space . . . 21
  - 3.2.3 Reward Functions . . . 22
- 3.3 Scenarios . . . 24
  - 3.3.1 Varying Traffic Conditions . . . 24

3.3.2 Varying Decision-Making Frequency . . . 24
3.3.3 Balancing Priority between Vehicles and Pedestrians . . . 25

4 Experimental Setup 27
4.1 Problem definition . . . 27
4.2 Simulation Environment . . . 28
4.2.1 Road Network . . . 28
4.2.2 Inflows . . . 30
4.3 RL Environment . . . 31
4.3.1 State Space . . . 31
4.3.2 Reward Function . . . 32
4.4 Scenarios . . . 33
4.4.1 Varying Traffic Conditions . . . 33
4.4.2 Varying Decision-Making Frequency . . . 34
4.4.3 Balancing Priority between Vehicles and Pedestrians . . . 34
4.5 Algorithm details . . . 34

5 Results 37
5.1 Waiting Time . . . 38
5.1.1 Varying Traffic Conditions . . . 38
5.1.2 Varying Decision-Making Frequency . . . 42
5.1.3 Fairness - Varying β . . . 43
5.2 Queue Length . . . 45
5.2.1 Varying Traffic Conditions . . . 45
5.2.2 Varying Decision-Making Frequency . . . 48
5.2.3 Fairness - Varying β . . . 49
5.3 Total Delay . . . 51
5.3.1 Varying Traffic Conditions . . . 51
5.3.2 Varying Decision-Making Frequency . . . 54
5.3.3 Fairness - Varying β . . . 55
5.4 Comments . . . 56

6 Conclusion 59
6.1 Limitations and Future Improvements . . . 60

References 61

Acknowledgments 65

Listing of figures

2.1 Agent-Environment Interaction 8

3.1 Road Network 20

4.1 Deep RL Framework for Traffic Light Control 28
4.2 Visual Representations of the Considered Intersection 29
4.3 Green Time Length for Moderate Traffic - 5sec scenario 33

5.1 Waiting Time - Metrics Comparison under Light Traffic 38
5.2 Waiting Time - Metrics Comparison under Moderate Traffic 39
5.3 Waiting Time - Metrics Comparison under Heavy Traffic 39
5.4 Waiting Time - Metrics Comparison across Varying Traffic Levels 40
5.5 Waiting Time - Metrics Comparison across Varying Frequency Levels 42
5.6 Waiting Time - Metrics Comparison across Varying Fairness Levels 43
5.7 Queue Length - Metrics Comparison under Light Traffic 45
5.8 Queue Length - Metrics Comparison under Moderate Traffic 46
5.9 Queue Length - Metrics Comparison under Heavy Traffic 46
5.10 Queue Length - Metrics Comparison across Varying Traffic Levels 47
5.11 Queue Length - Metrics Comparison across Varying Frequency Levels 48
5.12 Queue Length - Metrics Comparison across Varying Fairness Levels 49
5.13 Total Delay - Metrics Comparison under Light Traffic 51
5.14 Total Delay - Metrics Comparison under Moderate Traffic 52
5.15 Total Delay - Metrics Comparison under Heavy Traffic 52
5.16 Total Delay - Metrics Comparison across Varying Traffic Levels 53
5.17 Total Delay - Metrics Comparison across Varying Frequency Levels 54
5.18 Total Delay - Metrics Comparison across Varying Fairness Levels 55

1 Introduction

1.1 The Modern Scenario

Urbanization is an ever-growing trend in modern cities. The development of large urban centers undoubtedly generates advantages from an economic and technological perspective. However, on the other hand, it also introduces critical challenges from a social, urban planning and ecological point of view.

Today, over 40 million cars circulate in Italy; if we add to this the poor management of public transport across the majority of the country, it is clear that road traffic is a problem that should not be overlooked; urban centers face daily gridlocks, impacting not only individual mobility but also public and commercial transportation.

A European Commission report [1] estimates that traffic congestion costs the European Union €110 billion each year, approximately 1% of its Gross Domestic Product (GDP); in 2020, it is estimated [2] that the transport sector was responsible for 37.3% of nitrogen oxide (NOx) emissions in Italy, not to mention the economic damages due to delays or those related to noise pollution. These "negative externalities" represent a cost for the society in general, and reducing road congestion could yield substantial economic returns as well as improving the overall quality of life.

1.2 Integrating Technology into Urban Traffic Management

The dualistic perspective of the consequences brought by the technological advancement offers us also the instruments to address the traffic management problem.

The application of technological innovations at a "urban level" gave raise to the concept of *smart cities*; a comprehensive definition is provided by Dameri (2013) [3]:
"A smart city is a well defined geographical area, in which high technologies such as ICT, logistic, energy production, and so on, cooperate to create benefits for citizens in terms of well being, inclusion and participation, environmental quality, intelligent development; [...]".

Within this smart-city framework, traffic management plays a crucial role. To effectively reduce congestion, it is necessary to fully understand traffic patterns and have a wide comprehensions of its dynamics.

Advances in information technology and telecommunications, along with the exponential growth in computing power, have made it possible to gather vast amount of information and handle large-scale databases. This technological revolution allowed the construction of systems able to leverage data, sensors, and algorithms to dynamically optimize traffic flow, reducing delays and improving overall efficiency.

One such approach focuses on the optimization of traffic light systems, a critical component of urban traffic management. Traditional traffic lights operate on fixed timers, often failing to account for real-time traffic conditions. This results in inefficient green light durations, unnecessary stops, and extended queues. The inefficiencies of these static systems become more evident during peak hours or unexpected surges in traffic volume [4].

1.3 Leveraging Reinforcement Learning to solve Traffic Challenges

A promising approach in this domain is given by Reinforcement Learning (RL) [5]: an incredibly evolving realm of Machine Learning, perfectly suited for solving the traffic signal control problem, due to its ability of generalization, scalability and real-time applicability. By leveraging these capabilities, RL offers a highly effective solution for dynamically managing traffic in modern urban settings. Through this approach, smart cities can more effectively optimize

traffic flow, reduce congestion, and ultimately improve both the environmental and social well-being of urban areas.

The research presented in this thesis addresses the challenges discussed above through the development of an algorithm designed to enhance traffic light efficiency. Our model, by leveraging real-time data, makes adaptive adjustments to traffic light timings, prioritizing smoother vehicular flows while minimizing congestion. Taking into account factors such as vehicle density, queue length, and pedestrian activity, the algorithm is able to dynamically allocate green light time based on demand.

Significant attention has also been dedicated to the integration of pedestrians in the algorithm's decision-making process. Urban mobility is not limited to vehicles: pedestrians represent a significant and vulnerable group of road users, whose needs are often overlooked by traditional traffic systems; this lack of governance eventually results in unsafe crossings and inefficient movement patterns. By incorporating pedestrian flows into the optimization process, the proposed system aims to balance the needs of all road users, enhancing safety and reducing waiting time for both pedestrians and vehicles.

1.4 Related Works

1.4.1 Conventional Traffic Light Control

Early traffic light control methods can be broadly divided into two categories: pre-timed signal control and vehicle-actuated control method. There exists also a third category, which however represents an evolution from these "deterministic" systems in how they handle pre-determined rules: fuzzy logic control.

1. *Fixed-Time Traffic Control Systems*
   These systems are based on fixed-time control, where phases are calculated on pre-set schedules, without taking into consideration current traffic conditions. These systems rely on historical data to design specific phase subdivisions for each time of the day. While easy to implement, they are not capable of handling real-time fluctuations in traffic demand, hence resulting in poor performances under unusual traffic conditions, such as accidents or weather events. Nevertheless, they still represent the main approach adopted worldwide.

2. *Actuated Traffic Control Systems*
   These systems have been developed to overcome the limitation of fixed-time schemes. For instance, they exploit sensors like inductive loops embedded in the pavement to detect the presence of vehicles and adjust phases accordingly. In smaller towns, semi-actuated traffic signal systems often provided an effective solution for balancing efficiency and simplicity: sensors were put on major roads to adjust green light duration, while minor roads maintained prearranged timings. Despite being more responsive than fixed-time systems, their decision-making capabilities remain local (i.e. they address conditions at individual intersections without considering network-wide optimization) and still based on hand-crafted rules.

3. *Fuzzy Logic Traffic Control Systems*
   Fuzzy logic systems emerged as an alternative to traditional rule-based traffic light systems. These systems address the limitations of rigid, deterministic rules by introducing flexibility to handle uncertainties in traffic dynamics, allowing them to manage non-linear relationships. Nevertheless, they're still dependent on predefined rules and parameters, which limits their adaptability.

As already mentioned, these systems rely on hand-crafted rules tailored to current traffic conditions and do not consider future changes; as a result, they fail to achieve a globally optimal traffic solution. By moving beyond the strict frameworks of rule-based systems, fuzzy logic systems paved the way for more adaptive, data-driven approaches like Reinforcement Learning, which further leverage real-time feedback and continuous improvement.

1.4.2 RL Traffic Light Control

A lot of research has been done in academic and industry communities to build adaptive traffic signal control systems. Due to the limited computing power and simulation tools, early studies focus on solving the problem by fuzzy logic or linear programming. In these works, road traffic is modeled by limited information, which cannot be applied in large scale systems. With the development of deep learning and reinforcement learning, traffic control systems have become more adaptive and efficient, enabling real-time optimization of traffic flows and significantly reducing congestion.

One of the milestones in the field of deep RL dates back to 2015 and represents one of the most widely used and important algorithms in the current landscape. In that year, Mnih et al. [6] demonstrated human-level control through the Deep Q-Network (DQN) algorithm, which blends Q-learning with deep neural networks to perform complex tasks. Further studies

implemented enhanced version, as the Double DQN [7], developed to mitigate the overestimation bias in Q-value predictions, and the Dueling DQN [8]. These advancements have made DQN and its variants cornerstones of modern reinforcement learning research, with applications spanning a wide range of domains.

Building on this, various studies have applied similar RL methodologies to traffic management [9, 10, 11, 12, 13, 14, 15]. These works vary under several key aspects. First, in the state representation, including hand-crafted features (e.g., lane density, queue length, traffic light configuration, or image-like features as position and velocity. Second, they may differ in the reward design, including waiting time, average delay, queue length and outflow rate.

In terms of action space, there are two main schools of thought: one that uses binary actions (i.e.: maintain phase or switch to the next), and another that expands the action space by indicating how many seconds to extend the current phase, based on a pool of options.

In addition to the more frequent DQN-based models, some studies explored the performance of policy gradient algorithms, as Deep policy-gradient (PG) [12] or Actor-Critic methods [15], eventually achieving comparable results.

Recent advancements have expanded the RL environment to include mixed pedestrian - vehicle contexts. Zhu et al. [16], developed a context-aware multiagent broad RL framework, which accounted for both pedestrian and vehicle movements, incorporating context-awareness into the state space. Yazdani et al. [14] further refined this approach with the Intelligent Vehicle Pedestrian Light (IVPL) system, optimizing signals for both vehicles and pedestrians, emphasizing safety and efficiency.

This project builds on these advancements by employing double DQN while aiming at considering pedestrian traffic, with the goal of optimizing the overall traffic flow.

1.5 Method

Our approach followed the outlined works by making different new contributions:

1. **Selection of approaching phases**: Typically, the studies presented only took into consideration the possibility of either keeping the current phase or switching to the subsequent one, without the option to direct the flow to a specific phase. In our work, we decided to improve the boolean action space approach by moving towards a three-option based model, which allows the agent to be more flexible by choosing which of the three phases to select. Thanks to this method, our model can still decide to keep the current

phase by choosing the one it is currently running, or switch to the phase it considers optimal.

2. **Fairness implementation and focus on pedestrian flow**: Very few studies ([14, 16]) have considered both vehicle and pedestrian flow in the traffic-light control problem. We wanted to grant fairness in our system by teaching the model to weight the importance given to each of the two flows considered, based on request. Unlike previous approaches that either focused exclusively on vehicle flow or lacked dynamic adaptation for pedestrians, our system is designed to prioritize both flows as needed. To achieve this, we trained our model according to three different weight configurations, so as to demonstrate its performance under alternative scenarios.

To conclude this introduction, we leave a brief outline of the structure of the remainder of this project, composed by four chapters: a brief introduction to Reinforcement Learning will be provided in Chapter 2, so as to give to anyone a guidance on the flow of this work. Afterward, the problem and the approach adopted will be outlined in Chapter 3, focusing on the methodological aspects, and in Chapter 4, from a more technical perspective. The experimental results will be presented and analyzed in Chapter 5. Finally, in Chapter 6 we will exhibit our conclusions, the limitations of the study and future improvements.

2

Foundations of Reinforcement Learning

To clarify what it means to use RL algorithms, it is important to start with the basic concepts they are built upon. Therefore, in this chapter we are going to introduce RL in terms of its components, such as the agent-environment interaction. Following this, we will explore the concept of policy and its role in guiding the agent's actions. Afterward, we will come up to value functions and Bellman equations, which form the basis for evaluating policies.

Eventually, the chapter includes a brief analysis of some of the most common algorithms, such as Q-Learning [17] and DQN [6], crucial for our project.

This introduction covers only the relevant aspects for our work in the context of single-agent RL.

2.1 RL principles and MDPs

Reinforcement Learning is what it is more close to the concept of learning than ever has been formulated. If we think about the primal form of cognition we would probably remind about children, who learn how to act simply by interacting repeatedly with their surrounding environment, recording which actions lead to success and which result in failure. Similarly, RL models learn by trial and error, continuously adapting their behavior based on the feedback they receive from the environment they are placed in. An intuitive representation of the agent-environment relation is shown in Figure 2.1.

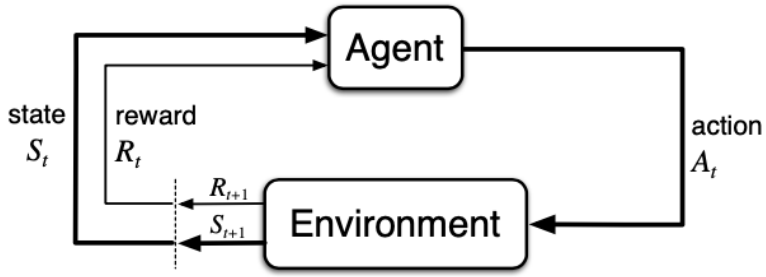


Figure 2.1: Agent-Environment Interaction - Image adapted from [5]

To actually exploit such framework, we need to formalize it. In RL, it is usually assumed that the environment can be described by a Markov Decision Process (MDP), which provides a mathematical framework to model decision-making in uncertain environments. It consists of:

- S: a finite set of States, i.e. all possible configurations the environment can be in;
- A: a finite set of possible Actions the agent can take;
- P: State Transition probability function. A function that describes the probability of moving from one state to another, given an action.
- R: Reward function. A function that assigns a scalar feedback (i.e. reward) to each state-action pair.
- γ: Discount Factor, $\gamma \in [0, 1)$

An MDP begins in an initial state $s_0 \in S$, which is sampled from a distribution of initial states μ. At each time step t, the agent observes the current state $s_t \in S$ and selects an action $a_t \in A$ based on its policy $\pi(a_t|s_t)$, which defines the probability of choosing action a_t given state s_t. Once the agent takes the action, the MDP transitions to a new state $s_{t+1} \in S$ with a probability given by $P(s_{t+1}|s_t, a_t)$, and the agent receives a reward $r_t = R(s_t, a_t, s_{t+1})$. These steps are repeated either until the process reaches a terminal state, completes a maximum of T time steps, or potentially continues indefinitely in the case of a non-terminating MDP.

2.1.1 Expected Discounted Return

A fundamental characteristic of the MDP framework is given by the Markov Property, from which it derives its name. It states that the reward and future state are conditionally independent of previous states and actions, given the current ones:

$$Pr(s_{t+1}, r_t | s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = Pr(s_{t+1}, r_t | s_t, a_t) \tag{2.1}$$

This implies that, in an MDP, the current state provides enough information to choose the optimal action. The goal of an agent, in fact, is to select an optimal policy able to maximize the *return* over time.

With *return* we refer to the total discounted reward from time step t, and it is formulated as in (2.2).

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

The discount factor γ is introduced to scale the importance of future weights. Logically, the closer the reward, the higher its importance; viceversa, the further it is from *t*, the lower will be the factor by which we'll multiply the corresponding reward, and so the lower the result.

If we set $\gamma = 0$, the agent is said to be myopic (or short-sighted), since it prioritizes immediate rewards rather than future ones. If $\gamma \simeq 1$, then the agent will place greater importance to long-term rewards.

2.1.2 Value Function

It is now appropriate to introduce the concept of value function, which plays a fundamental role in much of RL theory.

The value function measures the "goodness" of a state, and is defined as the expected sum of rewards the agent will receive while following a specific policy π, starting from a state s.

The value function, $V_\pi(s)$, for policy π, is given by:

$$V_\pi(s) = E_\pi[G_t \mid s_t = s] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s \right] \tag{2.3}$$

Similarly, the action-value function, also known as the Q-function, can be defined as the expected sum of rewards when taking an action a in a state s, and thereafter following policy π. The action-value function $Q_\pi(s, a)$ is defined as:

$$Q_\pi(s, a) = E_\pi[G_t \mid s_t = s, a_t = a] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right] \tag{2.4}$$

2.1.3 Policy

Strictly related to the value function is the concept of policy, which is a guideline that points to the agent which action it has to take. We can see the agent's policy as a "behavior function", a map from state to action. Consequently, the goal of maximizing cumulative rewards can be reinterpreted in terms of the policy, where the objective becomes now finding the optimal policy that eventually will yield the highest expected return.

We can have deterministic policies ($a = \pi(s)$), where, given a state, only one specific action can be taken, or stochastic policies, where the action is chosen based on some distribution calculation between the actions and the given state ($\pi(a|s) = P(A_t = a|S_t = s)$).

2.2 Bellman Equations

Up to now, we have just outlined the basic components of the RL framework. To explain how the agent actually "learns", we need to introduce the so-called Bellman Equations [18], which can be considered as the guidelines followed by the agent to reach its goal: maximizing the expected return. The Bellman equations formulate this objective in terms of a recursive relationship with the value function.

A policy π is considered better than another policy π' if the expected return of that policy is greater than that of π' for all $s \in S$, which implies $V^{\pi}(s) \geq V^{\pi'}(s) \quad \forall s \in S$.

Therefore, the optimal value function $V^*(s)$ can be defined as:

$$V^*(s) = \max_{\pi} V_{\pi}(s), \quad \forall s \in S \tag{2.5}$$

Similarly, the optimal action-value function $Q^*(s, a)$ can be defined as:

$$Q^*(s, a) = \max_{\pi} Q_{\pi}(s, a), \quad \forall s \in S,\ a \in A \tag{2.6}$$

Moreover, for an optimal policy, we can define the equation:

$$V^*(s) = \max_{a \in A(s)} Q^{\pi^*}(s, a) \tag{2.7}$$

Expanding equation (2.7) by using (2.4), we obtain:

$$
\begin{aligned}
V^*(s) &= \max_a \mathbb{E}_{\pi^*}(G_t | s_t = s, a_t = a) \\
&= \max_a \mathbb{E}_{\pi^*} \left(r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right) \\
&= \max_a \sum_{s'} p(s' \mid s, a) \left[r_t + \gamma V^*(s') \right] \qquad (2.8)
\end{aligned}
$$

which is commonly known as the Bellman Optimality equation for $V^*(s)$. For Q^*, instead, the optimality equation is given by:

$$
\begin{aligned}
Q^*(s, a) &= \mathbb{E}(r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a), \\
&= \sum_{s'} p(s' \mid s, a) \left[r_t + \gamma \max_{a'} Q^*(s', a') \right] \qquad (2.9)
\end{aligned}
$$

Here, $Q^*(s, a)$ is defined recursively based on equations (2.4) and (2.6).

If the transition probabilities and reward functions are known, the Bellman optimality equations can be solved iteratively. It is the case of model-based algorithms, which exploit *dynamic programming* to compute the optimal policy.

Conversely, many other algorithms assume that probabilities are unknown and must be estimated through policy and value function rollouts, which involve applying one or more simulation steps to test possible outcomes. These methods are known as model-free algorithms. Monte Carlo, Temporal Difference, and Policy Search are the most commonly used model-free paradigms. The remainder of this chapter will focus on this latter class of methods to introduce important algorithms such as Q-learning and, in the subsequent section, Deep Q-Learning.

2.3 Model-Free Methods

Model-free methods can be applied to several RL problems without requiring any prior knowledge of the environment's dynamics. This means that the agent does not need to know the transition probabilities or the reward function, but it learns the optimal strategy by interacting with the environment.

There are two main categories of model-free algorithms:

- Value-based methods: aim to learn the value function and derive the optimal policy from it; an example is Q-learning, which constitutes the basis of the algorithm used in this thesis and will therefore be explored more in depth shortly.
- Policy-based methods: directly optimize the policy without estimating the value function explicitly; examples include policy gradient methods.

These approaches can also be classified as either on-policy (e.g.: SARSA) or off-policy methods (e.g.: Q-Learning). On-policy methods use the current policy both to generate actions and to update the same policy . In contrast, off-policy methods use two policies: one that is learned about and that becomes the optimal policy - the *target policy* - and one that is more exploratory and is used to generate behavior - the *behavior policy*.

2.3.1 Monte Carlo Methods

Monte Carlo (MC) methods are based on the idea of Generalized Policy Iteration (GPI), an iterative framework consisting of two processes. The first process attempts to approximate the value function based on the current policy (policy evaluation step). In the second step, the policy is improved with respect to the current value function (policy improvement step). In Monte Carlo methods, the value function is estimated using the rollout technique, where the current policy is executed in the system. The value function is then updated based on the accumulated reward over the entire episode and the distribution of encountered states. The current policy is refined using a greedy technique. By iterating between these two steps, the algorithm is proven to converge to the optimal value function and policy. Although MC methods are simple to implement, they require many iterations to converge and suffer from high variance in value function estimation.

2.3.2 Temporal Difference Methods

Temporal Difference (TD) methods are still built on the idea of Generalized Policy Iteration (GPI), but differ from MC methods in the policy evaluation step. Instead of using the total sum of rewards, TD methods calculate the temporal error, which is the difference between the new estimate and the old estimate of the value function, considering the reward received at the current time step and using it to update the value function.

The value function update equation is given by:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{2.10}$$

Algorithm 2.1 Q-learning for MDPs (with ε-greedy policies)

Initialize: $Q(s, a) = 0$ for all $s \in S, a \in A$
Repeat for every episode:
for $t = 0, 1, 2, \ldots$
    Observe current state s_t
    With probability ε: choose random action $a_t \in A$
    Otherwise: choose action $a_t \in \arg\max_a Q(s_t, a)$
    Apply action a_t, observe reward r_t and next state s_{t+1}
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right]$
end for

where α is the learning rate, r is the reward received at the current time step, s' is the new state, and s is the previous state. Therefore, TD methods update the value function at each time step, unlike MC methods, which wait for the episode to complete before updating the value function. This type of update reduces variance but introduces the bias in value function estimation.

Therefore, TD algorithms can learn before the final outcome, allowing us to work with incomplete sequences and in continuing environments. MC methods could work only with episodic task, making them unsuitable for our project; hence, we'll focus now on the most characteristic TD algorithm: *Q-learning*.

Q-Learning is an off-policy temporal difference algorithm. Unlike SARSA, Q-Learning is off-policy because it directly approximates Q^*, independently of the policy being followed. An experience is defined as (s, a, r, s'), where the agent starts in state s, takes action a, receives a reward r, and transitions to a new state s'. The update to $Q(s, a)$ is performed by receiving the maximum possible reward from an action in s' and applying the following update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.11}$$

The algorithm is described in Algorithm 2.1.

The simplest method for storing the values of the Q-function for each state-action pair is the tabular form; however, this approach has some limitations. If the state and action spaces are very large, it becomes impossible to store all the values in a tabular format. The reason is straightforward: the memory required to save all this data is too vast. Additionally, even searching the table for a value in a particular state-action pair can be computationally prohibitive. Another limitation arises from the state space itself: if the space is continuous, it will be impossible to

use the tabular form unless the states are discretized. For these reasons, the tabular format is only applied to environments with a limited number of states and actions.

To overcome these issues, function approximators have been introduced to store the values of the Q-function. In this case, the Q-function is parameterized by a vector $\theta = (\theta_1, \theta_2, \ldots, \theta_n)^T$ and is denoted as $Q(s, a; \theta)$. The function approximator can be thought of as a mapping from the vector θ in $\mathbb{R}^n$ to the space of the Q-function. As long as the number of parameters in the approximator is less than the number of state-action values, changing the value of a certain parameter will affect the Q-function in multiple regions of the state-action space; this helps function approximators achieve better generalization in fewer training steps.

There are various methods in Reinforcement Learning for function approximation. This thesis will focus on the most relevant ones, i.e. *neural networks*, and on their application within the RL framework.

2.4 Deep Reinforcement Learning

With the term Deep Reinforcement Learning we refer to the exploitation of deep neural networks as function approximators for the value function - or the policy - in RL algorithms.

Since it's out of the scope of this thesis, we are not going to talk about general neural networks functioning. Instead, we will directly focus on their application within RL and why they are so essential for our work.

2.4.1 Deep Q-Learning

The Deep Q-Learning algorithm is an important evolution of the previously described Q-Learning algorithm. As stated above, the tabular approach used by Q-Learning makes it computationally infeasible when we have to deal with high-dimensional or continuous state space, as the one required in our project. Deep Q-Learning addresses this limitation by using a neural network to approximate the Q-value function, granting the ability of generalizing over such state spaces. The tabular version is therefore substituted with a neural network, which - given as input the state - outputs the estimated value for each possible action.

The loss now is defined as:

$$L_t = (y_t - Q(s_t, a_t; \theta))^2 \tag{2.12}$$

where:

Algorithm 2.2 Deep Q-Learning

Initialize: Neural network $Q(s, a; \theta)$ with random weights θ
for each episode
    Observe initial state s_0
    for $t = 0, 1, 2, \ldots$
        With probability ε: choose random action $a_t \in A$
        Otherwise: choose action $a_t \in \arg\max_a Q(s_t, a; \theta)$
        Apply action a_t, observe reward r_t and next state s_{t+1}
        Update the Q-value:
            $Q(s_t, a_t; \theta) \leftarrow Q(s_t, a_t; \theta) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) - Q(s_t, a_t; \theta) \right]$
    end for
end for

$$y_t = \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{otherwise} \end{cases} \tag{2.13}$$

The pseudocode is written in the algoritmh 2.2.

Despite the capability of handling high-dimensional/continuous spaces, this algorithm suffers two other important issues:

- **Moving target problem**
  The generalization capabilities offered by neural networks come with a downside. The constant modification of the agent's policy during training and the exploitation of bootstrapped target values, make the learning of the Q-value function quite challenging. This non-stationarity problem is worsen in deep RL, where updating the value for one state may unpredictably alter values for others (generalization), destabilizing even more the optimization.

- **Correlations**
  Taking consecutive steps as experience to train the agent generates a problem of correlation. This breaks the assumption of using i.i.d. (indipendent and identically distributed) data to train the function approximation, which may eventually lead to oscillating or diverging policy.

To address these challenges, we can implement two important features: the *target network* and the *experience replay*. The combination of these ideas brings us to one of the most relevant RL algorithm: Deep Q-networks (DQN) [6].

2.4.2 Deep Q-Networks

DQN can be considered a step forward from deep Q-learning, as it incorporates a couple of techniques to tackle the issues that affected the latter.

These methods are the two already introduced in the previous subsection, and they are explained as follows.

- **Target Network**
  To mitigate the instability in training associated with the moving target problem, we can implement an additional network known as *target network*. The latter is a periodic copy of the online network: they are initialized with the same weights, but the target network's weight are updated only after τ steps, and set equal to the online one. This "new" network can then be used in place of the main Q-value function to compute bootstrapped target values; thanks to this approach, the target values are close to the estimates of the main Q-value function while fixed for a discrete amount of steps, which has been proven to increase the stability of the learning process.

- **Experience Replay**
  With this technique, the agent's experience $e_t = (s_t, a_t, r_t, s_{t+1})$ is taken at each time step t and stored in a dataset $\mathcal{D} = \{e_t, e_{t+1}, \dots, e_n\}$, called replay buffer. Training is performed using the mini-batch technique, where a subset of experiences $\mathcal{B}$ is sampled uniformly at random from this buffer, $\mathcal{B} \sim \mathcal{U}(\mathcal{D})$. The use of this technique allows past experiences to be utilized in more than one network update. Additionally, the random selection of the subset from the replay memory helps break the strong correlation between consecutive experiences, thus reducing variance in the updates.

The MSE loss now is computed over the mini-batch $\mathcal{B}$:

$$L(\theta) = \frac{1}{B} \sum_{(s_k, a_k, r_k, s'_k) \in \mathcal{B}} \left(y_k - Q(s_t, a_t; \theta)\right)^2 \tag{2.14}$$

where the target y_k - related to the k_{th} experience - is computed as previously (see equation 2.13). Then, the code can be formulated as in the algorithm 2.3.

2.4.3 Double DQN

Even the standard DQN showed some flaws. Research has shown that using the same network both to select and evaluate actions could lead the algorithm to overestimate Q-values. To address this issue, an enhanced version was proposed by Van Hasselt et al. (2016) [7]: the *Double DQN*.

Algorithm 2.3 Deep Q-networks (DQN)

Initialize: online network with random parameters θ
Initialize: target network with parameters $\bar{\theta} = \theta$
Initialize: an empty replay buffer $\mathcal{D} = \{\}$
repeat for every episode:
    for time step $t = 0, 1, 2, \ldots$
        Observe current state s_t
        With probability ε: choose random action $a_t \in A$
        Otherwise: choose $a_t \in \arg\max_a Q(s_t, a; \theta)$
        Apply action a_t; observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
        Sample random mini-batch of B transitions (s_k, a_k, r_k, s_{k+1}) from $\mathcal{D}$
        if s_{k+1} is terminal
            $y_k \leftarrow r_k$
        else
            $y_k \leftarrow r_k + \gamma \max_{a'} Q(s_{k+1}, a'; \bar{\theta})$
        end if
        Loss $\mathcal{L}(\theta) \leftarrow \frac{1}{B} \sum_{k=1}^{B} \left(y_k - Q(s_k, a_k; \theta)\right)^2$
        In a set interval, update target network parameters $\bar{\theta} \leftarrow \theta$
    end for
until convergence

This algorithm employs the aforementioned networks - the *online network* and the *target network* - to compute the target value: the former is used to select the action, while the latter is used to estimate its value. This decoupling ensures that the selection of the best action is independent of its evaluation, thus mitigating the overestimation bias.

The target becomes now:

$$y_t = \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma Q\left(s_{t+1}, \arg\max_{a'} Q\left(s_{t+1}, a'; \theta_t\right); \bar{\theta}_t\right) & \text{otherwise} \end{cases} \tag{2.15}$$

Where:

- θ are the parameters of the *online network*, used to select the best action a via $\arg\max$.
- $\bar{\theta}$ are the parameters of the *target network*, used to evaluate the Q-value of the action selected by the online network.

Subsequently, the pseudo code of the algorithm used in this work is presented.

Algorithm 2.4 Double DQN

Initialize: online network with random parameters θ
Initialize: target network with parameters $\bar{\theta} = \theta$
Initialize: an empty replay buffer $\mathcal{D} = \{\}$
repeat for every episode:
    for time step $t = 0, 1, 2, \ldots$
        Observe current state s_t
        With probability ε: choose random action $a_t \in A$
        Otherwise: choose $a_t \in \arg\max_a Q(s_t, a; \theta)$
        Apply action a_t; observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
        Sample random mini-batch of B transitions (s_k, a_k, r_k, s_{k+1}) from $\mathcal{D}$
        if s_{k+1} is terminal
            $y_k \leftarrow r_k$
        else
            $y_k \leftarrow r_k + \gamma Q(s_{i+1}, \arg\max_{a'} Q(s_{i+1}, a'; \theta); \bar{\theta})$
        end if
        Loss $\mathcal{L}(\theta) \leftarrow \frac{1}{B} \sum_{k=1}^{B} (y_k - Q(s_k, a_k; \theta))^2$
        In a set interval, update target network parameters $\bar{\theta} \leftarrow \theta$
    end for
until convergence

3
Proposed Method

In this chapter we will focus on the methodological approach adopted to solve the Traffic Light Control (TLC) problem at a road intersection. Our aim is to develop an agent capable of reducing traffic congestion through the optimization of traffic light phases by leveraging RL techniques.

We will then describe the key concepts of the RL framework used, including the definition of states, actions and reward functions, and how these elements interact to guide the agent's behavior. We will avoid going into the technical details relating to the simulator or the experimental setup, which will be covered in the subsequent chapter.

The final part will be then dedicated to the different environmental conditions used to train our models: from the variation of traffic levels, to the different time intervals allocated to the decision-making process of the agent, to the oscillating importance attributed to pedestrian and vehicle flows.

3.1 Network building

The principle of the project was to create a fair and scalable model that could be applied across various scenarios. This flexibility would allow the agent to be deployed in many environments, whether in an elementary single intersection or in a more complex urban network, where many roads intersect.

In this project we opted for a standard two-way intersection, where each road is divided into three lanes, one for each driving direction (see Fig 3.1). Technical aspects will be outlined in chapter 4.

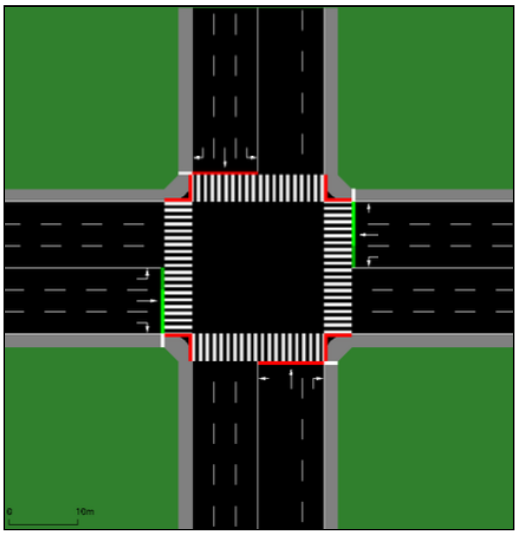


Figure 3.1: Road Network - Image generated with SUMO

3.2 Environment definition

For the environment we considered some of the different approaches presented in literature (section 1.4), and eventually selected the combination that best met our requirements.

3.2.1 State Space

Regarding the state space, we tried to capture simple but efficient information from all relevant "entities": from the vehicles situation, as the numbers and position, to the pedestrian flow, and eventually to the traffic light configuration, such as timing and phase. This was necessary to provide the agent with a comprehensive view of the current situation in the network.

As a matter of fact, we used five features that could gather the most from the environment; inspiration was taken from [9]:

- *Traffic Light Phase*
  It indicates the currently active traffic light phase. Together with the Elapsed *Green Time*, it gives a complete description of the actual TL configuration.

- Elapsed *Green Time*
  It measures the time that has elapsed since the traffic light turned green. This is a continuous variable, providing information about the duration the green light has been active.

- *Density*
  It indicates the proportion (i.e.: $\in [0,1]$) of the lane's capacity currently occupied by vehicles for each edge. It is computed for each lane in the network.

- *Queue length*
  It represents the length of the vehicle queue* in each lane, normalized to a range between 0 and 1.

- Number of *Pedestrians*
  As the name implies, it's a scalar integer value representing the number of pedestrians currently waiting in the network. As soon as they leave the pedestrian crossing, they are removed from the environment.

All these features are simple in the sense that they do not require any advanced sensor in order to be gathered. Indeed, thanks to modern technologies, they are straightforward to obtain in almost any real world scenario.

The first two features - i.e. the current TL phase and the elapsed green time - can be easily retrieved by interfacing with the traffic management infrastructure, since they are standard parameters and as such are continuously monitored. The remaining features - i.e. density, queue length and number of pedestrians - can be captured using advanced camera systems; the latter, leveraging computer vision techniques, are able to accurately detect both vehicles and pedestrians in real-time. Furthermore, we emphasize that this setup does not require invasive or specialized sensors, as cameras provide a non-intrusive and cost-effective solution for continuous monitoring of the traffic network.

In conclusion, the features exploited in the simulation are fully compatible with real-world implementations, ensuring a smooth transition from theoretical models to practical applications.

3.2.2 Action Space

Concerning the possible actions undertaken by the agent, we decided to adopt a straightforward logic that could be easily scaled to more complex future scenarios.

*A vehicle is said to be in queue when its speed is below an imposed limit

In the first stage of the project, the action space comprised only a binary choice: the agent could either maintain the current phase, in case the traffic flow being managed was still significant, or change to the next one. Then, the transition from one green phase to the next one was preceded by a yellow light phase, which is immutable and always present.

With the implementation of pedestrians, we had to increase the number of phases so as to allow their passage. By transitioning from a binary TL configuration to a triple one, we created the opportunity not only to alternate among phases, but also to choose the order in which to activate them. Consequently, we have expanded the action space to three, allowing the agent to choose which phase to trigger.

The underlying logic is similar to the previous one: the agent can either maintain the phase - simply by picking the current one - or switch to one of the others. This approach is in line with the scalability approach we mentioned earlier: the number of phases/actions can easily be increased depending on the characteristics of the junction considered.

This method ensures flexibility, since it allows the model to adapt to a wide variety of scenarios. For instance, it can provide extended and more frequent green phases to a specific road in case of unusual circumstances that result in significantly higher traffic flow on that route; conversely, it can even "shut down" the opposite road in response to sudden issues.

3.2.3 Reward Functions

First Stage

In the first stage of the project, we considered only the passage of vehicles in the network. Hence, we only monitored their behavior throughout the simulation. Three different reward functions have been implemented, each serving a specific purpose to optimize traffic management.

1. *Waiting Time*
   The first reward function quantifies the total time vehicles spend waiting (i.e.: when speed drops below a specified threshold) at traffic signals or intersections. By minimizing waiting time, the function aims to enhance traffic flow efficiency. We consider the difference in waiting time between consecutive episodes: this incentivizes the agent to minimize the current waiting time with respect to the previous one, and consequently to maximize the associated reward.

2. *Queue Length*
   The second reward function assesses the length of vehicle queues at the intersection. The

goal is to minimize queue length to alleviate congestion and prevent the accumulation of long lines of vehicles. With the reward defined as the negative of the queue length, the agent is committed to reduce queue sizes.

3. *Total Delay*
   The last reward function calculates the cumulative delay experienced by vehicles across the network, incorporating waiting times and additional delays caused by congestion. It is defined as the variation from the maximum speed allowed, and it's computed as:

$$Delay_{veh_i} = (speed_{max} - speed_{veh_i})/speed_{max}$$

   By minimizing average delay, the agent seeks to improve travel time reliability and efficiency. Even in this case, we took as reward the opposite of the total delay.

Second Stage

With the introduction of pedestrians, we had to adjust the reward function to enable the agent to consider also their state during the decision-making process.

The standard formulation of the reward function becomes:

$$Reward = -[(1 - \beta) * R_{veh} + \beta * R_{ped}] \tag{3.1}$$

As we can note, the function now comprehends a convex combination of the rewards associated, respectively, with the vehicle (i.e.: R_{veh}) and the pedestrian (i.e.: R_{ped}) flow. This grants the model the ability of balancing the importance between the two components, based on the weighting factor β. Additional details will be provided in subsection 4.4.3.

Concerning pedestrians, only the waiting time was took into consideration. The reasoning is the same applied previously: taking the negative value of the reward logically pushes the agent to reduce pedestrians' waiting time.

We add a brief comment on the implementation of these functions.

To operate effectively, they necessitate to compute additional information with respect to the features defined in the state space (see 3.2.1); we are referring to the waiting time of the objects and the speed of vehicles, useful to compute the overall network delay. These characteristics can be classified as "dynamic features", given the fact that they require tracking the entities' behavior over time: speed can be computed by analyzing the time interval between consecutive

frames, while waiting time requires also to identify when a vehicle or pedestrian is stationary, and recording the duration until they start moving again.

Although harder than static feature detection, this data can still be obtained thanks to advancements in computer vision techniques. It would be essential, however, the installation of high-resolution cameras with a sufficient frame rate, so as to ensure a precise tracking over time of the objects in the environment; additionally, these systems must be capable of recording and processing multiple streams of information simultaneously and for extended periods.

In conclusion, while the complexity of dynamic feature detection is undoubtedly higher than that of static feature detection, it remains manageable with state-of-the-art techniques.

3.3 Scenarios

The main advantage of RL based traffic lights is their ability of dynamically adapting to real-time traffic conditions. For this reason, we decided to test our algorithm under different dynamics.

3.3.1 Varying Traffic Conditions

The first situation involves the modification of the traffic levels on the two roads. We implemented 12 different vehicle inflows; each of these flows presents the same type of vehicle, with equal generation condition (i.e.: lane and speed options, as will be explained in section 4.2.2). Despite this, we decided to diversify the generation rate of the vehicles based on two different perspectives. The first perspective involves differentiation *by direction*. To ensure the creation of a flexible and diversified model, we opted to generate a larger volume of vehicles for the north-south route and a smaller volume for the opposite direction; this can be easily related to a real-world scenario, where a major road intersects with another one of lower relevance. The second regards differentiation *by magnitude*; it was introduced to create varying traffic levels, which, as we know, can fluctuate significantly in reality.

3.3.2 Varying Decision-Making Frequency

In this second scenario, we altered the time interval after which we perform an action-selection step. We wanted to analyze the balance between efficiency and computational costs required to deploy our model. Clearly, the adoption of a shorter window will create a more tailored division of time among the different phases, providing vehicles with a smoother experience.

On the other hand, the model should work several times more frequently with respect to "less responsive" models, increasing the operational cost of the algorithm.

Considering higher intervals, in fact, reduces the computational demands, leading to a potentially more cost-effective model in low-traffic scenarios. Vice versa, the problem here would be represented by rapidly changing traffic conditions, which could lead to unexpected congestion.

A moderate intervention will be built as a potential balance, aiming to provide sufficient reactivity while maintaining reasonable computational efficiency.

By analyzing these different scenarios, we aim to identify the optimal action-selection frequency for maintaining traffic flow without incurring unnecessary operational costs.

The exploitation of higher frequencies might cause excessively frequent phase switching, potentially causing system instability and unsustainable behavior. For this reason, a penalty mechanism will be introduced and adjusted to the corresponding time intervals; further details are provided in 4.3.2.

3.3.3 Balancing Priority between Vehicles and Pedestrians

The third and most relevant scenario explores the weights variation between vehicles and pedestrians, hence introducing the concept of fairness as a key factor in the traffic management framework.

To accomplish this, we introduced weights that influence the priority level assigned to vehicles versus pedestrians during traffic light decisions. Recalling back the used reward function (subsection 3.2.3), we multiplied both terms by a factor related to β: the higher β, the more importance is placed on pedestrian flow; logically, the lower β, the higher the importance placed on vehicle flow.

This approach enables the model to respond to context-specific demands: for example, by assigning a higher weight to pedestrians in areas near schools or shopping centers, where foot traffic peaks. Conversely, during times of heavy vehicle flow, especially during rush hours, the model can shift the weight toward vehicles to prevent excessive road congestion.

By adjusting these weights, we aim to evaluate the model's ability to maintain fairness across a range of real-world traffic scenarios. Through this approach, we can study the model's ability to balance flow efficiency and fairness, creating a traffic management system that aligns with the need of all road users.

4
Experimental Setup

This chapter is dedicated to the the technical side of the project.

Firstly, we will better outline the simulation environment and how we translated the real-world problem into a virtual framework. Subsequently, an in-depth review of the RL environment will be provided. The last part will be dedicated to the discussion of the algorithm, including the tuning of the hyperparameters, the neural networks used and the replay memory specifications.

4.1 Problem definition

In this section we are going to formulate the Traffic Light Control (TLC) problem as an RL task, by introducing the key components needed to define it:

- Physical/Simulation environment (1)
- RL environment (2)
- Agent (3)

The associated numbers refers to the relative section in Figure 4.1, where an abstract representation of the interplay between the aforementioned entities is displayed.

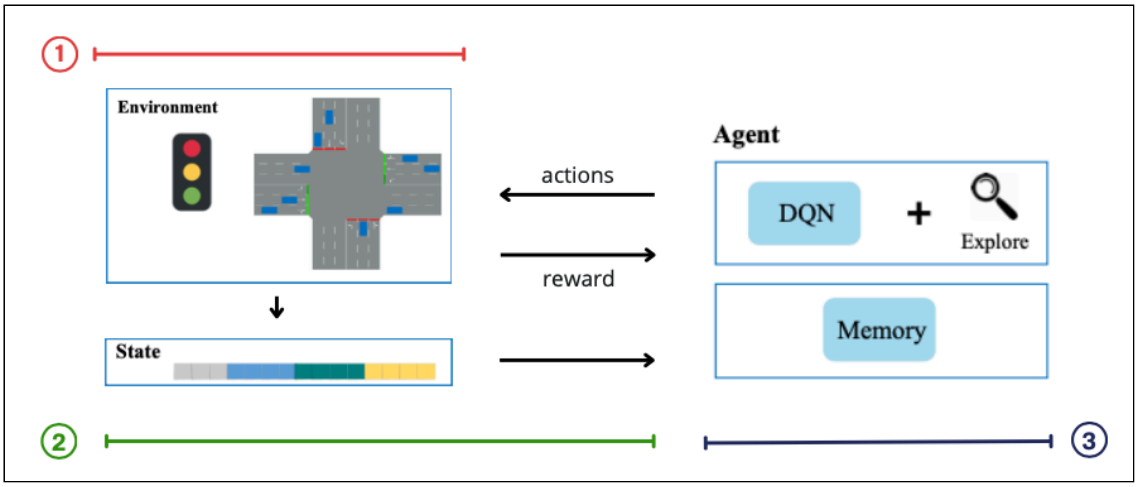


Figure 4.1: Deep Reinforcement Learning Framework for Traffic Light Control - Image adapted from [11]

4.2 Simulation Environment

To simulate a real-world scenario we used SUMO (Simulation of Urban MObility) [19], an open source traffic simulation platform designed for handling different road network configurations. In addition to that, we used Flow [20], a python library built on top of the SUMO traffic simulator and OpenAI Gym [21] to facilitate the implementation of RL algorithms within traffic environments. Basically, Flow acts as a bridge between SUMO and the RL model we are going to build, train and test.

4.2.1 Road Network

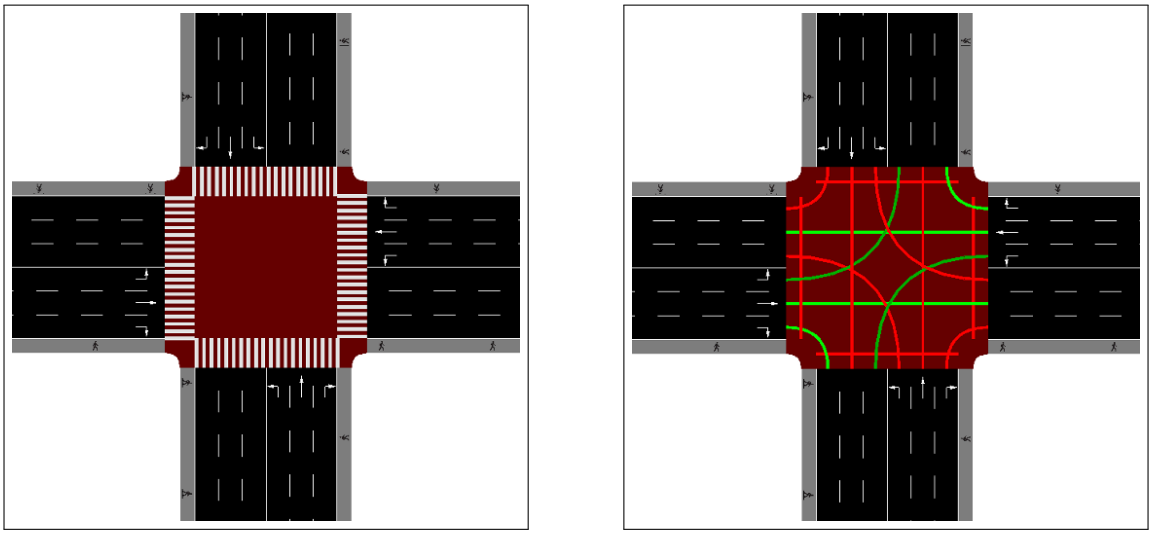
Our case study is based on a multi-lane two-way road intersection, where each road is divided into three lanes, one for each driving direction (Figure 4.2(a)). It has been built artificially using *netedit*, a graphical network editor included in SUMO.

The principal features we had to define are:

- *Nodes* - Fixed knots useful for dividing in space the network. The main attributes include the x and y coordinates.
- *Edges* - The edges are the actual streets of our network. The main attributes are the origin/destination nodes and the number of lanes.
- *Routes* - The routes are the sequence of edges vehicles can traverse given their position. The main attributes include the sequence of edges (i.e.: the route) and the probability to follow each sequence.

Additional elements, yet essential for our work, are:

- *Pedestrian Crossings* - intuitively, they represent the crossing areas dedicated to pedestrians. They are placed perpendicularly to edges, crossing all the lanes right after the stop line for vehicles.
- *Traffic Light* - the core of our project; the main attributes include the different light configurations (i.e. tl phases) and the corresponding time interval.



(a) Intersection Modeling

(b) Lane connections

Figure 4.2: Visual Representations of the Considered Intersection - Images generated with SUMO

On Figure 4.2(b), a focus on the internal lane connections is provided. We notice two characteristics:

- **Lane Connections**
  Each road is divided into three lanes, one for each driving direction. The rightmost lane is dedicated to vehicles turning right, the center lane is for those continuing straight, and the left lane is reserved for vehicles turning left and therefore crossing the intersection. In this last case, cars must yield to those going straight, as they have traffic on their right.
- **Flow Direction**
  In the figure, we can observe through the colored lines that one of the possible phases allows vehicle flow in the west-east direction. An additional detail is provided by the varying shades of green: the third lane is marked with a darker green, indicating that vehicles traveling in that direction must yield the right of way (as they need to turn left

and will encounter oncoming traffic from the opposite direction). The other phases logically correspond to the flow in the north-south direction, while the remaining phase is dedicated to pedestrian crossings in all directions.

This phase configuration was chosen due to a limitation in the current version of Flow: pedestrians have not yet been fully implemented in the library. To work around this, we created "virtual" pedestrians that are not actually generated in the simulator. Consequently, if we had allowed both pedestrians and vehicles to have the green light simultaneously, the vehicles would not be able to yield to the pedestrians properly. To avoid this inconsistency, we opted for phases that prevent overlap between vehicle and pedestrian movement.

Therefore, the phases are six in total, since we need to consider also the yellow light transition. For safety and compliance with current regulations, we have set the duration of the yellow traffic light to 5 seconds. For the same reasons, we set also a minimum duration of 5 seconds for the green phases.

We outline the sequence of phases:

1. `GGgrrrGGgrrrrrrr` - Green for vehicles from/to North-South, red for others
2. `yyyrrryyyrrrrrrr` - Yellow for vehicles from/to North-South, red for others
3. `rrrGGgrrrGGgrrrr` - Green for vehicles from/to East-West, red for others
4. `rrryyyrrryyyrrrr` - Yellow for vehicles from/to East-West, red for others
5. `rrrrrrrrrrrrGGGG` - Green for Pedestrians, red for others
6. `rrrrrrrrrrrrYYYY` - Yellow for Pedestrians, red for others

Note that the sequence is composed by 16 letters: the first 12 are allocated for vehicle lanes - 3 for each road - and the last 4 for pedestrian lanes. The meaning of each letter is easily reconducible to the different colors of the traffic lights, while the uppercase *"G"* is used when vehicles do have the right of way.

4.2.2 Inflows

To simulate traffic, Flow provides a functionality to generate vehicles within the network during the simulation. The class managing such process is called InFlows; an instance of it will then be provided as an input during the network creation process.

In order to add a specific flow we need to define:

- *Vehicle type*: whether human-driven or RL-controlled. Additional parameters include the controllers, useful for determining the behavior of vehicles. Being out of the scope of the thesis, we will not discuss further this section; more details can be found in [20].
- *Departure*: the spot in which the vehicle is generated. It consists of edge and lane; the former is chosen among the four possible edges, while the latter is set to "free", meaning vehicles occupy the rightmost available lane on the selected road.
- *Speed*: the speed at which the vehicles will enter the network. We set this value to 0.
- *Inflow rate*: how many vehicles will be added into the network.

In our project, we implemented 12 different traffic flows, 3 for each cardinal point of origin. More details will be provided at the end of the chapter (subsection 4.4.1).

For what concern pedestrians, as mentioned previously (see subsection 4.2.1), we created them outside the simulator. To do that, we defined a fixed hourly rate, diversified only by direction: 350×2 pedestrians for east-west direction, 500×2 pedestrians for north-south direction. For the removal, we decided to subtract 30% of them, with a maximum of 3, for each second in which the traffic light is green.

4.3 RL Environment

4.3.1 State Space

The state space comprehends five features, as listed in section 3.2.1. A representation is provided below.

$$State = \begin{bmatrix} Phase, Green\ Time, Density = \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \\ d_{41} & d_{42} & d_{43} \end{bmatrix}, Queue = \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \\ q_{41} & q_{42} & q_{43} \end{bmatrix}, Pedestrians \end{bmatrix}$$

- *Traffic Light Phase*
  One-hot encoded: we transformed the categorical variable corresponding to the phase into a binary vector, where each phase is represented by a vector with a 1 in the position corresponding to the current phase and 0s elsewhere. As a result, it is a vector of size equal to the number of phases, 6.

- Elapsed *Green Time* $\rightarrow x \in \mathbb{R}^+$
- *Density* $\rightarrow D \in \mathbb{R}^{4\times3}, \quad 0 \leq d_{ij} \leq 1 \quad \forall i \in \{1,2,3,4\},\ j \in \{1,2,3\}$
- *Queue length* $\rightarrow Q \in \mathbb{R}^{4\times3}, \quad 0 \leq q_{ij} \leq 1 \quad \forall i \in \{1,2,3,4\},\ j \in \{1,2,3\}$
- Number of *Pedestrians* $\rightarrow x \in \mathbb{N}$

Density and Queue Length are computed over a pre-determined part of the road, in our case equal to 75 meters (computed considering a maximum of 10 cars plus the safety space between vehicles). This factor will depend on the specific technology adopted, hence could be diminished or enlarged at will.

4.3.2 Reward Function

Recalling the equation formulated in Chapter 3, we introduce a new component: a penalty term, useful to prevent the agent from switching phases too frequently. The adjusted equation is formulated as follows:

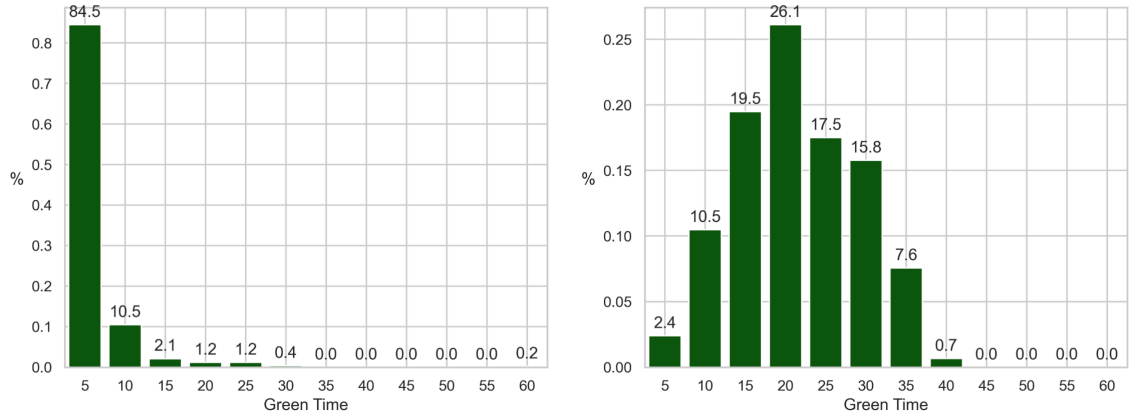
$$Reward = -[(1-\beta) \times R_{veh} + \beta \times R_{ped}] + pen \tag{4.1}$$

Depending on the function and on the time interval dedicated to the action selection, the reward is adjusted with a negative value. The functioning is simple but effective: the last n actions are checked, where n depends on the time span employed in the decision-making process; if all the phases are present, the penalization is considered. The penalty assumes the following values:

1. waiting time: 10 for 15 sec. window, 20 for 10 sec, 60 for 5 sec.
2. queue length: 20 for 15 sec. window, 40 for 10 sec, 100 for 5 sec.
3. delay: 30 for 15 sec. window, 60 for 10 sec, 250 for 5 sec.

The number of actions checked - n - is set to 3 for $10''$ and $15''$ intervention models, and increased to 5 for 5" models. The time span for a rapid intervention model covers logically a fraction of the time employed by moderate/slow intervention models, hence the enlarged window. Additionally, the weight is exponentially increased: our aim is to avoid several phase switches within a short amount of time.

The effects of this measure can be observed in Figure 4.3; the average traffic light duration has increased from 5 seconds to 20, with a much more uniform distribution and coherence within the proposed framework.



(a) Without penalty

(b) With penalty

Figure 4.3: Green Time Length for Moderate Traffic - 5sec scenario

4.4 Scenarios

The main advantage of RL based traffic lights is their ability of dynamically adapting to real-time traffic conditions. For this reason, we decided to test our algorithm under varying dynamics.

4.4.1 Varying Traffic Conditions

The first situation involves the modification of the traffic levels on the two roads. Vehicles traveling from north and south routes belong to the main road; oppositely, vehicles coming from east and west directions belong to the secondary road.

We used six different flows, which can be classified as:

1. *Light flow*: 400×2 vehicles on the secondary road, 750×2 vehicles on the main road
2. *Moderate flow*: 500×2 on the secondary road, 850×2 on the main road
3. *Heavy flow*: 600×2 on the secondary road, 1000×2 on the main road

For what concern pedestrians, we decided to use a single rate of flow to avoid introducing excessive complexity into the analysis. It consists of:

1. *Standard flow*: 300×2 pedestrians on the secondary road, 500×2 pedestrians on the main road

The training procedure involved in this scenario is slightly more complex than the subsequent ones; this because we wanted to create a single model able to face simultaneously the aforementioned scenarios. To achieve this, we alternated the different flows while training, and created a larger, more stable replay buffer by increasing the memory size to 20.000 and raising the batch size to 256. We trained the model for 1000 episodes, each lasting 2.000 time steps (1 time step = 1 second), for a total of $2e6$ time steps.

4.4.2 Varying Decision-Making Frequency

In this second scenario, we altered the time interval after which we perform an action-selection step. The proposed solutions are:

1. *Rapid intervention*: 5 seconds
2. *Moderate intervention*: 10 seconds
3. *Slow intervention*: 15 seconds

Here the proposed models are different from each other, and individually trained under the specified frequencies. We trained them independently for a total of 300 episodes, still for 2000 steps each. Replay buffer was reduced at 10.000 and batch size to 128.

4.4.3 Balancing Priority between Vehicles and Pedestrians

The third scenario explores the weights variation between vehicles and pedestrians. The configurations considered are the following:

1. $\beta = 0.4 \rightarrow 40\% Rew_{ped} + 60\% Rew_{veh}$
2. $\beta = 0.5 \rightarrow 50\% Rew_{ped} + 50\% Rew_{veh}$
3. $\beta = 0.6 \rightarrow 60\% Rew_{ped} + 40\% Rew_{veh}$

The training here followed the same approach used in 4.4.2.

4.5 Algorithm details

This section is dedicated to discuss in more details the characteristics of the algorithm introduced in the subsection 2.4.3, along with the selection of the different hyperparameters.

Network Architecture

The neural network used contains three fully connected layers: the first layer has 64 neurons, the second 32, and the third has 3, one for each possible action. Leaky ReLU activation functions are applied to the hidden layers to introduce non-linearity and prevent the dying neurons problem. The final output layer remains a linear transformation, providing raw Q-values for each action.

Learning phase

Based on the decision-making interval provided to the agent, the learning step is performed after $5, 10, 15$ or 20 time steps; these values derive from the three different frequency intervals plus the potential 5 seconds of the yellow light that precede a phase switch. The network employs RMSprop for optimization and mean squared error (MSE) as loss function.

The policy used is an elementary yet effective ϵ-greedy strategy: at each iteration, the agent has a $(1 - \epsilon)$ percentage of going greedy - hence selecting the action proposed by the current policy - or choosing a random action with probability ϵ. The first option ensures exploitation, while the second exploration. At first, exploration is essential for our agent to avoid local optima and keep learning; eventually, we prefer a more stable approach. For this reason, we do not use a static ϵ but we implement a decay over time; we set $\epsilon = 0.5$, linearly decreasing up to 0.001 with ϵ_{decay} based on the number of training steps.

Memory

At each "learning step", we store in memory the tuple $<s^t, a^t, r^t, s^{t+1}>$. Whenever the buffer is filled with enough knowledge (at least equal to one batch), the training is actually performed: a batch (size equal to 128 or 256) is retrieved and used to train the model. The total size of the buffer is set to 10.000; for "larger" models, it is doubled to 20.000 (see 4.4.1).

Hyperparameter	**Value**	**Hyperparameter**	**Value**
$learning\,rate$	0.001	τ	250
ϵ	$0.5 \rightarrow 0.001$	γ	0.99
$memory\ size$	$10k, 20k$	$batch\ size$	$128, 256$

Table 4.1: Summary of Hyperparameters

5 Results

This chapter is dedicated to the exhibit of the experimental results obtained by testing the DDQN algorithm over different scenarios (see section 4.4).

The baseline introduced for the comparison is a fixed-time traffic light. We confirm that the phases are the same introduced while training the model (see section 4.2.1), with the order unchanged and constant over the episodes. For what concern the time interval of each phase, we computed it through the standard Webster's formula [22]; the latter outputs the *optimal cycle length* of a network, given as input the observed traffic volume and the saturation flow (i.e.: the highest possible amount of vehicular flow for each road. It depends on the width of the latter and the dynamics allowed to car movements). The relative timings have been approximated to the subsequent multiple of 5; eventually, the cycles obtained behave as follows:

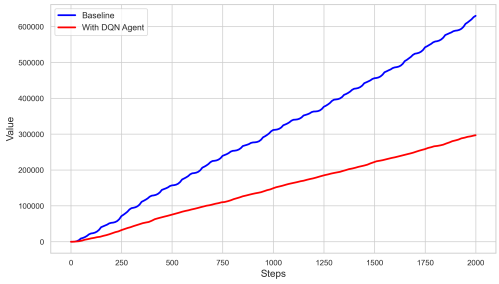
Phase	Veh. N	Veh. S	Veh. E	Veh. W	Ped.	**Low (s)**	**Mid (s)**	**High (s)**
I	green	green	red	red	red	35	40	45
II	yellow	yellow	red	red	red	5	5	5
III	red	red	green	green	red	20	25	30
IV	red	red	yellow	yellow	red	5	5	5
V	red	red	red	red	green	15	15	15
VI	red	red	red	red	yellow	5	5	5

Table 5.1: Baseline for TL phases - Variable traffic

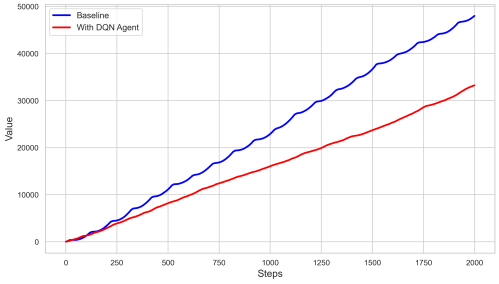
5.1 Waiting Time

5.1.1 Varying Traffic Conditions

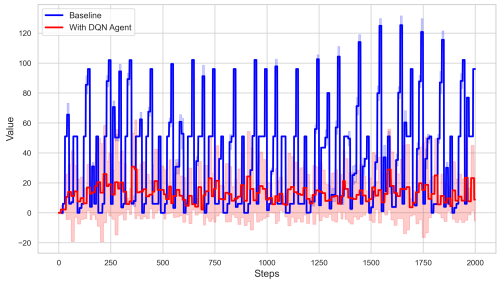
Light Traffic



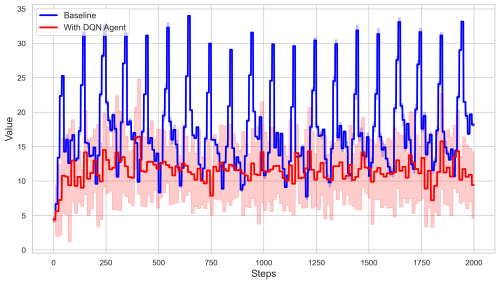
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative



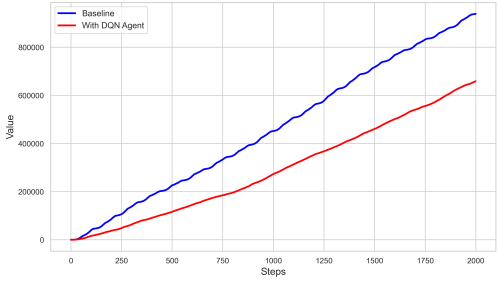
(c) Mean Aggregate Queue Length (m) - Instantaneous



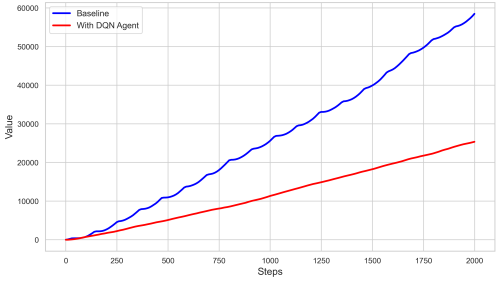
(d) Mean Aggregate Delay - Instantaneous

Figure 5.1: WT - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Light Traffic

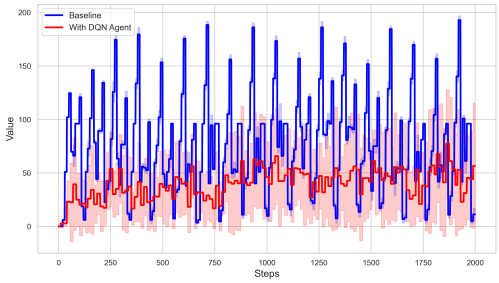
Moderate Traffic



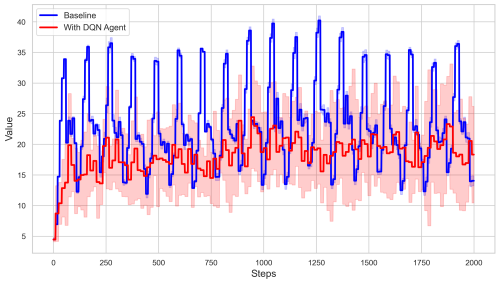
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative



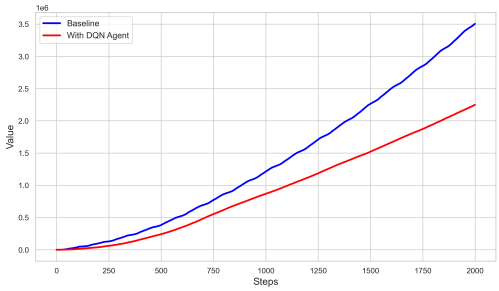
(a) Mean Aggregate Queue Length (m) - Instantaneous



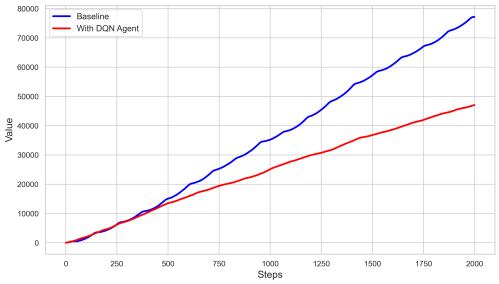
(b) Mean Aggregate Delay - Instantaneous

Figure 5.2: WT - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Moderate Traffic

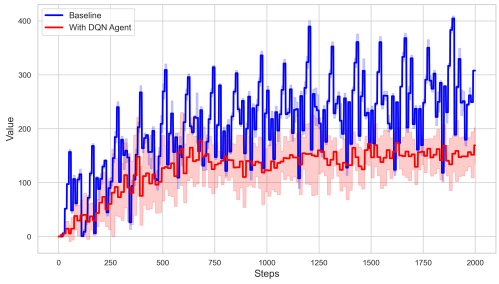
Heavy Traffic



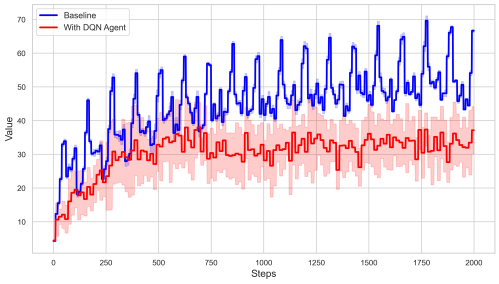
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative

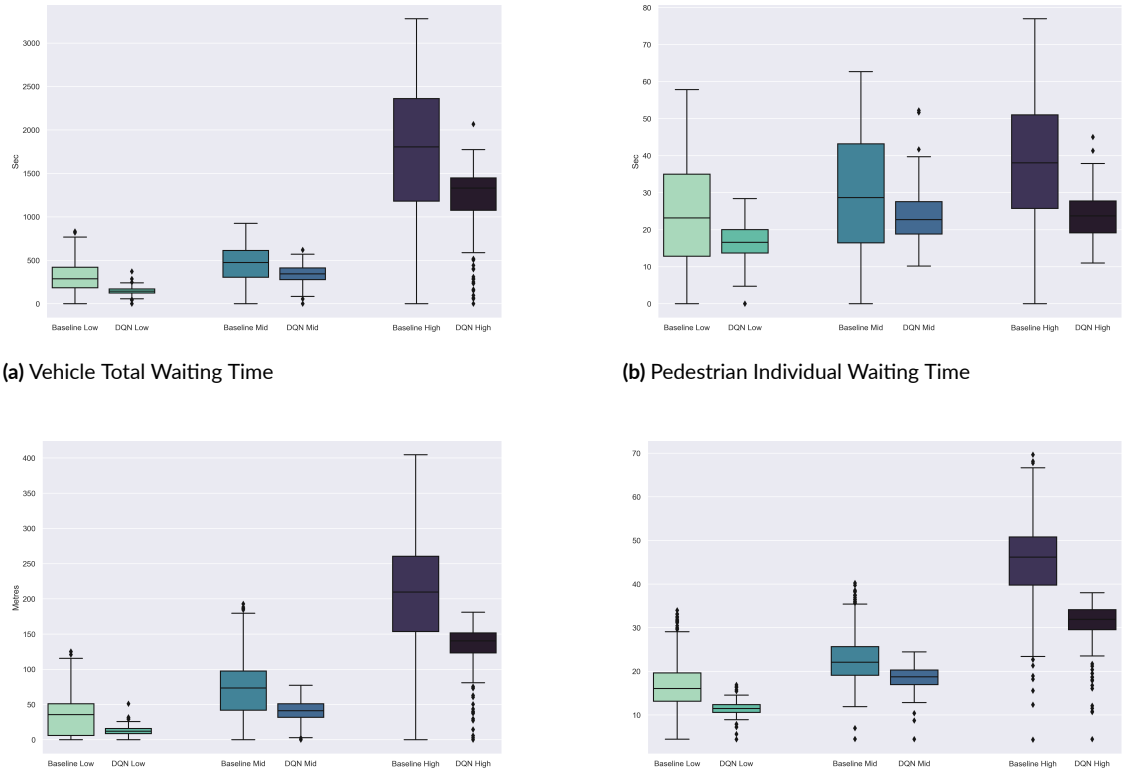


(c) Mean Aggregate Queue Length (m) - Instantaneous



(d) Mean Aggregate Delay - Instantaneous

Figure 5.3: WT - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Heavy Traffic



(a) Vehicle Total Waiting Time

(b) Pedestrian Individual Waiting Time

(c) Mean Aggregate Queue Length

(d) Mean Aggregate Delay

Figure 5.4: WT - Comparison across Varying Traffic Levels against Fixed-Time TL

In this section we presented the results obtained by our DDQN Agent under varying traffic conditions, and compared them to the baseline.

In the first part, we demonstrate how the model performed in each scenario separately, providing the evaluations recorded across the four primary metrics used. These metrics were analyzed from various perspectives, the relative description follows (the reference is made on the initial paragraph, but the format is consistently applied in the parallel sections).

1. 5.1(a) : Cumulative sum of vehicle total waiting time;
2. 5.1(b) : Cumulative sum of pedestrian average waiting time;
3. 5.1(c) : Instantaneous value of vehicle total queue length;
4. 5.1(d) : Instantaneous value of vehicle total delay.

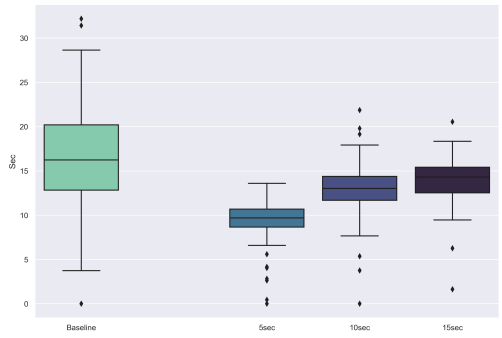
These metrics were recorded at regular intervals across 10 episodes, each lasting 2000 time steps, and the average values were subsequently computed.

In the second part, a comparison with the baseline is shown across the three models together, in order to provide a general overview of the results achieved by the model.

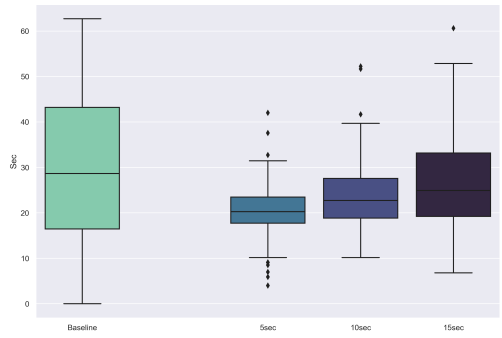
We conclude by confirming that the algorithm surpassed the baseline in all the three configurations. Waiting time was reduced by 52.8% in the light traffic scenario, by 27.2% and 32.7% in the moderate/heavy ones; by analyzing the magnitude of these relative improvements, we reveal cumulative gains of respectively $3.2e5$, $3.6e5$, $1.2e6$ seconds within the 2000-second time window for each scenario. Aggregate queue length was reduced respectively by 65.1%, 45.7% and 38.4%; total delay by 35.2%, 24.2% and 32.6%. Additionally, pedestrian waiting times were also reduced, with improvements ranging from 20% to 40% in all the configurations.

We can therefore conclude by validating the model's ability to handle highly diverse traffic volumes while simultaneously managing pedestrian interactions. In light traffic conditions the percentage improvements are substantial, the highest across the different configurations. In heavy scenario, instead, the baseline was not able to handle the vehicles load, as evidenced by the decreasing performance observed in 5.3 (c) and (d); the model, on the other hand, was able to prevent the congestion, thereby stabilizing the vehicles' conditions.

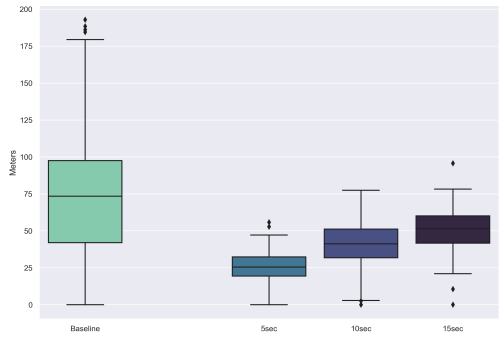
5.1.2 Varying Decision-Making Frequency



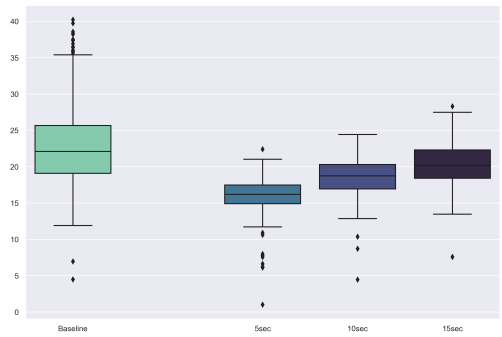
(a) Vehicle Individual Waiting Time



(b) Pedestrian Individual Waiting Time



(c) Mean Aggregate Queue Length



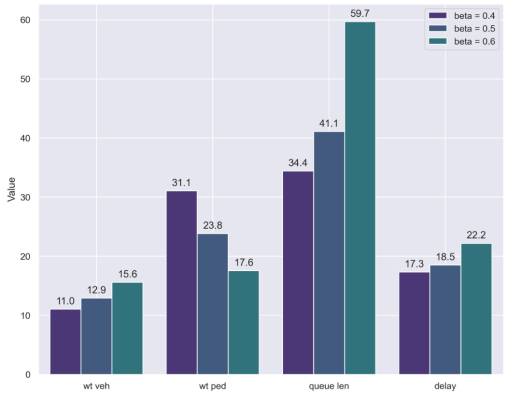
(d) Mean Aggregate Delay

Figure 5.5: WT - Comparison against Fixed-Time TL across Varying Frequency Levels

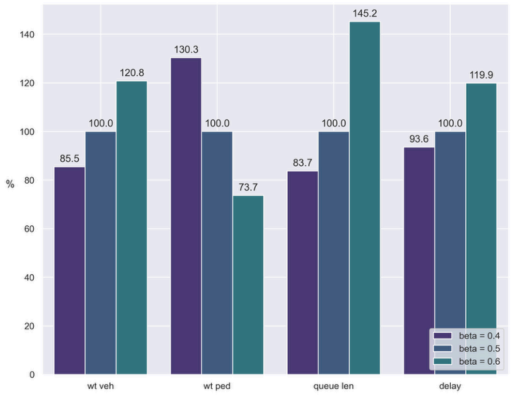
In this subsection, we compare the performance of the baseline with the three models trained under different decision-making frequencies.

As we can note, the performance of the three models is generally comparable, with the 5-second model showing a slight advantage. It is important however to underscore the practical implications of this high-frequency approach: if on one hand the agent is able to create more tailored divisions of the TL phases - resulting in better performance - on the other it has to perform three times more the computations required by the slow-intervention model, demanding much greater computational resources. We let ”supervisors” decide which version to implement, according to their specific needs and capabilities.

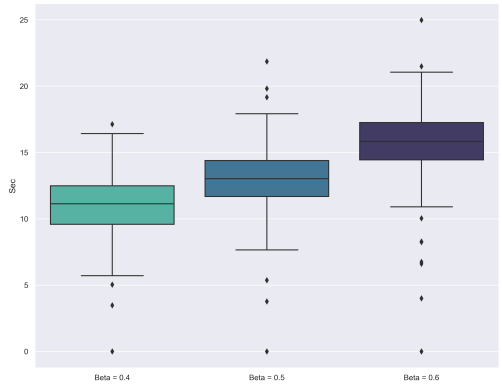
5.1.3 Fairness - Varying β



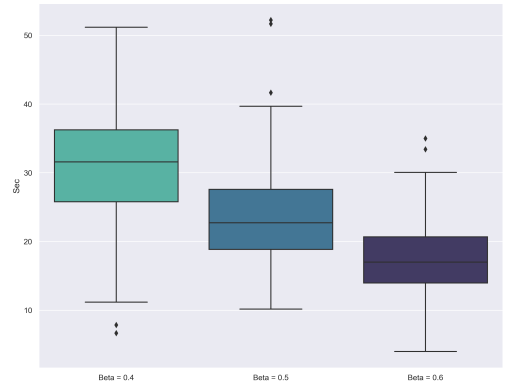
(a) General Comparison



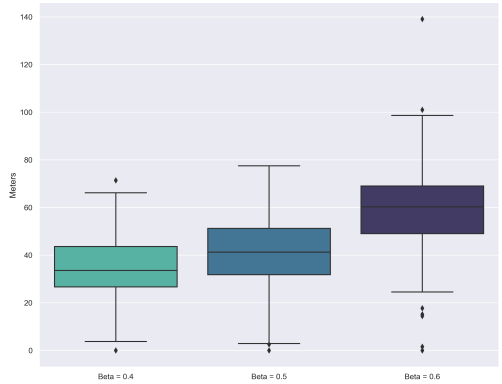
(b) General Comparison w.r.t. Neutral Scenario



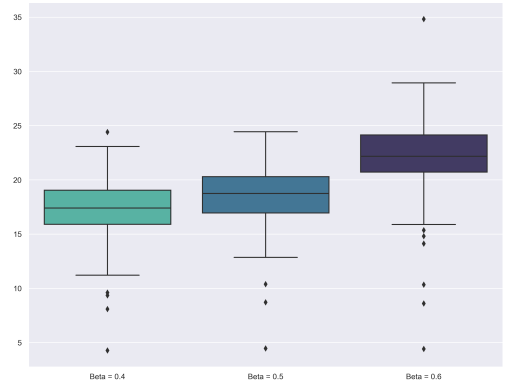
(c) Vehicle Individual Waiting Time



(d) Pedestrian Individual Waiting Time



(e) Mean Aggregate Queue Length



(f) Mean Aggregate Delay

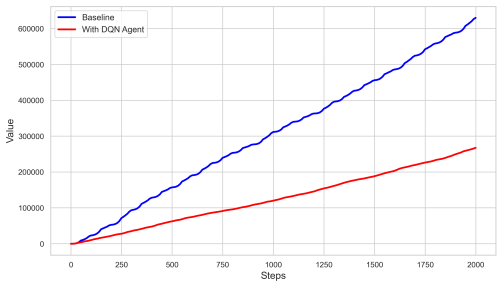
Figure 5.6: WT - Comparison against Fixed-Time TL with Varying Levels of Fairness

Here we compare the performance of the "neutral" model (i.e. where $\beta = 0.5$) with that of models featuring slight variations in the weight factor. By increasing β we favor pedestrians over vehicles: the former experienced a 26.3% reduction in their waiting time, while the latter saw their performance decreasing (i.e. higher waiting times, longer queues and larger delays). This is in line with the trade-off between the two categories we aimed at. Logically, by reducing the level of β we improve vehicles flow: when $\beta = 0.4$, vehicle waiting time decreased by 14.5%, queue length by 16.3% and total delay by 16.4%.

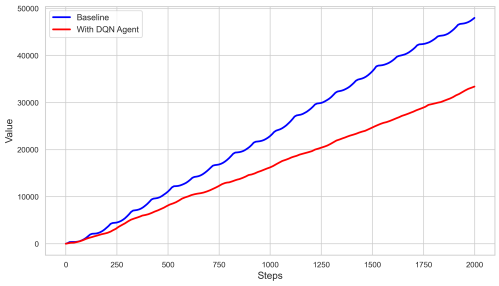
5.2 Queue Length

5.2.1 Varying Traffic Conditions

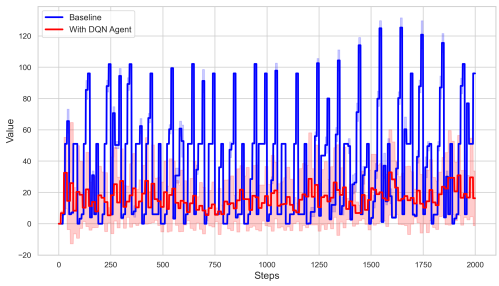
Light Traffic



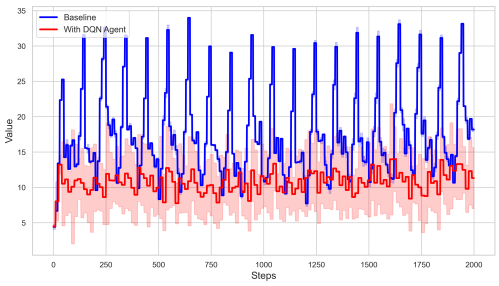
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative



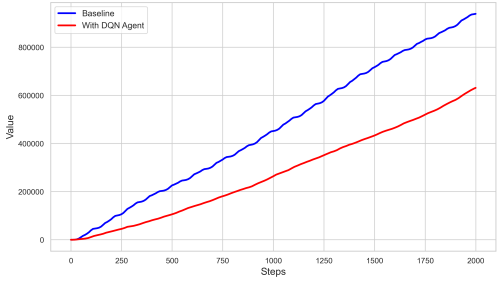
(c) Mean Aggregate Queue Length (m) - Instantaneous



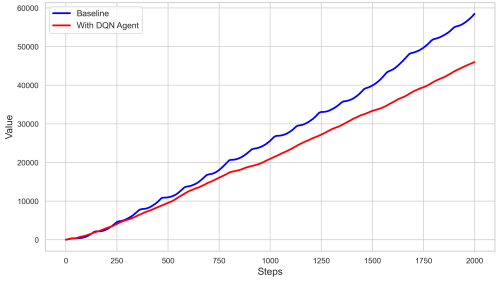
(d) Mean Aggregate Delay - Instantaneous

Figure 5.7: QL - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Light Traffic

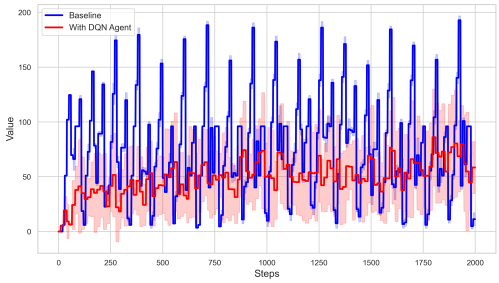
Moderate Traffic



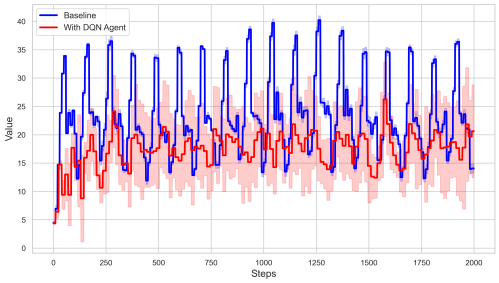
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative



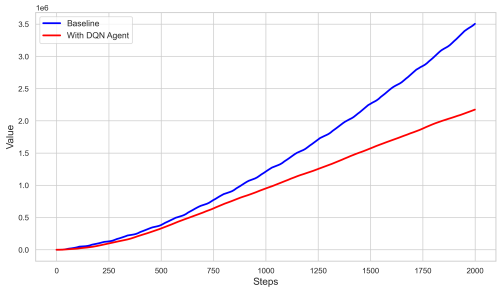
(a) Mean Aggregate Queue Length (m) - Instantaneous



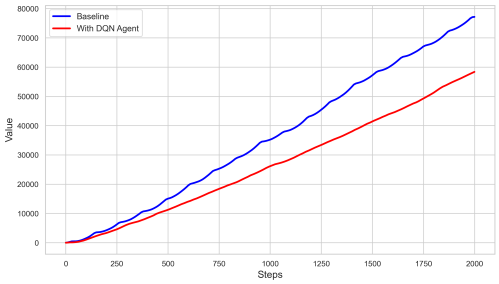
(b) Mean Aggregate Delay - Instantaneous

Figure 5.8: QL - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Moderate Traffic

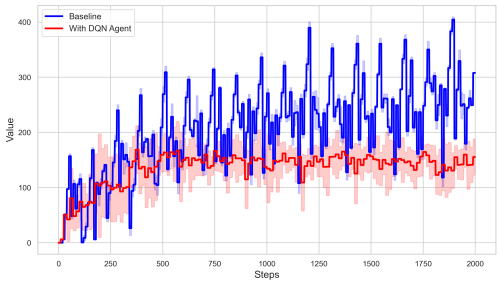
Heavy Traffic



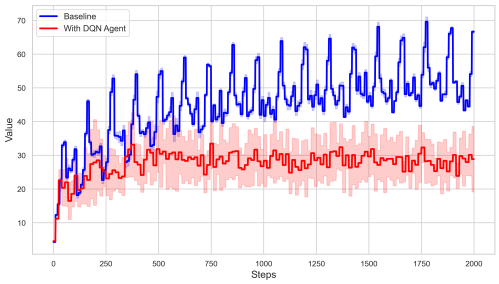
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative

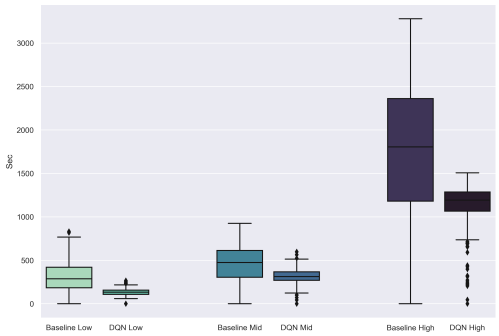


(c) Mean Aggregate Queue Length (m) - Instantaneous

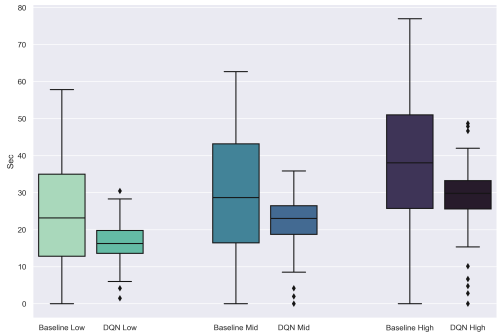


(d) Mean Aggregate Delay - Instantaneous

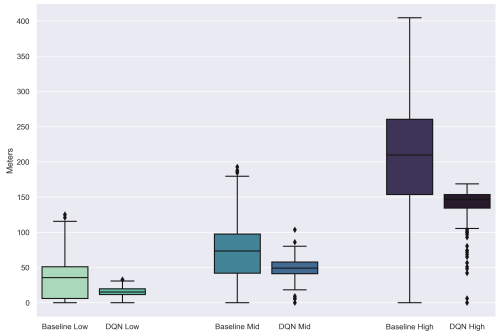
Figure 5.9: QL - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Heavy Traffic



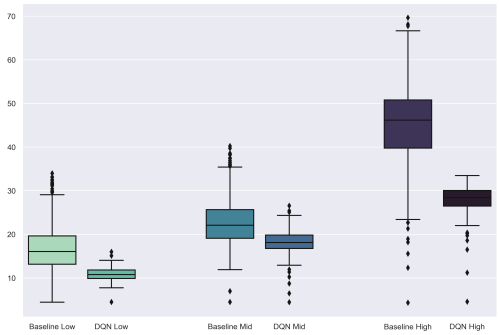
(a) Vehicle Individual Waiting Time



(b) Percentage improvement in vehicle w.t.



(c) Mean Aggregate Queue Length



(d) Mean Aggregate Delay

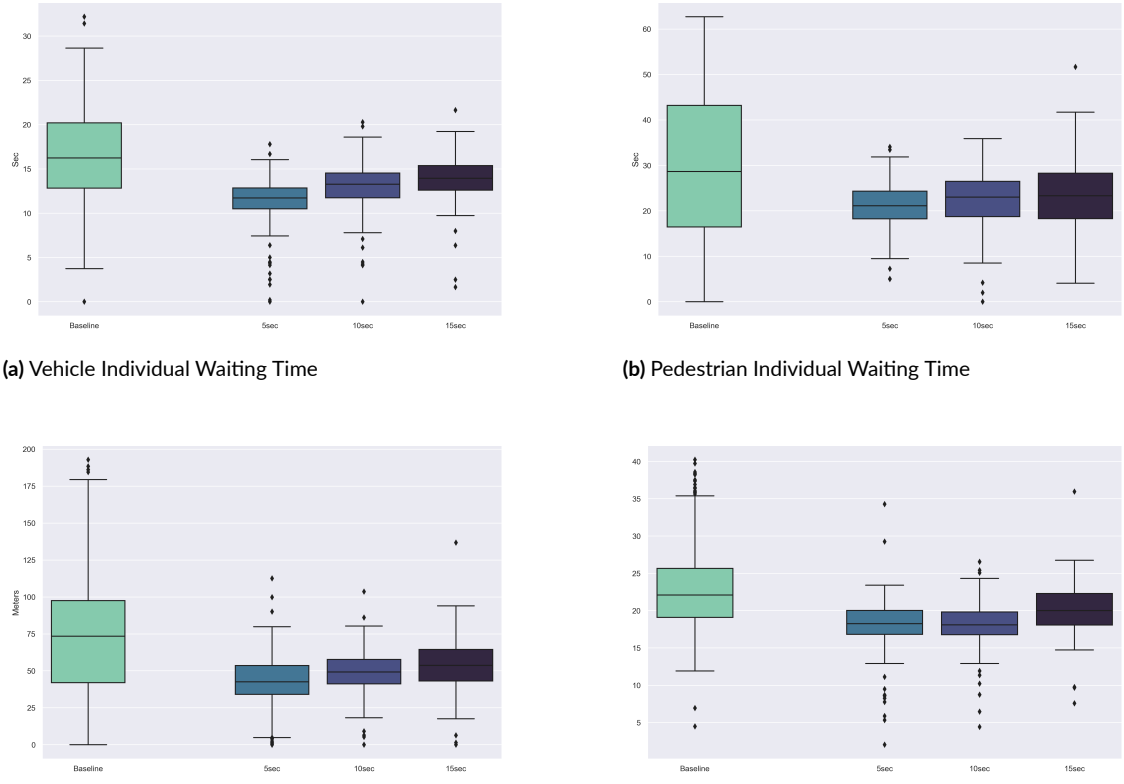
Figure 5.10: QL - Comparison across Varying Traffic Levels against Fixed-Time TL

The conclusions are similar to the one provided for the previous reward function, given the comparability between the results.

Vehicle waiting time was reduced respectively by 57.3%, 32.4% and 37.2% in the three different scenarios. Aggregate queue length was reduced by 56%, 34% and 34%; total delay by 38%, 28% and 36%. Pedestrian waiting time by $25 - 30\%$.

Even in this case, the model outperformed the fixed-time TL achieving competitive results in every scenario.

5.2.2 Varying Decision-Making Frequency



(a) Vehicle Individual Waiting Time

(b) Pedestrian Individual Waiting Time

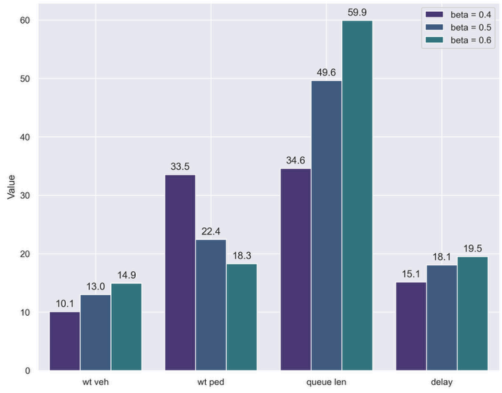
(c) Mean Aggregate Queue Length

(d) Mean Aggregate Delay

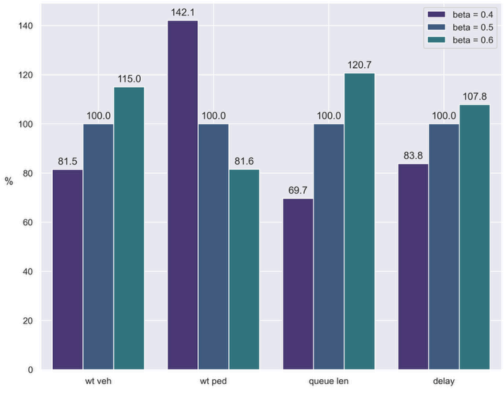
Figure 5.11: QL - Comparison against Fixed-Time TL under Varying Frequency Levels

In this context as well, we confirm what stated before: now more than before, the variation within decision-making frequencies does not create huge differences. Rapid intervention models, indeed, being more granular in their approach, grant a more effective subdivision of TL phases, achieving performances higher than $10 - 20\%$ with respect to the 5-sec intervention models. However, this improvement must be compared to the increased computational costs incurred by these models.

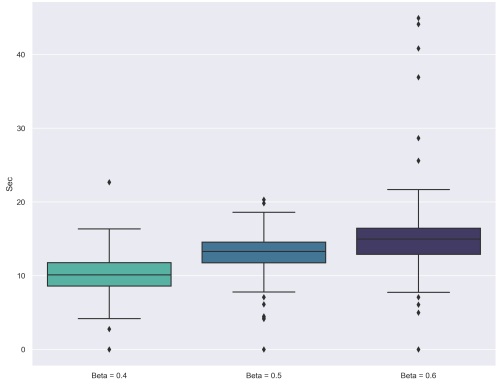
5.2.3 Fairness - Varying β



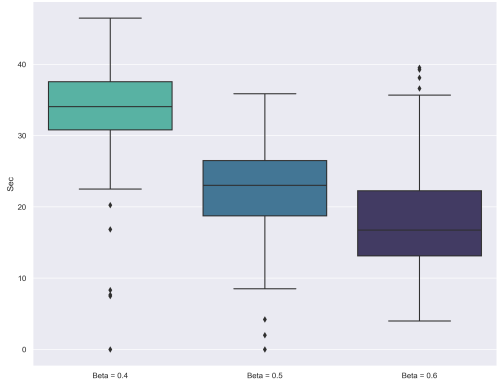
(a) Overall Comparison



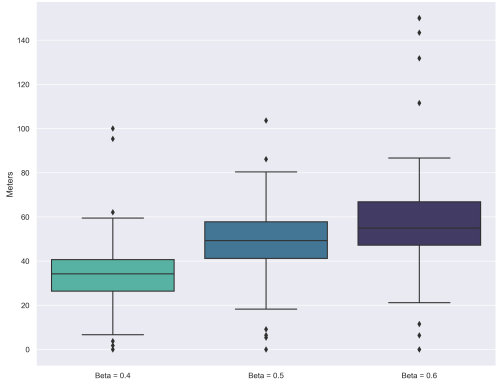
(b) Overall Comparison w.r.t. Neutral Scenario



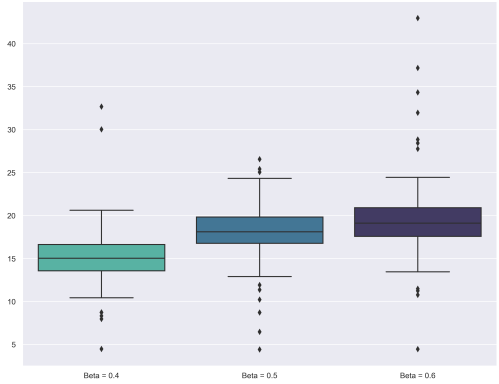
(c) Vehicle Individual Waiting Time



(d) Pedestrian Individual Waiting Time



(e) Mean Aggregate Queue Length



(f) Mean Aggregate Delay

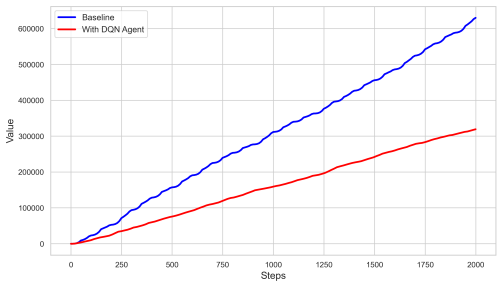
Figure 5.12: QL - Comparison against Fixed-Time TL with Varying Levels of Fairness

In this scenario, the results are more pronounced than in previous cases. With $\beta = 0.4$ we improve the vehicles conditions by nearly 20% in all the metrics: compared to the neutral scenario, waiting time decreases to 81.5%, queue length to 69.7% and total delay to 83.8%. These improvements come at the expense of reducing the priority for pedestrians, whose waiting time increases by 42.1%. Conversely, raising β to 0.6 leads to minor losses in vehicles efficiency - waiting time increases by 15%, queue length by 20.7% and total delay by 7.8% - while pedestrian waiting time decreases by 18.4%.

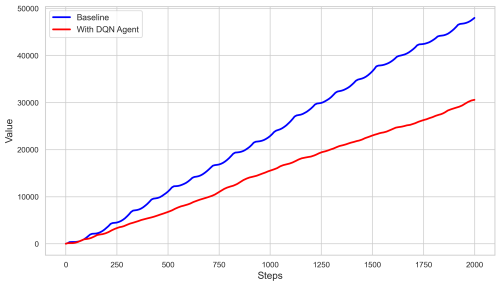
5.3 Total Delay

5.3.1 Varying Traffic Conditions

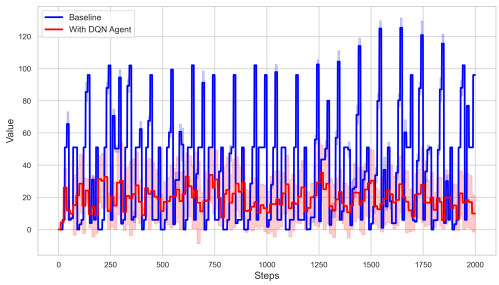
Light Traffic



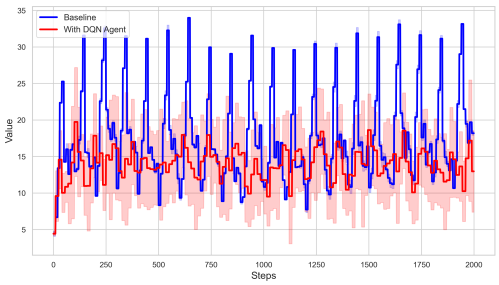
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative



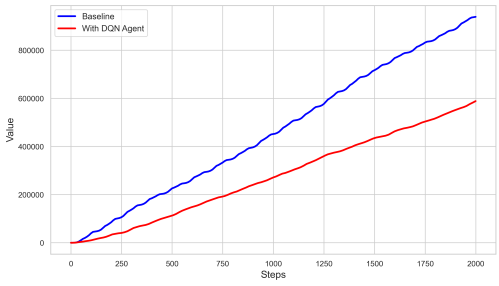
(c) Mean Aggregate Queue Length (m) - Instantaneous



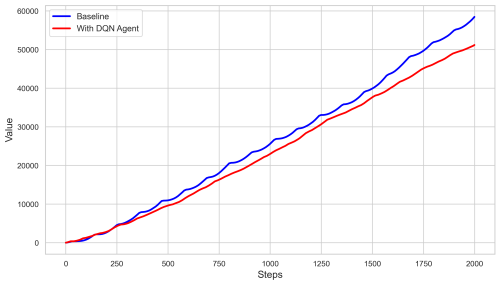
(d) Mean Aggregate Delay - Instantaneous

Figure 5.13: TD - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Light Traffic

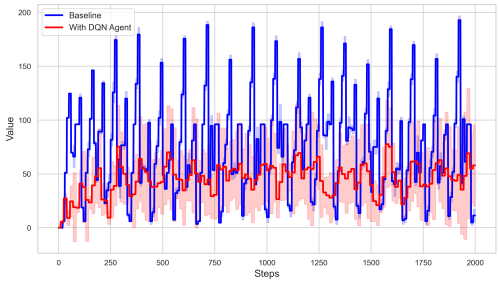
Moderate Traffic



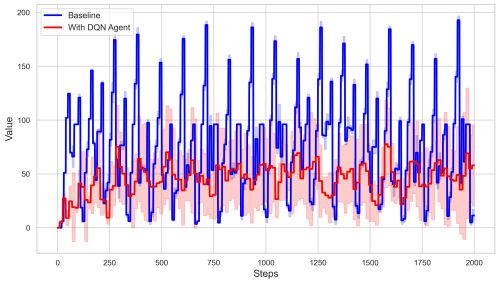
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative



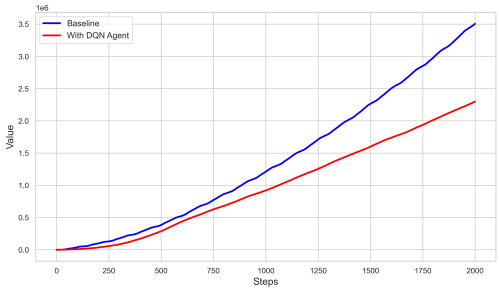
(a) Mean Aggregate Queue Length (m) - Instantaneous



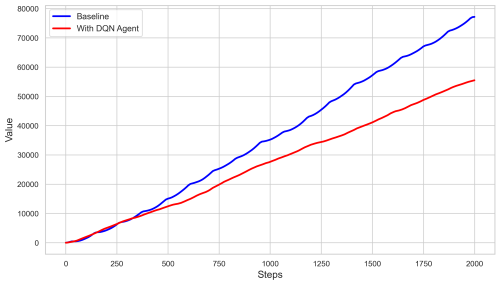
(b) Mean Aggregate Delay - Instantaneous

Figure 5.14: TD - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Moderate Traffic

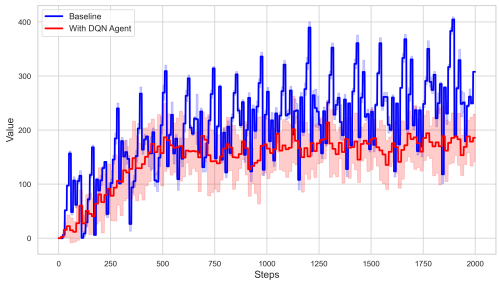
Heavy Traffic



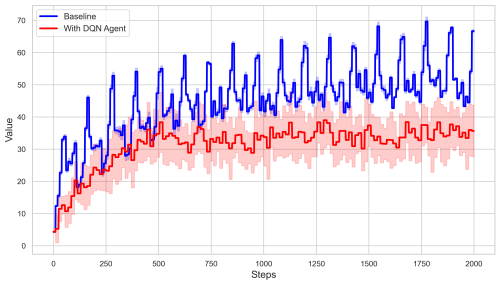
(a) Vehicle Total Waiting Time (s) - Cumulative



(b) Pedestrian Average Waiting Time (s) - Cumulative

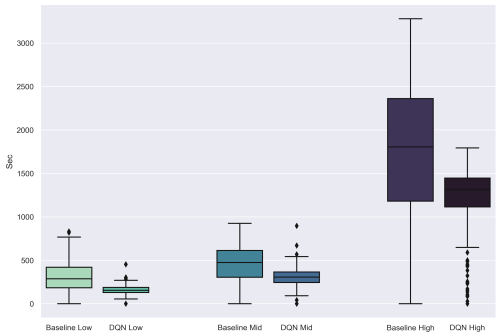


(c) Mean Aggregate Queue Length (m) - Instantaneous

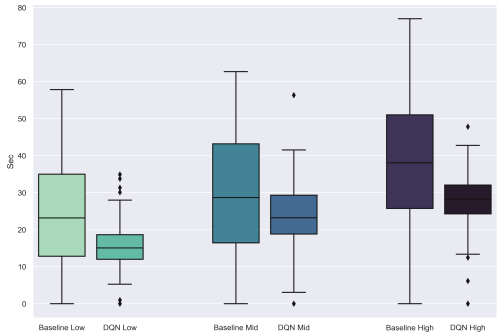


(d) Mean Aggregate Delay - Instantaneous

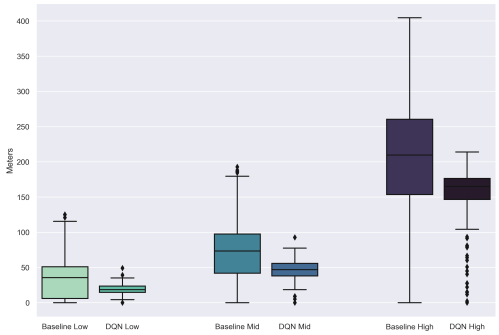
Figure 5.15: TD - Metrics Comparison: Temporal Analysis of Vehicle and Pedestrian Flow in Heavy Traffic



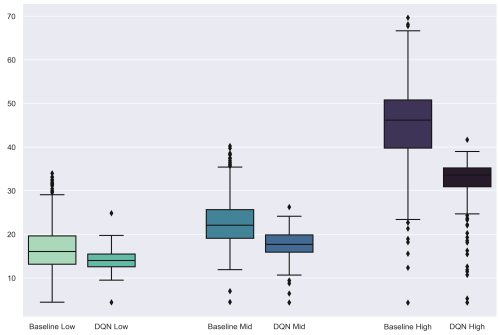
(a) Vehicle Waiting Time



(b) Percentage improvement in vehicle W.T.



(c) Mean Aggregate Queue Length



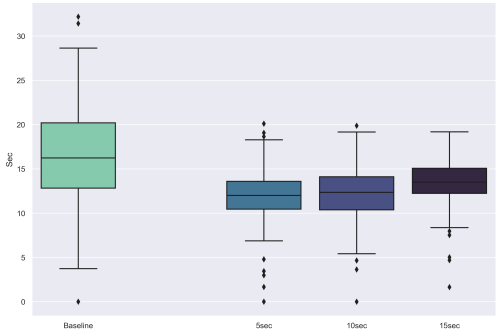
(d) Mean Aggregate Delay

Figure 5.16: TD - Comparison under Varying Traffic Levels against Fixed-Time TL

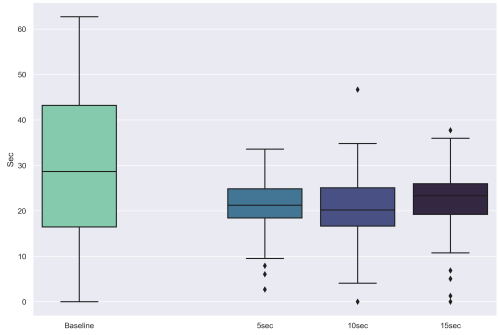
The results are consistent with the previous findings.

Vehicle waiting time was reduced respectively by 49.7%, 35.4% and 32.8% in the three different scenarios. Aggregate queue length was reduced by 48.1%, 38.5% and 27.5%; total delay by 20.6%, 25.3% and 30.2%. Pedestrian waiting time by 34.0%, 21.5% and 28.4%.

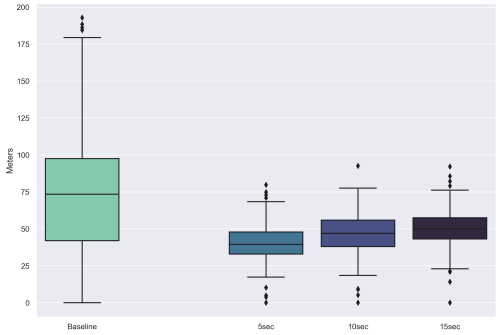
5.3.2 Varying Decision-Making Frequency



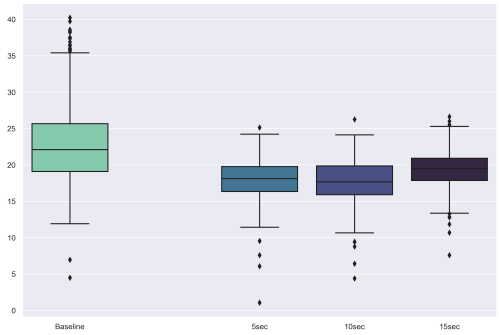
(a) Vehicle Individual Waiting Time



(b) Pedestrian Individual Waiting Time



(c) Mean Aggregate Queue Length

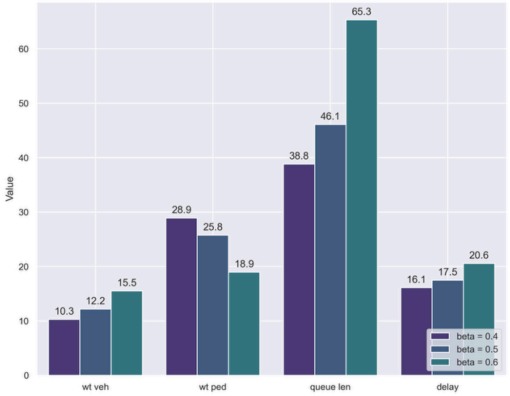


(d) Mean Aggregate Delay

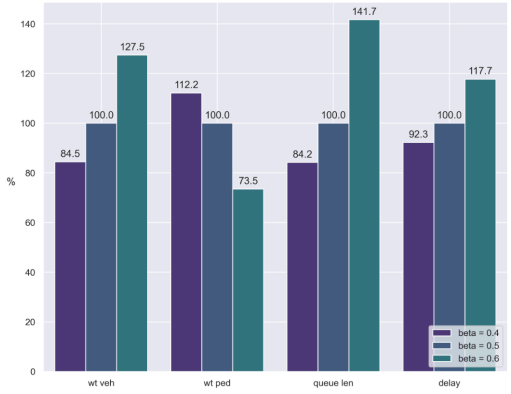
Figure 5.17: TD - Comparison against Fixed-Time TL under Varying Frequency Levels

We reaffirm the conclusions provided earlier. All three models achieved reductions with respect to the baseline, ranging from 10% to 50%; by narrowing the time interval, these improvements get slightly higher across all metrics, particularly in vehicle waiting time and queue length. If no limits are imposed on computational resources, then the fast intervention model should always be preferred; vice versa, at less congested intersections, it is advisable to carefully assess this trade-off to determine the optimal solution.

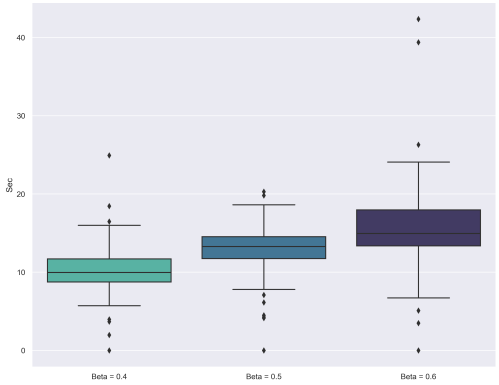
5.3.3 Fairness - Varying β



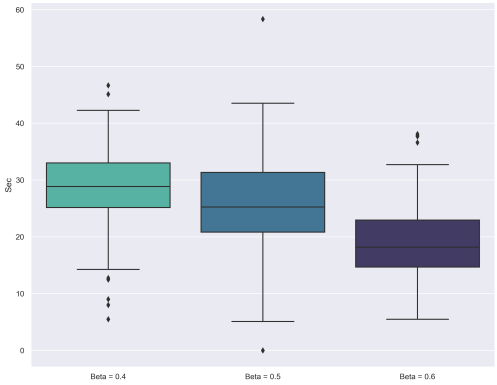
(a) General Comparison



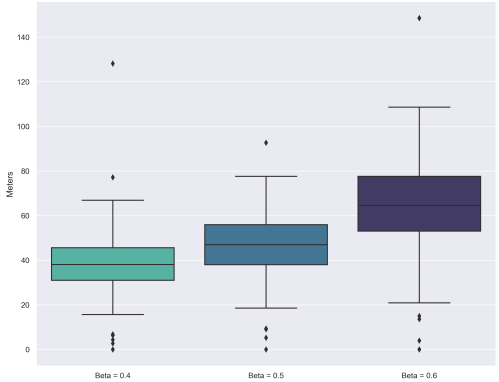
(b) General Comparison w.r.t. Neutral Scenario



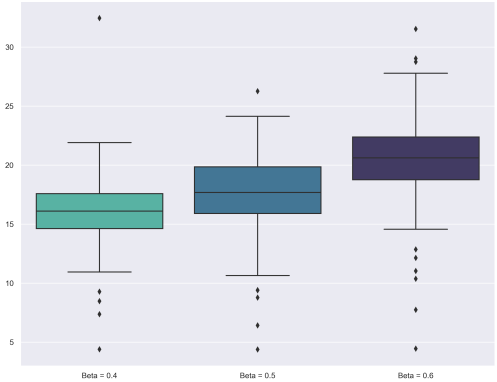
(c) Vehicle Individual Waiting Time



(d) Pedestrian Individual Waiting Time



(e) Mean Aggregate Queue Length



(f) Mean Aggregate Delay

Figure 5.18: TD - Comparison against Fixed-Time TL with Varying Levels of Fairness

Even in this case we can confirm what stated earlier. An increase in the weight factor β attributes more attention to pedestrians, while a reduction gives priority to vehicles. The results are almost identical to the one obtained in the first case, and can be seen in the figure above.

5.4 Comments

The results obtained are very promising, with all the three different branches displaying models that outperformed the baseline under every aspect. In the following, we will refer to the waiting time based model as the 1st model, the queue length based model as the 2nd model, and the total delay based model as the 3rd model. We are going to repeat the general results, so as to provide also a general overview of the models' performances.

For what concern traffic variations, the first two models achieved higher performances across all metrics. For light traffic, optimizing queue length or waiting time provided the highest benefit, as the system is less constrained by congestion: vehicle's waiting time decreased by more than 50% in all the three models; pedestrian's waiting time decreased by $30 - 34\%$; aggregate queue length decreased in the three models respectively by 65%, 56% and 48%; total delay decreased by $35 - 38\%$ in the first two models, and surprisingly only by 20% in the delay-based model. This suggest that such approach may not fully address congestion, and further measure should be considered.

The results for moderate traffic are less pronounced, but more stable across the different models: vehicle's waiting time decreased by an average of 30%, pedestrian's waiting time by 20%, aggregate queue length by nearly 40% and total delay by 25%. These results are consistent across the three models, with only a $2/3$ percentage point variation from the mean. The improvement with respect to the baseline is less significant than the one obtained in the other two traffic scenarios; this might be attributed to the capacity of pre-set TL schedules to handle standard urban traffic flows. It is important to underline, in fact, that the tests have been performed under constant flows, an environment in which fixed-time TL - if properly adjusted - can achieve high cost-effectiveness ratios.

The third scenario involved an important volume of vehicles, which lead to congestion in both tests. As shown in 5.3 - 5.9 - 5.15, the DDQN agent has been able to limit and handle correctly the situation: it reduced vehicle's waiting time by 35% in all the models and simultaneously improved also the pedestrian's flow by 40% in the 1st model and $25 - 28\%$ in the other two. Queue length was reduced respectively by approximately 38%, 34% and 27%;

Total delay was reduced by 32%, 38% and 30%.

Shifting our focus to the models based on frequency variation, we observed the expected results: reducing the time window led to slight improvements across all performance metrics, including reductions in waiting time, queue length, and total delay. However, these gains came at the cost of increased computational complexity and additional operational overhead. As previously mentioned, the choice of which model to use is left to the discretion of the appropriate authority.

Lastly, an overview on the balance between vehicles and pedestrians is provided. We notice that an increase by 0.1 in β lead the models to prioritize pedestrians, with an average reduction in waiting time of 23% with respect to the standard model ($\beta = 0.5$). On the other hand, a reduction by 0.1, by prioritizing vehicles, reduces their waiting time by 17%, the queue length by 15% and the total delay by 7%; in the 2nd model, these results are significantly higher, reaching respectively reduction of 19%, 30% and 16%.

These results confirm the effectiveness and adaptability of using different models tailored to various scenarios: situations that demand greater consideration and attention for vulnerable road users can benefit from models with higher beta values; conversely, scenarios where road traffic reaches critical and challenging levels can leverage models with lower beta values. Such models have proven to significantly reduce vehicle waiting times, cut down queues, and minimize delays, thereby ensuring smoother traffic flow and substantial savings.

6

Conclusion

In this thesis we explored the potential of RL techniques to optimize traffic management in urban environments.

We started by introducing the problem and the relative implications on the current scenario. Right after, we analyzed the several approaches presented in literature, which gave us a wider perspective on the possible solutions to tackle the phenomenon through *Reinforcement Learning* techniques. Consequently, we briefly explained the single-agent Reinforcement Learning framework, with a focus on the application of neural networks; the latter, being universal approximators, were essential for handling high-dimensional environments like the one considered here.

In Chapter 3 and 4 we delved into the core of the thesis: we proposed our approach, first from a theoretical point of view, and then from the relative practical implementation. We discussed the network generation process in SUMO and the corresponding virtual representation with FLOW.

Additionally, we defined the agent's state and action spaces, and established three different reward functions. Lastly, we better defined the algorithm - the *Double Deep Q-Network*, along with its features and the hyperparameters applied.

Finally, Chapter 5 was dedicated to the analysis of the results. Here we noticed the advantage proven by RL techniques over standard traffic control baselines used nowadays.

6.1 Limitations and Future Improvements

The advantage showed by our models marks only the beginning of what is possible. This work was not intended as an exhaustive or fixed solution to a specific scenario; instead, the aim of this thesis was the one of introducing and analyzing a very promising technique to both improve the environmental conditions harmed by heavy traffic scenarios and - simultaneously - shed a light on *Vulnerable Road Users*, in particular pedestrians. The latter are rarely considered in literature, while they would need much more attention: any kind of city, from small towns to metropolis, do have a significant flow of VRUs who want to take care of.

This work demonstrated that a balanced approach benefiting both vehicular traffic and VRUs is possible, and the advantages would be priceless; indeed, future works should put their attention on refining this equilibrium and tailoring the reward functions to more effectively serve these two categories. This necessity stems from a double perspective: first, from the desire of reaching a greater balance and characterization between the two reward components; and secondly, from the ethical implications that may arise from the choice of which model to deploy in specific contexts and why.

Another interesting proposal would be the implementation - and testing - of dynamic traffic flows: this would unlock the true potential of the RL-based Agent, which excel in learning and specially adapting its behavior in real-time, unthinkable for any kind of rule-based TL. Unfortunately, these capabilities have not been fully demonstrated in this study, given the challenges associated with the implementation of highly diverse traffic conditions using the current version of Flow. Addressing these limitations in future work will further enhance the robustness and applicability of RL-based traffic solutions.

References

[1] E. C. of Auditors, "Urban mobility in the eu," European Court of Auditors, Tech. Rep., 2019. [Online]. Available: https://www.eca.europa.eu/lists/ecadocuments/ap19_07/ap_urban_mobility_en.pdf

[2] I. ♦. I. S. per la Protezione e la Ricerca Ambientale, "Italian greenhouse gas inventory 1990-2021: National inventory report 2023," ISPRA, Tech. Rep., 2023. [Online]. Available: https://www.isprambiente.gov.it/files2023/pubblicazioni/rapporti/rapporto-385_2023_iir2023.pdf

[3] R. P. Dameri *et al.*, "Searching for smart city definition: a comprehensive proposal," *International Journal of computers & technology*, vol. 11, no. 5, pp. 2544–2551, 2013.

[4] R. E. Allsop, "Delay at a fixed time traffic signal—i: Theoretical analysis," *Transportation Science*, vol. 6, no. 3, pp. 260–285, 1972.

[5] R. S. Sutton, "Reinforcement learning: An introduction," *A Bradford Book*, 2018.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[8] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[9] L. Alegre, "Sumo-rl," *GitHub, github. com/LucasAlegre/sumo-rl*, 2019.

[10] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247–254, 2016.

[11] H. Wei, G. Zheng, H. Yao, and Z. 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*, 2018, pp. 2496–2505.

[12] S. S. Mousavi, M. Schukat, and E. Howley, "Traffic light control using deep policy-gradient and value-function-based reinforcement learning," *IET Intelligent Transport Systems*, vol. 11, no. 7, pp. 417–423, 2017.

[13] X. Liang, X. Du, G. Wang, and Z. Han, "A deep reinforcement learning network for traffic light cycle control," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1243–1253, 2019.

[14] M. Yazdani, M. Sarvi, S. A. Bagloee, N. Nassir, J. Price, and H. Parineh, "Intelligent vehicle pedestrian light (ivpl): A deep reinforcement learning approach for traffic signal control," *Transportation research part C: emerging technologies*, vol. 149, p. 103991, 2023.

[15] M. Aslani, M. S. Mesgari, and M. Wiering, "Adaptive traffic signal control with actor-critic methods in a real-world traffic network with different traffic disruption events," *Transportation Research Part C: Emerging Technologies*, vol. 85, pp. 732–752, 2017.

[16] R. Zhu, S. Wu, L. Li, P. Lv, and M. Xu, "Context-aware multiagent broad reinforcement learning for mixed pedestrian-vehicle adaptive traffic light control," *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 19 694–19 705, 2022.

[17] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[18] R. Bellman, "Dynamic programming," *science*, vol. 153, no. 3731, pp. 34–37, 1966.

[19] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE, 2018, pp. 2575–2582.

[20] C. Wu, A. R. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: A modular learning framework for mixed autonomy traffic," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1270–1286, 2021.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[22] APSEd. (2022) Apsed. [Online]. Available: https://www.apsed.in/post/traffic-signal-design-webster-s-formula-for-optimum-cycle-length

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Gian Antonio Susto, and to my "co-supervisor" Matteo Cederle, for giving me the opportunity to work on this project and for guiding me through its development with expertise and dedication.

To my parents, for their curiosity and care towards a subject they knew little about, but always made an effort to understand and support, something I deeply appreciate.

To my friends, for always listening and offering reassurance in every moment, providing their encouragement when I needed it most.

To my family, for always believing in me and backing me unconditionally throughout this journey.

I dedicate this achievement to all of you who have supported me, not just in these months, but throughout all the years leading up to this moment.