TÉCNICO LISBOA

Traffic Light Control using Deep Reinforcement Learning

Pedro Pinto Santos

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Francisco António Chaves Saraiva de Melo
Prof. José Alberto Rodrigues Pereira Sardinha

Examination Committee

Chairperson: Prof. Paolo Romano
Supervisor: Prof. Francisco António Chaves Saraiva de Melo
Member of the Committee: Prof. Rui Miguel Carrasqueiro Henriques

January 2021

Acknowledgments

Throughout my academic journey, I received a great deal of support and assistance. To everyone that accompanied me through these moments, I express my acknowledgments.

I would like to express my profound gratitude to my supervisors, Prof. Francisco Melo and Prof. Alberto Sardinha, for all the feedback, wisdom, and guidance. I thank them for believing in my capabilities and providing me the opportunity to receive a research grant while working on this dissertation. This work was partially supported by national funds through the Portuguese Fundação para a Ciência e a Tecnologia under project ILU, with reference DSAIPA/DS/0111/2018.

I deeply thank Guilherme Varela, without whom this work would never be possible. The contents that led to the writing of this thesis came to life from a fruitful and close collaboration with Guilherme. Thank you for all the work, dedication, availability and company.

I also acknowledge the lab managers at GAIPS for providing all the needed hardware-related support.

I would like to thank my family for their love, support, and making me who I am today. For comprehending all the time I spent away while working on this dissertation. I especially thank my parents for their unconditional caring, and my sister for the friendship and company, as well as insightful discussions related to this dissertation. I also thank my parents for providing me all the funding that allowed me to pursue this degree, as well as study abroad during one year.

Last, but not least, a special thanks to my friends and colleagues for making my academic journey much more enjoyable. I especially thank Sara, Margarida, Francisco Catarrinho, Francisco Neves, Mariana, João and Pilar.

I thank each and every one of you.

Abstract

This work addresses the development of intelligent Traffic Signal Controllers (TSCs) using Reinforcement Learning (RL) techniques, assessing how the different dimensions of the control problem affect the efficiency of the resulting controllers under a variety of simulation scenarios. Firstly, we provide a novel methodology for the development of intelligent TSCs. Such methodology contributes one step further towards a wider application of RL in traffic light control, by ensuring some level of standardization at the different stages of the experimental process: simulation setup, TSC design — problem formulation and selection of the RL method — as well as performance estimation and comparison. Secondly, while following the proposed methodology, we provide a thorough study of different state spaces and reward functions, as well as of three different RL algorithms (Q-learning, Deep Q-network, and Deep Deterministic Policy Gradient). A fully decentralized approach to traffic light control featuring a constrained action space definition is considered. The performance of the developed TSCs is assessed with respect to different traffic demands, under three traffic networks extracted from the city of Lisbon: a single intersection, an arterial network, and a grid network.

Keywords

Traffic light control; Reinforcement learning; Intelligent transportation systems; Urban mobility; Machine learning.

Resumo

Este trabalho aborda o desenvolvimento de controladores semafóricos inteligentes (TSCs) usando técnicas de aprendizagem por reforço (RL), avaliando de que forma as diferentes dimensões do problema de decisão afetam a eficiência dos controladores resultantes. Em primeiro lugar, apresentamos uma nova metodologia para o desenvolvimento de TSCs. Esta metodologia contribui no sentido de uma aplicação mais ampla de RL no problema de controlo semafórico, por garantir um nível de standardização nas várias fases do processo experimental: configuração da simulação, *design* do TSC — formulação do problema e seleção do método de RL — bem como estimação da *performance* e comparação. Seguindo a metodologia proposta, fazemos um estudo de diferentes espaços de estados e funções de recompensa, assim como três algoritmos de RL (*Q-learning*, *Deep Q-network*, e *Deep Deterministic Policy Gradient*). Consideramos uma abordagem totalmente descentralizada do problema e um espaço de ação restrito. O desempenho dos controladores semafóricos desenvolvidos é avaliado usando diferentes cenários extraídos da cidade de Lisboa: uma interseção isolada, uma via arterial e um conjunto de ruas do centro da cidade.

Palavras Chave

Controlo semafórico; Aprendizagem por reforço; Sistemas inteligentes de transporte; Mobilidade urbana; Aprendizagem automática.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
ANOVA	Analysis of Variance
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
GD	Gradient Descent
ITS	Intelligent Transportation System
LSTM	Long Short Term Memory
ML	Machine Learning
MDP	Markov Decision Process
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
TD	Temporal Difference
TLC	Traffic Light Control
TSC	Traffic Signal Controller

1

Introduction

Contents

Transportation systems play a crucial role in modern societies. Ensuring that the movement of individuals and goods operates smoothly and efficiently is a priority. The continuous population increase and growth in social and economic activities in urban areas lead to the rise in demand for transportation. Nowadays, traffic infrastructures are frequently pushed beyond their capacity, resulting in increased congestion, travel delays, and aggravated environmental impacts.

In order to improve urban mobility, one solution is to expand the capacity of current transportation systems with the development and construction of additional infrastructures. This process is usually expensive and protracted, which can worsen the problem in the short term. Other solution is to improve the efficiency of current infrastructures. One way to accomplish this is with the development of Intelligent Transportation Systems (ITSs), which provide flexible approaches to manage and control traffic, improving the efficiency of transports in a number of situations by taking advantage of the growing sensory data gathered by devices such as ground sensors, video cameras and radars [2–4]. Moreover, ITSs allow increasing the capacity of existing infrastructures while avoiding high construction costs and short-term disadvantages [5]. This work addresses the development of intelligent Traffic Signal Controllers (TSCs) based on recent advancements from the domain of Artificial Intelligence (AI), specifically using Reinforcement Learning (RL) techniques.

In recent years, significant progress has been made in solving challenging problems across various domains using RL. The field has experienced dramatic growth in attention after promising results in tasks such as game-play [6, 7] and robotics [8]. Reinforcement learning is a Machine Learning (ML) paradigm where an agent learns what to do, i.e., how to map situations to actions, so as to maximize a numerical reward signal. The agent is not told beforehand which actions are better for a given situation, but instead must discover which ones yield the most reward through trial and error.

Several lines of research proposed reinforcement learning methods for traffic light control [4], as the problem can be easily cast into the RL framework: the agent represents the traffic signal controller, the environment is the state of the traffic, and the actions are the traffic signal phases. This new direction of research came to life supported by the increasing amount of traffic-related data and computational power [2]. The core idea of RL methods, learning from trial and error without requiring an environment's model, is actually what makes them so appealing to the development of TSCs, and different from the currently used transportation engineering methods. Nowadays, traffic signal controllers still heavily rely on oversimplified traffic models and rule-based methods. Reinforcement learning methods, on the other hand, attempt to learn good Traffic Light Control (TLC) strategies through repeated interaction with the environment, without imposing assumptions about the traffic behaviour. The application of RL methods is, therefore, a promising research direction that may achieve improved adaptability, thus delivering state of the art controllers for TLC [9].

While the application of reinforcement learning techniques to TLC may seem straightforward, there are downsides one should be aware of. The following list identifies the main challenges in the development of RL-based traffic signal controllers:

1. The need for simulation: due to the intrinsic exploratory nature of RL algorithms we cannot allow an agent to learn optimal behaviour by direct interaction with the real environment. Agents usually start their learning process by trying out actions completely at random, something that is completely inappropriate in the context of this work. Due to this, RL agents are usually trained using simulators.

2. The quality of the simulations: while the use of a controlled environment may have some advantages, it is important to notice that the final agent performance is limited by the quality of the simulations and how well they mimic real-world behaviour. If the simulator is not correctly configured to the real traffic scenario, the learnt behaviour may perform poorly in the real environment.

3. The quality of the traffic signal controllers: this point resides on the quality of the RL methods themselves, and how their performance is evaluated. Several RL-based controllers should be studied and their performance and robustness thoroughly assessed under different scenarios. Moreover, the controller must be designed in such a way that safety standards are met. Classical transportation methods should also be included in this comparison. Meaningful and significant performance measures must be used.

4. The need for continuous adaptability: it might be important that the agent continues to adjust its behaviour while deployed to the real world due to the highly variable and complex traffic patterns. However, it is for real-world embedded agents that most warnings about potential dangers of artificial intelligence are heeded. Thus, carefully designed safety features must be taken into consideration.

This work is mainly focused on the third point above mentioned, although bearing in mind the first three challenges. Concretely, the present study develops different RL-based TSCs and assesses their performance under diverse simulation scenarios, therefore offering a comprehensive and consistent comparison between different key TSC design elements. Such study is important given the fact that, while numerous studies propose different RL-based approaches to TLC, it is challenging and most of the time meaningless to compare the different works, due to the highly heterogeneous simulation scenarios and performance metrics. Despite the vast literature published on TLC, only a small minority of the works focuses their attention on the comparison of different RL methods for TLC [10–12]. Therefore, it is hard to infer from the literature what are the pros and cons of the different approaches. This study aims to continue this underexplored line of research, providing insightful contributions regarding the design of RL-based TSCs.

In contrast to the majority of the published works, the present thesis adopts a rather constrained action space definition. Such choice allows for the improved interpretability of the learned behavior, as well as the creation of TLC strategies that more closely adhere to the standards used by governmental transportation departments throughout the world [13]. In line with the most recent research in the field, a special emphasis is given to the use of deep learning techniques.

Additionally, in an attempt to unify the development of RL-based TSCs, this work also provides guidelines regarding the steps required for the creation of such controllers, from simulation setup, to performance assessment and comparison. The creation of such methodology allows for the fair and consistent comparison between the different studied approaches.

1.1 Contributions

The major contributions of this thesis are related to the experimental comparison between different RL-based approaches to TLC (experimental survey), providing worthy insights into how the choice of the key TSC design elements affects the observed performance. In summary, the following list enumerates the three main contributions:

- Propose a methodology for the development of intelligent TSCs that allows a fair and meaningful comparison between different approaches, from scenario extraction to performance estimation and comparison. The methodology contributes to the standardization of all the steps required to train and evaluate RL-based TSCs, providing new tools that allow for a more comprehensive and interpretable comparison of the performance of different methods.

- Validate the idea that RL methods exhibit useful behaviour in the context of TLC, providing quantitative and qualitative measures that support this finding. More specifically, show that, despite the fact that the present study makes use of a more constrained action space definition, RL controllers are still able to learn efficient strategies for TLC, comparing their performance with well-established transportation engineering methods.

- Study different RL-based approaches to TLC, namely investigating, both quantitatively and qualitatively, the effect of different TSC design parameters under a diverse set of scenarios extracted from real-world locations. It is provided a thorough comparison of several state and reward function definitions, as well as three RL algorithms: Q-learning, Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG).

A research article, covering the proposed methodology, was accepted for full presentation at the AI for Urban Mobility workshop of the Thirty-Fifth AAAI Conference on Artificial Intelligence. A journal article has been submitted to the Proceedings of the IEEE.

1.2 Organization of the document

The following chapter (Chapter 2) provides some additional background information required to understand the rest of the document. Chapter 3 presents and discusses the different approaches that contribute to increasingly efficient TSCs, from transportation engineering to RL-based methods. Chapter 4 presents the proposed methodology for the development of intelligent TSCs, from scenario setup to performance assessment, which is further concretized in the following chapter (Chapter 5) with the implementation of a set of scenarios and RL-based TSCs. The implemented controllers are then evaluated under different simulation scenarios, and the obtained results are presented and discussed in Chapter 6. Finally, Chapter 7 summarizes the findings and points out some possible future research directions.

2

Background

Contents

In this chapter, the two main topics on top of which deep reinforcement learning methods arise from, deep learning and reinforcement learning, are briefly described. The first two sections concisely describe, respectively, the deep learning and reinforcement learning fields. Finally, the last section provides a brief overview of some core concepts regarding traffic light control.

2.1 Deep learning

Deep learning [14, 15] is an area within machine learning that studies the application of Artificial Neural Networks (ANNs) under supervised, semi-supervised and unsupervised learning tasks. Recently, several RL-based approaches to TLC propose the usage of ANNs as universal function approximators, allowing to learn useful TLC strategies from high dimensional input data. A commonly used type of neural networks are the Convolutional Neural Networks (CNNs), which are specialized in the processing of data with a grid-like topology.

2.1.1 Artificial neural networks

A neural network is a model that maps an input vector $\boldsymbol{x}$ into an output vector $\hat{\boldsymbol{y}}$, through a series of chained non-linear transformations. The network defines a mapping $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters of the model. Moreover, the function f is represented by a composition of K different functions, i.e.,

$$f(\boldsymbol{x}) = (f^{(K)} \circ f^{(K-1)} \circ \cdots \circ f^{(2)} \circ f^{(1)})(\boldsymbol{x}), \tag{2.1}$$

where each function $f^{(k)}$, $k \in \{1, ..., K\}$ represents a layer of the model and is parametrized by a different set of parameters $\boldsymbol{\theta}_k$. For feedforward, fully connected neural networks, each layer computes the input of the next layer, $\boldsymbol{h}_k$, by applying a linear mapping between the output of the previous layer, $\boldsymbol{h}_{k-1}$, and a set of weights $\boldsymbol{W}_k \in \boldsymbol{\theta}_k$, adding a bias term $\boldsymbol{b}_k \in \boldsymbol{\theta}_k$, and applying a non-linear activation function a_k,

$$\boldsymbol{h}_k = \begin{cases} \boldsymbol{x}, & k = 0, \\ a_k(\boldsymbol{W}_k \boldsymbol{h}_{k-1} + \boldsymbol{b}_k), & k \in \{1, ..., K\}. \end{cases} \tag{2.2}$$

At present, the most popular activation function is the Rectified Linear Unit (ReLU) [16], defined as $a(z) = \max(z, 0)$. Commonly used last layer activation functions, a_K, include the identity function for regression tasks and the sigmoid or softmax functions for classification tasks.

Artificial neural networks are trained to minimize a loss function $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$ using Gradient Descent (GD), given a dataset $\mathcal{D}$. Loss functions quantify the error between the model's output $\hat{\boldsymbol{y}}$ and the true value $\boldsymbol{y}$, for each point $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$. Common loss functions include the mean absolute error or mean squared error for regression tasks and the cross-entropy loss or hinge loss for classification tasks. In

gradient descent optimization, the loss function — and thus the error — is minimized by iteratively updating the parameters of the model in the direction of the negative gradient of $\mathcal{L}$. For each parameter $\boldsymbol{\theta}_j \in \boldsymbol{\theta}$, the batch GD update is

$$\boldsymbol{\theta}_j^{(t+1)} = \boldsymbol{\theta}_j^{(t)} - \alpha \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \frac{\partial \mathcal{L}(\boldsymbol{x},\boldsymbol{y};\boldsymbol{\theta}_j^{(t)})}{\partial \boldsymbol{\theta}_j^{(t)}}, \tag{2.3}$$

where α denotes the learning rate. It turns out that, due to the functional decomposition of the model (Eq. 2.1), and through the chain rule of calculus, the partial derivatives $\partial\mathcal{L}/\partial\boldsymbol{\theta}_j$ can be efficiently computed for all $\boldsymbol{\theta}_j \in \boldsymbol{\theta}$ using the back-propagation algorithm [17]. Further variations of the batch GD update rule have been proposed, increasing the speed and stability of the learning process. Among these methods is the commonly used ADAM optimizer [18].

2.1.2 Convolutional neural networks

Convolutional Neural Networks [19] are a particular class of ANNs specialized in the processing of data that has a known grid-like topology, such as image data. A convolutional layer, the core building block of a CNN, consists of a set of learnable filters that are convolved across the width and height of the input. Using the backpropagation algorithm, the model learns filters that detect useful features in the input. A convolutional neural network is an ANN that includes at least one convolutional layer.

2.2 Reinforcement learning

Reinforcement learning [1] addresses problems that can be modeled as Markov Decision Processes (MDPs). An MDP is a discrete time, stochastic control process that provides a mathematical framework to model sequential decision making under uncertainty. The decision maker, called the *agent*, continually interacts with a stochastic *environment* by selecting *actions*. The environment responds to the agent by presenting new situations (*states*) and giving a numerical value, the *reward*, which the agent seeks to maximize over time through its choice of actions. Moreover, an MDP verifies the Markov property, which states that the conditional probability distribution of future states depends only upon the present state and action, not on the sequence of events that preceded it. In the context of the present work, this assumption seems more or less reasonable depending on the exact state definition. Nevertheless, previous works showed that, even in non-fully Markovian conditions, RL methods can attain useful behavior [6].

Formally, a Markov Decision Process is defined as a tuple (S, A, p, r, γ), where S is a set of states, A is a set of actions, $p : S \times A \to \mathcal{P}(S)$ is the transition dynamics distribution, where $\mathcal{P}(S)$ is the set of probability measures on S, and $r : S \times A \to \mathbb{R}$ is the reward function. At each time step, the agent

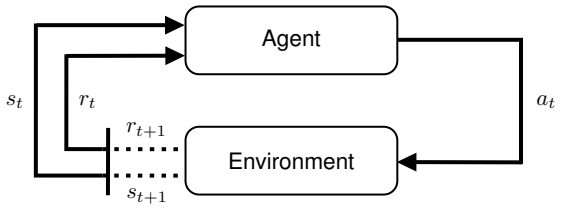


Figure 2.1: The agent-environment interaction loop (adapted from [1]).

observes the environment's state $s_t \in S$ and selects an action $a_t \in A$. As a consequence of the chosen action, at the next time step, the agent receives a numerical reward r_{t+1} with an expected value of $r(s_t, a_t)$ and the environment evolves to a new state $s_{t+1} \in S$ with probability $p(s_{t+1}|s_t, a_t)$. Figure 2.1 illustrates this interaction.

A *policy* $\pi(a|s)$, $\pi : S \to \mathcal{P}(A)$, is a mapping from states to probability distributions over actions. The agent's goal is to obtain a policy which maximizes the expected discounted cumulative reward. An optimal policy π^* is thus defined as

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s \right], \text{ for all } s \in S, \tag{2.4}$$

where the parameter $\gamma \in [0, 1)$ is the discount factor. Reinforcement learning algorithms usually involve the estimation of value functions, which capture the expected discounted cumulative reward while following policy π, such as

$$V^{\pi}(s) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s \right], \text{ for all } s \in S, \tag{2.5}$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, a_0 = a \right], \text{ for all } s \in S \text{ and } a \in A. \tag{2.6}$$

Given a complete description of the environment's dynamics, both optimal value functions V^* and Q^* can be computed using dynamic programming (*model-based* RL).

2.2.1 Temporal difference learning

Temporal Difference (TD) methods allow an agent to learn directly from raw experience in an online, incremental fashion, by estimating Q-values from observed *trajectories* (sampled sequences of states, actions and rewards). This class of methods is capable of attaining optimal behavior without requiring a complete model of the environment's dynamics (*model-free* RL). Temporal difference methods are divided into two main classes: *on-policy* methods attempt to evaluate the policy that is used to generate trajectories, whereas *off-policy* methods evaluate a different policy than the one that is used to generate

the data. In the off-policy setting, the policy used to sample trajectories is usually known as the *behavior* policy, while the policy being evaluated is referred to as the *target* policy.

Q-learning [20] is an off-policy TD method that directly approximates $Q^*(s, a)$, for all states s and actions a, using the following update rule

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right) \right], \tag{2.7}$$

where α denotes the learning rate. Convergence to the optimal policy is guaranteed if all state-action pairs are infinitely visited by the behavior policy. Usually, this is guaranteed with the use of an ϵ-greedy policy. An ϵ-greedy policy chooses, most of the time, an action that has maximal action value for the current state, but with probability ϵ selects a random action instead.

2.2.2 Value function approximation

Previously described methods have well-defined conditions that assure convergence to optimal behavior. Unfortunately, they do not scale with the size of the problem because it becomes impossible to estimate Q-values for each state-action pair individually. Instead, the agent must be able to generalize from previous experience.

In *function approximation* methods, the action-values $Q(s, a)$ are approximated by a differentiable function $\hat{Q}(s, a; \theta)$. A common goal is to use GD methods to find the model's parameters θ that minimize

$$J(\theta) = \mathbb{E}_{(s,a)\sim\beta} \left[\left(T - \hat{Q}(s, a; \theta) \right)^2 \right], \tag{2.8}$$

where β is the stationary (s, a) distribution under the behavior policy and T is the *target*: an estimate of $Q(s, a)$. The semi-gradient update, which ignores the fact that the target may also depend on the parameters θ, is given by

$$\Delta\theta = \alpha \left(T - \hat{Q}(s_t, a_t; \theta) \right) \nabla_\theta \hat{Q}(s_t, a_t; \theta), \tag{2.9}$$

where α denotes the learning rate. The Q-learning algorithm using function approximation is obtained by setting

$$T = r_{t+1} + \gamma \max_{a \in \mathcal{A}} \hat{Q}(s_{t+1}, a; \theta), \tag{2.10}$$

Deep Q-Network [6] is a well-known deep-RL method that uses a neural network as function approximator. It is a variation of the previously described Q-learning algorithm with function approximation that features two major modifications: (a) GD is performed with batches of experiences $(s_t, a_t, r_{t+1}, s_{t+1})$ that are randomly sampled from a buffer memory, known as the *replay buffer*; and (b) the target network parameters, i.e., the parameters of the neural network used to compute the target (Eq. 2.10), remain fixed (not updated) for a fixed number of successive steps. Further improvements to the standard DQN al-

gorithm include techniques such as double Q-learning [21], prioritized replay [22], dueling networks [23] and multi-step targets [24].

2.2.3 Deterministic actor-critic methods

All methods until now involve the estimation of state/action-value functions and then selection of actions based on those estimates, i.e., the policy is implicitly represented by state/action-value functions. However, we can also consider methods that instead learn a direct mapping from states to actions.

Policy gradient methods explicitly represent the policy with a parameterized function, differentiable with respect to its parameters. Consider a deterministic policy $\mu_\phi : S \to A$ and a stochastic behavior policy β. In the context of this work, the goal is to use GD methods in order to find the parameters ϕ that maximize the performance objective

$$J_\beta(\mu_\phi) = \sum_s \rho_\beta(s) Q^{\mu_\phi}(s, \mu_\phi(s)), \tag{2.11}$$

where $\rho_\beta(s) = \sum_{s'} \rho_0(s') \sum_{t=0}^{\infty} \gamma^t P_\beta^t(s|s')$ denotes the discounted state distribution under β, ρ_0 is the distribution of the initial state, and $P_\beta^t(s|s')$ denotes the density at state s after t steps (under β and starting at s'). The deterministic policy gradient theorem [25] establishes that

$$\nabla_\phi J_\beta(\mu_\phi) = \mathbb{E}_{s\sim\rho_\beta} \left[\nabla_\phi \mu_\phi(s) \nabla_a Q^{\mu_\phi}(s, a)|_{a=\mu_\phi(s)} \right] \tag{2.12}$$

A *critic* estimates $Q(s, a; \theta) \approx Q^{\mu_\phi}(s, a)$, off-policy, from trajectories generated by the behavior policy β, constructed by adding noise sampled from a noise process $\mathcal{N}$ to the *actor* policy,

$$\beta(s_t) = \mu_\phi(s_t) + \mathcal{N}. \tag{2.13}$$

The Deep Deterministic Policy Gradient [8] algorithm is a well-known RL method, in which the critic and the actor are parameterized by two neural networks. Similarly to the DQN algorithm, it uses a replay buffer where the tuples $(s_t, a_t, r_{t+1}, s_{t+1})$, generated by the behavior policy, are stored. Batches of experiences are sampled from the replay buffer and used to update both the actor and critic networks using batch GD. The actor is updated accordingly to Eq. 2.12, whereas the critic is updated in a similar way to Section 2.2.2 (Eq. 2.9), using the target

$$T = r_{t+1} + \gamma Q(s_{t+1}, \mu_{\phi'}(s_{t+1}); \theta'), \tag{2.14}$$

where ϕ' and θ' are the parameters of the target networks, a periodic copy of the ϕ and θ parameters.

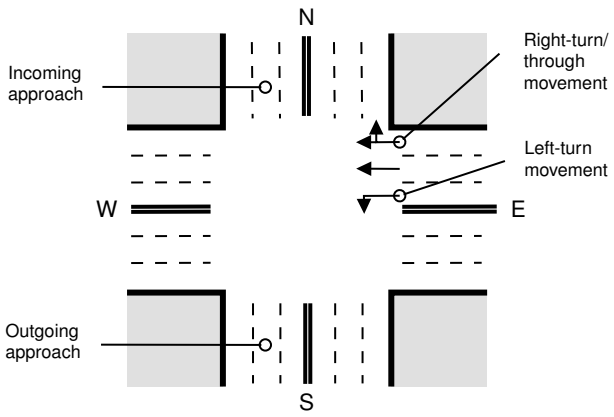


Figure 2.2: Intersection with four incoming approaches, each composed of three lanes.

2.3 Traffic light control basics

This section gives a brief overview of some important concepts regarding traffic light control, specifically the used terminology[1] and the main objectives in TLC.

2.3.1 Traffic light control terminology

A traffic infrastructure can be roughly represented as a network where *roads* and *junctions* are, respectively, the edges and nodes of a graph. A road connects two nodes of the network and has a given number of *lanes*, a maximum speed and a length. Traffic light controllers are typically installed at *intersections*, a common type of road junctions. Figure 2.2 illustrates a typical signalized intersection with four incoming and four outgoing *approaches*, each approach composed of three lanes.

A *signal movement* refers to the transit of vehicles from an incoming approach to an outgoing approach. It is generally categorized into the left-turn, through and right-turn movements. For example, East-South (equiv. to East left-turn) and East-West (equiv. to East through), are examples of signal movements accordingly to the intersection illustrated on Figure 2.2. A green signal indicates that the respective movement is allowed, whereas a red signal indicates that the movement is prohibited. A yellow signal is usually set as a transition from a green to a red signal.

A *signal phase* is a combination of non-conflicting signal movements, i.e., the signal movements that can be set to green at the same time. For example, (North through, South through and South right-turn) is a valid signal phase, however (North through, South left-turn) is not. A *signal plan*, for a single intersection, is a sequence of signal phases and their respective durations. Usually the time to cycle through all phases, known as *cycle length*, is fixed. Therefore, the *phase splits* - the portion of time allocated for each signal phase - are normally defined as a ratio of the cycle length.

[1] The presented terminology is adapted from Wei et al [4].

A traffic *demand* refers to the number of vehicles that are inserted to the network during a given time interval, whereas a *route* refers to a sequence of edges (*path*) that each of the vehicles traverses in order to reach its desired destination. A *trip* is a route that a given vehicle performs during a certain time interval.

2.3.2 Traffic light control objective

The major objective of traffic light control is to increase the efficiency of traffic infrastructures without compromising safety. But what exactly does it mean to increase the efficiency of a traffic infrastructure? While there is not a single answer to this question, different measures have been proposed to quantify the efficiency of traffic infrastructures:

- *Travel time* [26]: the average travel time for all vehicles in the network. For each vehicle it is defined as the elapsed time since the vehicle enters the system until it leaves the system.
- *Travel delay* [27]: the average travel delay for all vehicles in the network. The delay of a vehicle is defined as the time a vehicle has traveled within the environment minus the expected travel time.
- *Waiting time* [27]: the average stopped time for all vehicles in the network.
- *Number of stops* [28]: the total number of stops for all vehicles in the network.
- *Queue length* [27]: the total number of queued vehicles in the road network.
- *Throughput* [27]: the number of vehicles that completed their trip or crossed a given traffic infrastructure during a certain time interval.

The minimization/maximization of one or more of the aforementioned measures is, therefore, the main goal of traffic light controllers, being the minimization of the average travel time the most commonly used objective. The travel time metric is sometimes complemented or replaced by proxy metrics such as the waiting time, queue length, number of stops or throughput, being the later fundamentally different from the previous since it is capable of encoding information related with the vehicles' speed. Among these proxy metrics, the waiting time metric provides additional information in comparison to the queue length and number of stops metrics since it takes into account how long each vehicle has been waiting in queue and not just whether the vehicle is waiting or not. Eom et. al. [29] provide a thorough list of the several measures that have been proposed to quantify the efficiency of traffic infrastructures.

In the context of the present study and as commonly used in the traffic engineering field, we select the minimization of the average travel time as the main objective of RL-based TSCs. However, alongside the travel time metric, we also report secondary metrics such as the average waiting time and the average number of stops, the two being fundamentally different from each other.

3

Related work

Contents

Throughout the years, several methods have been used to develop increasingly efficient traffic signal controllers. This section gives an overview of the main approaches to traffic light control.

3.1 Methods in transportation engineering

Several traffic light control systems have been developed throughout the years by transportation research institutes and enterprises worldwide. The first automated TSCs were introduced during the 1950s, giving rise to a sequence of increasingly efficient generations of traffic control technologies. Along this evolution, TLC methods became progressively more robust and adaptable to changing traffic conditions. Developed approaches transitioned from offline to online control, from intersection to network level control, and from fixed-time to actuated/adaptive systems.

The first generation of TSCs comprises fixed-time methods and optimization techniques for offset calculation between neighboring intersections. Fixed-time control systems deterministically cycle through each of the light phases at constant time intervals. Usually, these methods take into account daily demand variability by dividing the traffic flow within a day into multiple periods and optimizing the traffic light controller settings for each interval independently. Among the first generation of controllers is the well-known Webster method [30], used to calculate the cycle length and phase splits for a single (isolated) intersection. While assuming that the traffic is uniform for a given time period, the Webster method derives closed-form equations to calculate the optimal phase splits and cycle length. It can be shown that the Webster method optimizes the total travel time for vehicles crossing the intersection. Other first generation controllers include, for example, the Maxband [31] and Greenwave methods. While these systems are simple and commonly used nowadays, they are usually not able to adjust to the changing traffic conditions and lack adaptability.

The following generations of traffic controllers comprise actuated/adaptive systems, which use input from sensors to dynamically adjust the signal plan scheme (cycle length and phase splits) or change the current signal phase. Therefore, actuated/adaptive control systems are more flexible than fixed-time systems because they are responsive, up to a certain extent, to changing traffic conditions. While some systems just use a set of rules to determine when to switch to the next signal phase, others use internal models to evaluate which traffic plans fit better the observed traffic conditions. The max-pressure [32] is an example of an adaptive controller that aims to improve the flow of vehicles between adjacent intersections. The *pressure* for a certain signal movement is defined as the difference between the number of vehicles on the incoming lanes and the number of vehicles on the outgoing lanes. The method aims to minimize the overall pressure for an intersection by picking, at each timestep, the phase with the maximum pressure. It can be shown that max-pressure control maximizes the throughput of the whole road network.

Several other actuated/adaptive systems, such as SCATS [33], SCOOT [34] and RHODES [35], were developed throughout the years. A comprehensive review of classical transportation engineering methods for TLC can be found in one of the published surveys/manuals [3, 36, 37].

While the previously described transportation engineering methods greatly contribute to increase the efficiency of traffic infrastructures all around the world nowadays, they feature some shortcomings. Most of the methods cast the TLC scenario into an optimization problem; however, they usually simplify the problem using unrealistic assumptions about the traffic flow. For most of the actuated/adaptive systems, the data gathered by ground sensors is often low quality (noisy and low-frequency), thus not providing enough information about the traffic state to the TLC system. In addition, the used traffic models are usually oversimplified. As a result, the majority of used TLC systems still rely a lot on manually designed traffic signal plans and are not truly able to adapt to real-time traffic patterns [4, 5].

While classical transportation engineering methods are commonly used in practice nowadays, the study of RL approaches to TLC has increasingly become more popular. Reinforcement learning methods provide a completely new approach to tackle the traffic light control problem. This new direction of research is, in fact, barely inspired by the previously described methods. Therefore, this work uses classical methods mostly as baselines for comparison with the proposed approaches.

3.2 Generalized optimization approaches

Besides the application of RL-based methods to TLC, other AI-related techniques such as fuzzy logic [38–42], genetic algorithms [43–47], swarm intelligence [48, 49], neural networks [50, 51], and dynamic programming [52] have been applied to the development of intelligent TSCs. Jácome, et al. [53] provide a concise review of the different AI-related techniques used in the development of intelligent TSCs.

3.3 Reinforcement learning-based traffic light control

In recent years, the study of reinforcement learning approaches to traffic light control has progressively become more popular, supported by the increasing amount of traffic-related data and computational power [2]. The use of reinforcement learning methods features one key advantage over transportation engineering methods: RL methods are capable of learning good control strategies without requiring a model of the system [9], therefore making them appealing to be used in the context of TLC.

The reinforcement learning framework straightforwardly accommodates the traffic light control problem: the agent represents the traffic light controller, the environment is the state of the traffic, and the actions are the traffic signal phases. However, proposed solutions vary on the exact MDP formulation and several different learning approaches are used, from model-based to model-free methods, from tab-

ular to function approximation methods, and from value-based to policy-based/actor critic methods. For multi-intersection scenarios, some works propose the use of multi-agent RL, while others decompose the problem into multiple independent sub-problems that are solved using single-agent techniques.

This section gives an overview of the RL literature for TLC. For convenience, the analysis of the different approaches is broken down into two main parts: Section 3.3.1 explores how the TLC problem can be modeled into the MDP framework, detailing and discussing the most used state, action and reward definitions among the research community; Section 3.3.2 provides a review of the broad and diverse RL-based methods proposed to address the TLC problem. The first section is, therefore, more focused on the description of the MDP formulation, whereas the second is mostly focused on providing an algorithmic overview of the used RL methods, as well as describing and discussing the most important works. Further details can be found in one of the published surveys addressing this topic [4, 54].

3.3.1 Markov decision process formulation

A very important step towards the development of RL-based traffic signal controllers is to formulate the TLC problem under the MDP framework. In the context of this domain, it is usually impossible to fully describe the underlying MDP, especially because the transition probability function is unknown or too complex to be modeled. Hence, in the majority of the works three main components are modeled: the action space, the state space, and the reward function.

Action space definition

Defining the action space is one of the most important steps in the development of RL-based TSCs since it determines the flexibility of the agent, i.e., how it is able to affect the environment's state through the selection of its actions. Therefore, this choice may greatly limit performance and restrict the state space and reward function definitions, as well as the used RL method. Thus, this design choice should be carefully considered at an early stage of development.

The literature proposes four different categories of action space definitions:

- *Set phase split* [55]: the cycle length has a fixed duration, and the action consists of setting the phase splits for the next cycle. Usually, the splits are picked from a pre-defined set of candidate time ratios.
- *Set phase duration* [28]: the action consists of setting the duration of the current signal phase. Usually, the phase duration is picked from a pre-defined set of candidate time periods.
- *Keep or change phase* [56]: the TSC decides, at each time-step, whether it changes to the next phase or keeps the current phase for one more time-step.

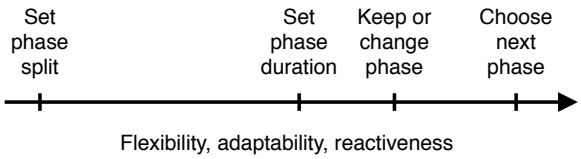


Figure 3.1: Illustration of the relative flexibility of the different action space definitions (not to scale).

- *Choose next phase* [26]: at each time-step, the action consists of picking the next phase to be executed. If the selected action corresponds to the current active phase then the green time for that phase is extended.

It is important to understand that the four definitions exhibit different degrees of flexibility and reactiveness. Figure 3.1 displays the overall flexibility trend for the different definitions. The *set phase split* is clearly the less flexible, since the cycle time and phase order is kept fixed. Furthermore, the agent is only able to interact with the environment at the beginning of each new cycle. Therefore, its reactiveness is lower in comparison with the other definitions. The *set phase duration* and *keep or change phase* definitions, on the other hand, do not have a fixed cycle time (they are acyclic) and, depending on the exact implementation, may be able to skip phases, therefore being more flexible. The latter, *keep or change phase* definition, exhibits a higher degree of reactiveness in comparison to the former, *set phase duration*, since its frequency of interaction with the environment is higher. Finally, the *choose next phase* exhibits the highest degree of flexibility as it is acyclic, and able to jump to an arbitrary phase (it does not need to go through the phases sequentially). It is also highly reactive since it interacts with the environment at each time-step. These different levels of flexibility and reactiveness play a key role in the time-scale adaptability of the agent. While for the more reactive definitions the agent is able to proactively adjust to traffic patterns that emerge within seconds, for the less reactive definitions the time-scale is rather coarser (minutes instead of seconds).

The majority of the works use the *choose next phase* action definition, arguing that it allows a higher degree of flexibility. However, a similar behaviour can be achieved with the *set phase duration* or *keep or change phase* approaches by allowing a signal phase time of zero, as suggested by Aslani et al. [28]. On the other hand, the fact that the *set phase split* definition uses a constant cycle length, in contrast to the other definitions, is advantageous as it may allow an easier synchronization between TSCs at adjacent intersections.

While highly flexible action definitions are the norm among the research community, the race towards more reactive TSCs unveils some important shortcomings. As the flexibility of the action space increases, it becomes easier for the agent to exploit the traffic simulator and learn strategies that yield a good performance in simulation but may be useless in practice, and which may even violate safety standards. As an example, the majority of the works allow the agent to arbitrarily skip phases. While this

may look appealing due to the fact that the agent is able to skip phases whenever they are empty, it also opens up the possibility for the agent to completely neglect phases that have a very low traffic demand (e.g. only give green time to certain phases every 5 minutes). This is clearly unfair and, furthermore, can be undesirable as there may be pedestrians waiting to cross the street. Thus, it is important to keep in mind that the agent should guarantee a minimum green time for all phases, something that can be easily ensured with the less flexible *set phase split* action definition.

From the previous discussion, it appears that there is not a closed answer to which action definition is the best. However, it is certain that a successful application of RL methods for TLC needs to appropriately balance a tradeoff between the flexibility and the fairness/safety of the action space definition. The duality between efficiency and safety is however, common across the traffic engineering field, as "*safety may be seen as an element needed to be sacrificed in order to achieve improvements in efficiency and meet ever-increasing demands. The reality is that traffic signals can, and in fact must, serve both operational efficiency and safety based on the conditions.*" - U.S. Department of Transportation, Traffic Signal Timing Manual, p. 1-2 [5].

State space definition

Various definitions of state space have been proposed, and the literature is very diverse with respect to this matter. State space definitions vary from simple representations such as the number of vehicles waiting (queue lengths) [57], to more complex representations that take into account the position and velocity of the vehicles approaching an intersection [26]. The following elements/features[1] have been proposed to describe the environment's state in the context of TLC:

- *Queue length* [27, 57]: the total number of waiting vehicles (stopped or travelling with a speed that is below a certain threshold) at each phase. One common state definition is to use the maximum queue length associated with each signal phase.
- *Waiting time* (delay-based) [11]: the total time vehicles spent in queue (stopped or travelling with a speed below a certain threshold) for each of the signal phases.
- *Volume* [28]: the total number of waiting and approaching vehicles for each phase of the intersection.
- *Position of vehicles* [26]: the position of the vehicles is usually represented with a boolean matrix that discretizes the approaches of the intersection into segments, where a value of one indicates the presence of a vehicle on that location, and a value of zero represents the absence of a vehicle on that location.

[1] The terminology regarding the state space and reward definition is not consensual in the research community, therefore, a mismatch between the names presented in this document and the ones that appear in the cited articles may occur.

- *Speed of vehicles* [26]: the speed of the vehicles for a given phase, usually normalized by the speed limit of the respective lane. The velocity of the vehicles can be either aggregated into a scalar by averaging, or encoded in a real-valued matrix, in a similar way to the previous feature.

Most works combine one or more of the aforementioned elements in the construction of a state space representation. Depending on the action space definition, the current signal phase and/or the elapsed time since the last phase switch, may also be added to the state.

Recently, the use of more complex state definitions is increasingly becoming more popular. These high dimensional representations are used in the hope to gain better insights about the traffic state. However, they make RL agents less sample efficient and may not necessarily boost performance [58]. Genders et al. [58] model and compare three state definitions with different resolutions, from occupancy and average speed to individual vehicle position and speed. Reported results show that low-resolution state representations perform almost identically to high resolution state definitions.

One important aspect resides in the fact that each state representation satisfies the Markov property to a different extent. More descriptive state definitions, such as the ones that take into account the position and velocity of the vehicles are more "Markovian" in comparison to simpler, less descriptive definitions. Therefore, given the assumptions stated in Section 2.2, more descriptive definitions may be more suitable to be used in the context of RL.

Additionally, it is important to keep in mind that, for real-world deployment, the TSC must be able to infer the current environment's state using the available technologies (sensors, cameras, etc.). Therefore, while some state space definitions may look appealing in the simulated environment, they may not be suitable for real-world deployment as it may be difficult, or even impossible, to infer the state due to technological constraints. This concern is mainly directed to more descriptive and complex state representations, such as the state definitions that use the exact position and velocity of the vehicles.

Reward function definition

Finally, various reward functions have been proposed to address TLC. It turns out that, while the minimization of the average travel time is a commonly used objective in the development of TSCs, it is not well suited to be used as a reward function in the context of RL-based methods, due to the fact that the total travel time for each vehicle is unavailable at decision time [59]. Moreover, the reward should be computable from the state and action alone, thus observability issues with the state also influence the choice of reward.

In the literature, the reward function is usually defined as a weighted sum composed of one or more of the following elements:

- *Queue vehicles* [27, 57]: similar to the definition of the *queue length* state space, this reward formulation aims to minimize the queue lengths for all phases of the intersection. As an example,

Oliveira et al, 2006 [60] define the reward as the difference of the sum of squared maximum queue lengths between decision steps.

- *Waiting time* (delay-based) [11, 26]: aims to minimize the time spent in queue by the vehicles. For example, Nishi et al, 2018 [59] define the reward as the negative sum of the waiting times of all vehicles at the intersection.
- *Volume* [28]: minimize the volume of the intersection's incoming approaches.
- *Throughput* [27, 56]: maximize the number of vehicles that cross the intersection in a given time period.
- *Number of stops* [11]: minimize the number of stops for all the vehicles. Intuitively, the less the number of stops, the smoother the traffic flows.
- *Speed* [55]: maximize the speed of the vehicles. Intuitively, the higher the average speed, the sooner the vehicles will reach, on average, their desired destinations.
- *Pressure* [61]: minimize the pressure of the intersection, i.e., the difference in volume between the incoming and outgoing approaches.

All previous reward definitions can be implemented in two different ways: (a) as an absolute value at a given decision point; or (b) as a difference between two consecutive decision points. However, it is still not well understood which implementation is the best and it is quite likely that the best formulation depends on the action space definition.

In an attempt to study how the choice of reward function impacts the performance of RL-based TSCs, Touhbi et al. [27] explore several reward definitions in a single real-world intersection from downtown Toronto. The developed TSC is based on a tabular Q-learning algorithm which uses a variation of the *queue length* state space representation to decide when and to which signal phase the system should change. The results show that the performance of the proposed method is highly influenced by the reward definition and traffic volumes at the intersection. Among the tested reward definitions (queue length, delay-based and throughput), for high traffic volume, the *delay-based* reward showed improved performance, achieving reduced queue lengths and average vehicles' delays. In contrary to the other definitions, the *throughput* reward revealed to be more unstable, producing heterogeneous results.

3.3.2 Methods

The reinforcement learning literature for traffic light control is very rich with respect to the variety of proposed methods. The present section provides an overview of those approaches by segmenting the works according to their RL taxonomy (model-based or model-free, value-based or policy-based, etc.).

In order to keep the review short and concise, the exact problem formulation is omitted for the majority of the works. However, we note that the choice of the learning method is not independent from the MDP formulation.

The first major difference characterizing the proposed works depends on whether the used method relies on a detailed transition probability function or not, i.e., if it is model-based or model-free. While the large majority of the methods are model-free, some works propose the use of model-based approaches [62–65] that primarily rely on planning using dynamic programming techniques. For example, Wiering et al. [63] use a discrete model-based algorithm that aims to minimize the overall vehicles' waiting time by estimating road-user-based value functions [62].

In the context of this work, model-based approaches to TLC are not considered due to the limitations associated with this class of methods. While model-based methods are appealing due to improved sample efficiency, in the context of TLC it is usually impossible to provide a complete description of the environment's dynamics given the highly unpredictable traffic behaviour. Therefore, we follow the research community and focus on model-free methods to TLC.

Tabular methods

The first model-free methods used in the development of TSCs consisted of the application of tabular Q-learning [27, 66] or SARSA [67, 68] algorithms. As an example, Abdulhai et al. [66] propose the use of tabular Q-learning to control an isolated two-phase intersection. The developed TSC measures the queue length for each of the incoming approaches of the intersection and decides whether to extend or change the current signal phase in such a way that the average number of waiting vehicles is minimized. Presented results show that the developed controller is able to reduce up to 44% of the delays, in comparison to a pre-timed controller.

El-Tantawy et al. [11] provide a comparison between Q-learning and SARSA based on three state definitions and four reward functions, in a real-world traffic network of downtown Toronto. Despite the fact that both algorithms generally outperformed fixed-time and actuated controllers, experimental results showed no significant performance difference between the Q-learning and SARSA algorithms.

Even though tabular methods offer theoretical convergence guarantees to optimal behaviour, they are not suitable to handle large or continuous state spaces. Therefore, the previously presented approaches were restricted to the use of simpler state representations such as queue lengths, volumes or cumulative delays. Furthermore, the large majority of the state spaces are inherently continuous. As such, a carefully tuned discretization procedure needs to be carried in order to be able to use this class of RL methods. Nevertheless, the results presented in the aforementioned works are promising [11]. As such, it is important for tabular methods to be included in our study.

Function approximation methods

The use of function approximation methods, namely deep learning techniques, is undoubtedly at the core of current RL-based TLC research. Although some works propose the use linear approximators consisting of radial basis functions [28,69], triangular-shaped functions [69] or tile-coding [28,69,70], the majority use ANNs and consist of variations of the well-known DQN algorithm [26,56,59,71–73]. High dimensional state space representations that take into account the position and velocity of the vehicles are the norm, while varied action and reward definitions are used.

Genders, et al. [26] use a convolutional neural network with DQN to estimate action-values. The input of the neural network consists of a matrix that encodes both vehicles' positions and velocities, as well as the current signal phase. Authors argue that this raw state space representation is able to provide a superior insight about the traffic conditions in comparison to more abstract representations such as the number of queued vehicles or the vehicles' flow. The TSC decides, at each decision step, which is the most adequate signal phase from a set of pre-defined actions. The reward is defined as the change in vehicle delay between decision steps. Experiments with a one-intersection topology showed improved results in comparison to a shallow network that used only the number of queued vehicles and the current signal phase as state space.

Further works proposed the use of alternative ANN architectures [59,71,72,74]. Choe et al. [72], for example, also use a DQN-oriented approach but combine a multi-layer perceptron with a Long Short Term Memory (LSTM) cell, allowing the agent to memorize some information across time steps. Reported results show that the proposed architecture outperforms the standard ConvNet-based DQN, reducing the overall waiting time of the vehicles. Zheng et al. [73] propose a neural network architecture specifically designed to tackle the TLC problem. The developed network is phase-invariant and prioritizes signal phases with a higher demand. According to the presented results, the proposed method achieves improved sample efficiency and outperforms fixed-time methods, actuated systems, as well as standard DQN methods such as [56,75].

Even though it is not straightforward to understand from the previous works whether the use of function approximation methods actually improves performance over the simpler tabular methods, Shabestary et al. [76] suggest that, indeed, function approximation methods can outperform discrete methods. The work develops a TSC based on a deep Q-network and compares its performance with a tabular Q-learning version that uses queue lengths as state and is trained to minimize the average cumulative delay. Experimental results using a single isolated intersection show that the DQN-based method outperforms the discrete Q-learning algorithm for all evaluation metrics. However, it is important to notice that the evaluation scenario consists of an oversimplified traffic topology with just one intersection. As such, the presented results may not generalize to other settings.

Actor-critic methods

Finally, actor-critic methods were also proposed for the development of TSCs [28, 55, 77, 78]. One of the major advantages of these methods is that they are able to directly handle continuous action spaces, as opposed to the previously presented works.

Aslani et al. [28] propose and compare different actor-critic algorithms based on different types of function approximation, such as tile coding and radial basis functions. The state space is composed of two elements: the current active signal phase and the volume of the lanes leading to the intersection. At the beginning of each signal phase, the TSC is able to pick the duration of the current green phase from a set of discrete values. The reward is defined as the negative total volume for the intersection. The results show that the proposed actor-critic architectures achieve improved performance in comparison to fixed-time, actuated systems, and discrete state approaches.

Casas [55] controls the phase splits of a signal plan with a DDPG algorithm. In this work, the TSC is provided with the average speed of the vehicles captured by ground sensors and is trained to maximize the average speed for all vehicles in the network. In contrast to Aslani et al. [28], the proposed action space is continuous.

3.3.3 Discussion

Given a multi-intersection traffic infrastructure, the problem of RL-based TLC can be seen as an inherently cooperative game between different agents, each responsible for the control of the traffic lights at a given set of intersections. In its simpler, *fully decentralized* form, as in most RL-based approaches presented in the previous section, an RL controller is developed for each of the intersections, and no explicit coordination mechanisms are considered between agents [28].

A fully decentralized approach presents some advantages, such as minimal communication overhead. However, it is important to notice that the actions taken by one agent often influence the performance of the neighboring agents, and poor coordination significantly impacts performance [79]. Consequently, it may be important for each agent to be aware of the intentions of its neighbors and that they, cooperatively, learn useful strategies that not only improve traffic conditions locally to each intersection, but also improve the global road network performance.

Several different coordination strategies have been proposed to address RL-based TLC. Some works are focused on the study of fully centralized approaches where a single controller jointly outputs the actions for all TSCs in the network [55], while other approaches explicitly consider coordination mechanisms among adjacent agents using coordination graphs [75], or optimize a joint reward function [59]. Unfortunately, such approaches do not come without their own caveats, despite the fact that the advent of deep learning greatly contributed to improve the scalability of the methods [15].

Another key aspect in the development of RL-based TSCs resides on the robustness of the resulting policies. It is important for the TSC to reliably perform in highly dynamic urban environments, adapting to unpredictable changes in traffic patterns, as well as traffic disruptions, accidents, sensor failures, among others. Rodrigues et al. [57] assess the performance of several RL configurations under different scenarios, namely under demand surges caused by special events, capacity reductions from incidents, and sensor failures. Empirical results show that, in order for deep RL controllers to be robust to these uncertainties, they need to experience these scenarios during training. Aslani et al. [28] developed another work that considers a wide variety of traffic scenarios. The robustness of the different algorithms is tested under scenarios comprising events such as traffic incidents and impatient pedestrians crossing, as well as noisy sensor data.

In spite of the rich literature that explores the use of RL for TLC, it is hard to understand what is the relative performance of each approach. This is due to the fact that the works are highly heterogeneous with respect to the experiments' setup: different evaluation metrics and simulation scenarios are used in different works, making it difficult, often even impossible, to compare different approaches.

Due to this fact, it is not straightforward to understand from the previous discussion what are the most suitable MDP formulations to tackle the TLC problem. Works such as El-Tantawy et al. [11] provide some hints with respect to different MDP formulations that are worth to be implemented and compared, however, there is still a lot left to understand. For example, with respect to the different proposed action spaces, it is still not clear whether highly flexible definitions are required to attain state-of-the-art performance. More restrictive action spaces should not be forgotten, as they are more aligned with current transportation engineering policies [13].

Regarding the proposed RL algorithms, it is also hard to infer their relative performances as the majority of the works are more focused on proposing new RL methods for TLC rather than comparing different approaches. The work of Aslani et al. [12] is one of the few focused on comparing the performance of different RL methods for TLC: the performance of Q-learning, SARSA, and actor-critic algorithms for both discrete and continuous state spaces is assessed. Genders & Ravazi [10] provide a comparison between two RL methods, DQN and DDPG, on a synthetic scenario comprising two intersections. However, aside from a small minority of works, such as these two, it is hard to grasp the wider picture, regarding the pros and cons of the different RL-based approaches to TLC.

Finally, it is worth to mention that, while the previously presented works seem to support the hypothesis that, indeed, RL-based TSCs may attain state-of-the-art performance for TLC, the analyses of the methods carried out in a number of studies feature some shortcomings. The following list summarizes some of these flaws (related to the challenges introduced in Chapter 1), which should be carefully taken into consideration in future research:

1. *Incomplete evaluation*: some works do not provide a complete evaluation of their proposals and/or

do not compare the obtained results with traditional TLC methods. Some provide comparisons with fixed-time and simple actuated systems, however, modern adaptive systems are usually not taken into account. Moreover, the majority of the works do not carefully study the resulting policy and do not provide comparisons with traditional transportation engineering policies (challenge number 3).

2. *Highly heterogeneous evaluation metrics and simulation scenarios*: it is hard to compare different approaches due to the various simulation scenarios and performance metrics used among works. The TLC problem lacks benchmarking scenarios (challenge number 3).

3. *Oversimplified traffic network topologies*: a good number of works evaluate the performance of the developed methods using oversimplified topologies that are not well-representative of real road networks (challenge number 2).

4. *Oversimplified and unrealistic traffic flows*: the majority of the works use simulated traffic flows that are not supported by real traffic data. Moreover, works seldom take into account the highly variable traffic patterns that emerge in urban areas, and less common events such as traffic disruptions or accidents (challenge number 2).

5. *Not supported choice of design parameters*: most works do not empirically justify the choice of the design parameters, especially the choice of problem formulation: state space, action space and reward function (challenge number 3).

The work in this thesis is specifically developed bearing all previous issues in mind. It provides a consistent comparison between different classes of RL methods, using a complete and meaningful set of metrics, for quasi-realistic traffic simulations. The work also comprises the study of different design parameters. Thus, the present study continues the line of research introduced by Aslani et al. [69] and Genders & Ravazi [10], studying a fully decentralized approach to TLC.

In contrast with most works addressing RL-based TLC, we adopt a constrained action space definition, where each agent is able to adjust the phase splits of a given intersection. Three factors support this choice: (a) the agents' behaviour is more inline with the expectations of transportation engineering departments; (b) the restricted action space definition allows for improved interpretability of the resulting policies; and (c) since the agents have a fixed cycle length, it is easier to coordinate adjacent controllers - therefore a certain degree of synchronization can still be ensured even though no coordination mechanisms are explicitly considered. The work assesses the performance of different RL methods (Q-learning, DQN and DDPG), as well as different state space and reward function definitions, under different simulation scenarios.

The following chapter proposes a methodology for the development of intelligent TSCs that allows a fair and meaningful comparison between different approaches, further concretized, in Chapter 5, with the implementation of a set of MDP formulations and RL controllers.

4

Methodology

Contents

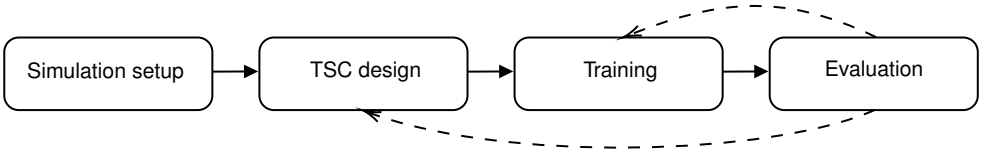


Figure 4.1: Diagram illustrating the proposed methodology, composed of four stages. Solid arrows denote the main development flow, whereas dashed arrows denote the iterative process of model tuning.

This chapter proposes a novel four-stage methodology for the development of RL-based TSCs that aims to mitigate some of the shortcomings identified in the previous chapter, allowing to properly develop and fairly assess the performance of different TSCs. Figure 4.1 illustrates the proposed methodology, comprising four phases: simulation setup, TSC design, which consists of the MDP formulation and selection of the RL method, training and evaluation. In order to make the exposition clearer, a running example concretizes the methodology by implementing a DQN agent for a simplified traffic network. A careful discussion of the obtained results is postponed to Chapter 6.

4.1 Simulation setup

The first stage of the proposed methodology is the simulation setup phase.[1] As previously discussed, RL controllers must be trained by resorting to simulators that are able to provide a realistic response to the agent's actions during the learning process. The main objective of the simulation setup stage is to prepare all the simulations needed to carry out such training. It includes gathering simulation-related data, such as the topological data of the roads network and vehicles demand/routes data, as well as setting up the traffic simulator.

4.1.1 Traffic simulation

Throughout the years, several traffic simulators have been developed, contributing to the improvement of urban mobility in a number of ways. Among other purposes, simulators are used to prototype new infrastructures and evaluate changes to current traffic management policies. Nowadays, there are several softwares available for urban mobility simulation, such as SUMO [80], AIMSUM [81] and Paramics [82]. Some programs focus on the behaviour of each individual vehicle (microscopic models), while others only simulate the flow of the traffic (macroscopic models). Moreover, softwares not only differ on the background calculations used for trajectory estimation, but also on aspects such as the customizability

[1] Although a detailed discussion is outside the scope of the present study, it is important that the research community agrees on a set of benchmark environments/traffic networks that may be used as a first test stage for the algorithms explored in the context of TLC [10]. The existence of such benchmarks would enable a proper comparison of different models and algorithms in a common set of environments, enabling a clearer assessment of the strengths and weaknesses of different alternatives.

of the simulation, visualization and scalability. For a thorough comparison of the main traffic simulators we refer to [83, 84].

This work uses the SUMO micro-simulator. The fact that the software is open-source, widely used, properly documented, and supported by the deep RL framework FLOW[2], are the main factors that support this choice.

4.1.2 Road networks topology

Open source services such as OPENSTREETMAP[3] allow to extract geospatial data from segments of cities' districts. During the simulation setup stage such information can be prepared and fed to the simulator, thus opening up the possibility of simulating a rich set of networks relevant to real-world traffic signal control.

The present study considers real-world topologies from Lisbon's downtown. Proper setup of the configuration files needed by the SUMO simulator involves: (a) extracting the region of interest from OPENSTREETMAP and opening the resulting file with JOSM[4], an extensible editor for OPENSTREETMAP files, in order to fine-tune the network; (b) converting the (edited) OPENSTREETMAP file into the SUMO network format using the `netconvert`[5] tool; and (c) opening the resulting SUMO network file with the `netedit`[6] tool, a graphical network editor for SUMO, in order to ensure that all TSCs are properly setup, namely check whether all traffic phases and links (connections between lanes) are correct.

The running example starts off by extracting a simple traffic network composed of a single two-phased intersection. Figure 4.2 displays two screenshots of the intersection, extracted from Lisbon's metropolitan area, near Marquês de Pombal square. The previously presented steps were followed in order to transform the OPENSTREETMAP file, displayed in Figure 4.2(a), into the final SUMO file, displayed in Figure 4.2(b). As it can be seen, the network has been rotated using the JOSM editor for aesthetic reasons.

4.1.3 Traffic demands and routes

Traffic demands and routes can be either synthetic [56], or derived from real-world data using origin-destination matrices [28] or induction loops counts [57]. While data-driven estimated demands and routes open up the possibility of running a set of situations resembling real-world observations, the simulation setup is considerably harder. This is due to the fact that additional steps need to be taken in order to properly pre-process the data, namely requiring manual validation and, most of the times,

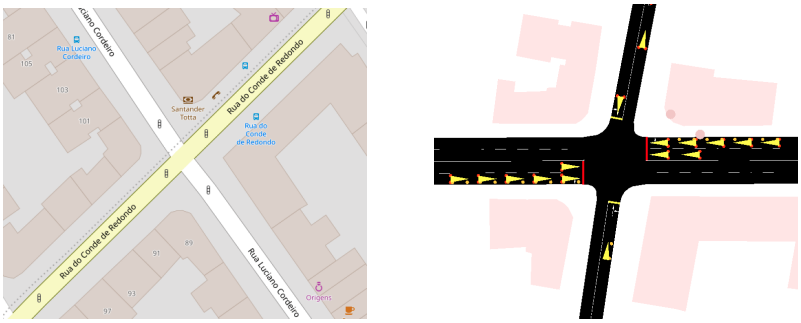
[2] FLOW [85] is an open-source Python framework that provides an accessible way to solve vehicle and traffic control problems, enabling the fast prototyping and evaluation of RL-based TSCs.

[3] https://www.openstreetmap.org

[4] https://josm.openstreetmap.de/

[5] https://sumo.dlr.de/docs/netconvert.html

[6] https://sumo.dlr.de/docs/netedit.html



(a) OPENSTREETMAP screenshot.

(b) SUMO screenshot.

Figure 4.2: *Intersection* network composed of two streets: Rua Luciano Cordeiro (vertical) and Rua do Conde de Redondo (horizontal). The network is composed by a single two-phased intersection. Phase 1 allows the movement of vehicles in the vertical direction, whereas phase 2 allows the movement in the horizontal direction.

scenario simplifications [86]. Moreover, depending on the selected city area, the data may be very low-frequency, noisy and erroneous, or even unavailable [87]. Although bearing in mind that the use of real-world data for demands/routes estimation is important as it may help to close up the simulation-reality gap, such topic is outside the scope of this work and, therefore, the present study considers synthetic demands/routes. Nevertheless, additional efforts are taken in order to make the simulations as realistic as possible.

We consider synthetic demands defined, for each network, as a mapping $d : \mathbb{N} \rightarrow [0, 1]$, where $d(k)$ defines the probability of inserting new vehicles at each second of the simulation, for each k-laned source edge of the network, i.e., $d(1)$ defines the insertion probability of all one-laned source edges of the network, $d(2)$ defines the insertion probability of all two-laned source edges, and so on. For the simplest demand type considered, named *constant* demand type, the mapping d is kept unchanged throughout the simulation.

With respect to the routes generation procedure, the `duarouter`[7] tool is firstly used in order to retrieve the shortest routes for all source-sink edges pairs. Afterwards, the routes are weighted by a synthetic procedure, but aligned with real-world traffic patterns: for each source edge, each of the possible shortest routes r is assigned a weight given by $w_r = 1/(\text{numberOfTurns}(r) + 1)$. For each source edge, the calculated weights are then normalized using a softmax function with a temperature coefficient. This helps to make the resulting distribution skewed. Accordingly to experimental testing, this criteria was picked due to its simplicity and good results: it greatly decreased the number of gridlocks[8] while keeping

[7] https://sumo.dlr.de/docs/duarouter.html

[8] A *gridlock* occurs when a queue from one bottleneck creates a new bottleneck somewhere else, and so on in a vicious cycle that completely stalls the circulation of vehicles [88].

the routes generation procedure stochastic. Furthermore, it is supported by the real-world evidence that vehicles tend to follow a certain cardinal direction while traveling on a grid network.

In our running example, the simulation setup is completed at this point by defining the traffic demands mapping, and weighting the possible routes using the described procedure. Specifically, the mapping $\{d(1) = 0.1, d(2) = 0.22\}$ is used: as it can be seen, the defined insertion probabilites are roughly proportional to the number of lanes of the respective edge.

4.2 Traffic signal controller design

The second stage of the presented methodology consists of the description of the traffic light control problem as a Markov decision problem (or a multiagent version thereof), as well as selection of the RL method to be used as a learning component for the TSC.

4.2.1 Markov decision process formulation

As seen in Sections 2.2 and 3.3, an MDP comprises five elements: the state space, the action space, the transition probabilities, the reward function, and the discount factor. Most works applying RL to the TLC domain do not specify a transition dynamics distribution, therefore, three key components need to be specified: the state space, i.e., the information upon which the agent will base its decisions, the action space, i.e., how the agent is able to influence the environment through the choice of its actions, and the reward function, which implicitly encodes the goal of the agent.

The first and perhaps most important component to define is the action space of the agent. As previously discussed, this choice reveals to be fundamental since it may greatly impact the final performance of the controller. Afterwards, the definition of the state space and reward functions should follow. Following our discussion in Chapter 3, we implement a set of definitions and compare their performance. Depending on the exact action space definition, it may be necessary to time-aggregate information in order to calculate the observed states and rewards.

In our example scenario composed of a single intersection, the running example continues with the definition of the underlying MDP. The action space is set to the *set phase split* definition since it allows to easily ensure a fair and safe behaviour. Concretely, an arbitrarily sized set of signal plans, given a priori to the system by a user, defines the action space. We consider seven signal plans, i.e., seven actions. Table 4.1 shows the allocations for each of the actions. The cycle length is fixed to 60 seconds, optimized via grid-search, and the yellow signal time to six seconds.

Table 4.1: Signal plans that define the discrete action space. The displayed values are percentages of the total cycle length (60 seconds), thus each phase includes a six seconds yellow signal by default. As an example, action (30,70) produces (assuming that the signals for all phases except the one that is currently active are set to red): (i) 12 seconds of green signal to phase 1; followed by (ii) six seconds of yellow signal to phase 1; followed by (iii) 36 seconds of green signal to phase 2; followed by (iv) six seconds of yellow signal to phase 2.

Action/Signal Plan	Phase 1 allocation	Phase 2 allocation
(30,70)	30.0%	70.0%
(36,63)	36.6%	63.4%
(43,57)	43.3%	56.7%
(50,50)	50.0%	50.0%
(57,43)	56.7%	43.3%
(63,37)	63.4%	36.6%
(70,30)	70.0%	30.0%

With respect to the state space and reward definitions, we implement a waiting time-based formulation [59]. The state consists of a two dimensional vector, $(w_c^{p_1}, w_c^{p_2})$, containing the observed waiting times for each of the phases, p_1 and p_2, at a certain cycle c. Each component is defined as

$$w_c^{p_i} = \frac{1}{K} \sum_{k=0}^{K-1} \sum_{l \in L_{p_i}} \sum_{v \in V_l^k} \text{stopped}(v, k), \tag{4.1}$$

with

$$\text{stopped}(v, k) = \begin{cases} 1 & \text{speed}(v, k) < 0.1 \\ 0 & \text{otherwise,} \end{cases} \tag{4.2}$$

where K is the cycle length, L_{p_i} denotes the set of lanes associated with phase p_i, V_l^k denotes the set of vehicles traveling on lane l at the timestep k, and $\text{speed}(v, k)$ gives the normalized velocity of vehicle v at timestep k. The action-independent reward is defined as

$$r_c = -\sum_{i=1}^{2} w_c^{p_i}. \tag{4.3}$$

The agent is thus penalized if the average waiting time of the vehicles is high. The discount factor γ is set to 0.98.

4.2.2 Reinforcement learning methods

The TLC domain is inherently continuous, i.e., the information used for state creation is usually real-valued. Furthermore, the different features that compose the state space often exhibit different ranges and scales, which vary from phase to phase and from intersection to intersection. Therefore, with respect to tabular methods, an appropriate discretization procedure must be developed, for example with the usage of tile-coding. On the other hand, for RL methods that rely on function approximation, such as

the DQN algorithm, the number of parameters of the model should be appropriately tuned. In some cases, it may be necessary to normalize the inputs before feeding them into the function approximator. If this pre-processing step is not carefully addressed, it will limit the final performance of the agent and, in more extreme cases, make the learning procedure unstable.

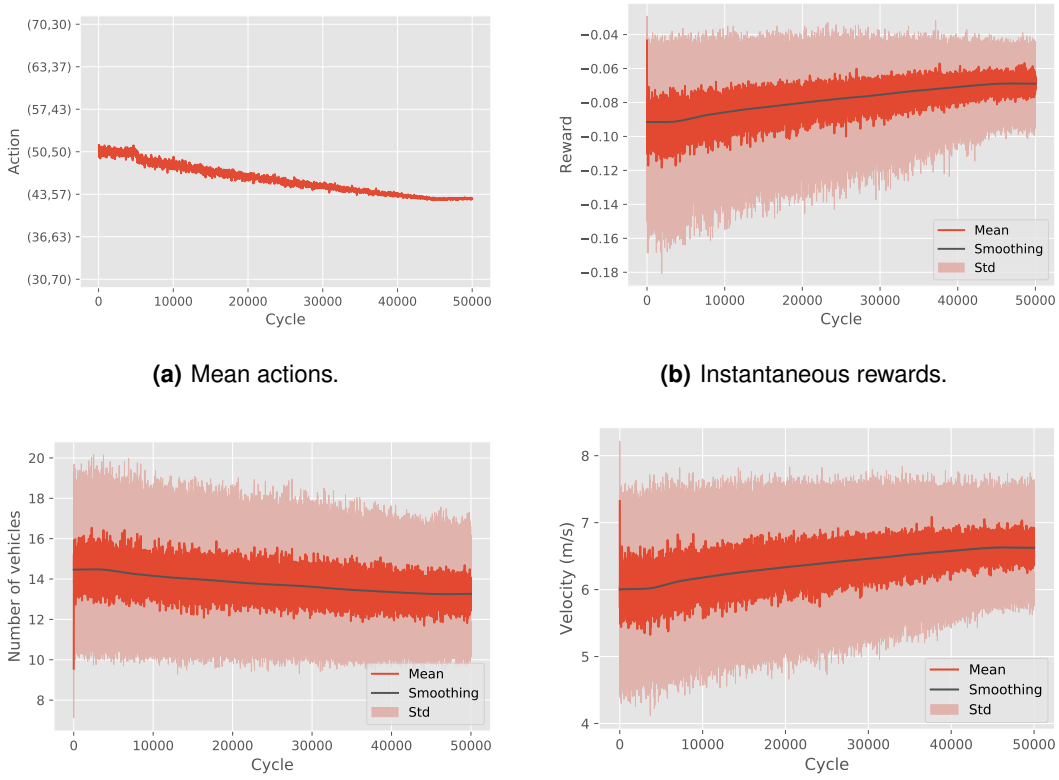
In our running example, we complete the RL controller setup by implementing the well known DQN algorithm. The neural network used to approximate the state-action values consists of a three-layer feed-forward ANN, with an [8,16,8] architecture, and ReLU non-linearities.

4.3 Training

Reinforcement learning controllers might overfit to the training experience, showing good performance during training but performing poorly at deployment time [89]. For example, even in the same domain, different simulation environments may generate inconsistent results, and the hyperparameters setup will depend on the specific environment [90]. Additionally, simply changing the random seeds used to generate the simulations may influence, in a statistically significant manner, the outcome of the RL algorithm [91]. As such, much as in supervised learning, results in RL should be reported under a variety of conditions, ensuring that the reported performance holds for multiple training seeds and test conditions [92]. Therefore, in the context of TLC, it is important to run multiple instances of the training process, using different seeds, in order to correctly assess the learning ability of the proposed RL methods. During the training procedure, particular attention must be paid to ensure that the simulations are properly running. Gridlocks should be avoided or properly processed, for example by restarting the simulator, adjusting the vehicles' arrivals, or teleporting vehicles.

The DQN-based agent from our running example is trained using sampled trajectories. In order to properly balance between exploration and exploitation, we use an ϵ-greedy policy, with ϵ being linearly decayed from 1.0 to 0.01 throughout the train duration. Thirty training runs are performed. Figure 4.3(a) displays the mean actions taken during training. As it can be seen, the mean actions converge towards lower-indexed signal plans, which can be justified by the intersection's structure (Figure 4.2(b)): the horizontal direction (phase 2) serves more vehicles in comparison to the vertical direction (phase 1), as it has a higher number of lanes and the insertion probabilities are roughly constant for each lane. Furthermore, the observed instantaneous reward increases as the training proceeds (Figure 4.3(b)), as well as the throughput of the network (Figures 4.3(c) and 4.3(d)).

The outcome of the training process consists of 30 policies, each one resulting from a different training run, that need now to be properly evaluated. The next and final step of the proposed methodology addresses how this can be accomplished.



(a) Mean actions.

(b) Instantaneous rewards.

(c) Number of vehicles.

(d) Vehicles' velocity.

Figure 4.3: Observed curves during the learning procedure (averaged over 30 training runs).

4.4 Evaluation

In the context of traffic light control, as discussed in Section 2.3.2, different performance metrics have been proposed. From all these metrics, the minimization of the average travel time is usually the main goal in the development of traffic signal controllers, and arguably the most important metric to report. Since most of the times algorithms are unable to directly optimize the travel time, it is useful to report additional metrics that are more closely related with the formulated agent's objective, such as the queue length or waiting time. It is also important to run some baseline algorithms, such as those presented in Section 3.1, under the same scenario. These runs are of extreme importance, since they allow to compare the performance of the RL controller(s) against well-established, commonly used traffic engineering methods.

4.4.1 Performance estimation

The performance of each TSC should be assessed using a set of accordingly seeded simulations in order to rule out any influence of the simulation seeds in the results, as previously discussed.

Table 4.2: Observed performance metrics for different controllers, per vehicle's trip, under the *intersection* network. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values, whereas non-tuple entries display the mean value. The static baseline corresponds to the fixed signal plan that yielded the lowest average travel time, optimized via grid-search. The actuated controller dynamically extends the current phase, up to a maximum value, if a continuous stream of incoming vehicles is detected. For all baselines, a set of vehicles' trips is gathered from 30 simulations of 24 hours, from which the mean and standard deviation values for each of the metrics are calculated. With respect to the RL controller, each of the 30 policies is evaluated by running 3 evaluation rollouts of 24 hours. Afterwards, the trips of all evaluation rollouts are concatenated, and the mean and standard deviation values calculated for each of the metrics.

Method	Number of stops	Waiting time (s)	Travel time (s)
Static	0.47	(8.3, 10.2)	(25.0, 12.5)
Webster	0.49	(8.5, 10.0)	(25.4, 12.4)
Max-pressure	0.48	(6.0, 6.9)	(23.4, 9.0)
Actuated	0.47	(8.0, 10.5)	(24.9, 12.8)
RL controller	0.47	(8.3, 10.3)	(25.0, 12.6)

With respect to non-trainable algorithms such as the methods from transportation engineering, the performance estimation is simpler since the main source of variability affecting the results arises from the evaluation simulations themselves.

On the other hand, for trainable algorithms such as the RL controller(s), two main sources of variability affect the reported results. In the first place, the training step produces a set of policies, which can have different performances. Secondly, when evaluating a policy that resulted from a given training run, the evaluation procedure itself induces another level of uncertainty. Therefore, we propose to evaluate each policy with a set of evaluation rollouts and, if needed, aggregate the results per policy a posteriori.

4.4.2 Performance analysis & comparison

The simplest and most straightforward way to present and compare the performances of the different methods is through point estimation, commonly used in the TLC domain. Table 4.2 displays the mean and standard deviation point estimations for a set of performance metrics and TSCs, including the previously developed RL controller. An in-depth discussion of these results is postponed to Chapter 6, however, we can note right away that the results exhibit high standard deviation values. In fact, most of the times, there is an overlap in performance for different TSCs. Therefore, it might be important to understand whether there is enough evidence to say that the observed difference in performance is statistically significant or not.

A better insight into the performance of the methods can be achieved by plotting the distributions of the travel time means for each of the TSCs, as seen in Figure 4.4. The same can be done for all the other metrics. It is noticeable, again, an overlap in performance between the different distributions. In order to understand whether the observed mean differences are statistically significant or not, statistical hypothesis testing should be used.

In the context of this work, we are mostly interested in understanding whether two or more popula-

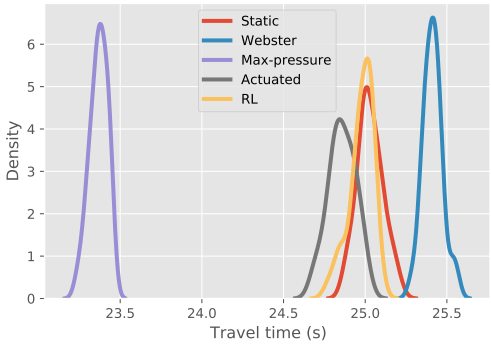


Figure 4.4: Kernel density estimation of the travel time means, computed using 30 samples for each method. For all baselines, each sample corresponds to the mean travel time of each simulation. With respect to the RL controller, each sample corresponds to the mean travel time for each of the policies (calculated using 3 evaluation rollouts).

tion means (the performance metrics of the different TSCs) are equal. It is highly likely that the previous hypothesis is rejected. As such, post hoc comparisons need to be performed in order to understand between which mean pairs exists a statistically significant difference. In order to perform such evaluation, we can use a one-way Analysis of Variance (ANOVA) test, followed by the Tukey's range test for post hoc comparisons. Unfortunately, the previous tests make some assumptions related with the data distributions that not always hold, especially regarding the observed performance in the case of RL agents. Namely, they assume independence between the groups, that the data is normally distributed, and that all groups share the same variance. Therefore, it is worth checking whether the assumptions are met, especially the last two. This can be done, for example, with the Shapiro–Wilk test [93], and the Levene's test [94]. If the assumptions are violated, it is recommended to switch the aforementioned tests with their respective non-parametric versions.

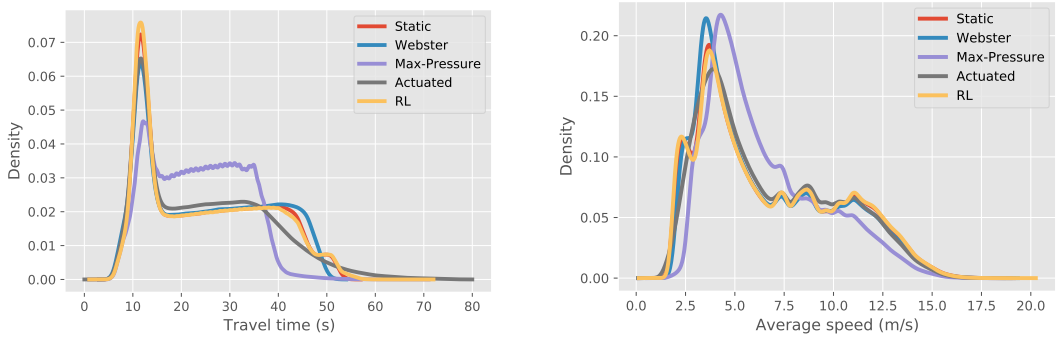
Regarding our running example, the ANOVA test yields a p-value $< 10^{-5}$, thus the null hypothesis (that all TSCs have the same mean performance) is rejected. Table 4.3 displays the results of the post hoc analysis using the Tukey's range test. As it can be seen, between the actuated, static and RL controllers, the confidence interval on the means' difference either includes zero, or its bounds are close to it, hinting that there might be no significant difference in performance between the three methods. Once again, a more detailed discussion of the results is postponed to Chapter 6.

As a complementary analysis, we plot in Figure 4.5 the kernel density estimation of the average travel time and average speed distributions. As can be seen, for both metrics, the underlying distributions are highly skewed.

The previously presented methods provide a robust way to assess and compare the performance of different controllers for TLC. However, regarding the RL controller, they do not explain "how" the method is able to attain such performance, and what are the key behavioural features that underlie the observed performance, something of great interest and importance in the TLC domain. In the continuation, we

Table 4.3: Post hoc Tukey's range test results. The mean diff column displays the difference between the means of group 2 and group 1. The following two columns display the bounds of the confidence interval for the difference in means.

Group 1	Group 2	Mean diff	CI lower bound	CI upper bound
Actuated	Max-Pressure	-1.49	-1.54	-1.44
Actuated	RL	0.12	0.07	0.17
Actuated	Static	0.17	0.12	0.22
Actuated	Webster	0.56	0.51	0.60
Max-Pressure	RL	1.61	1.56	1.66
Max-Pressure	Static	1.66	1.61	1.71
Max-Pressure	Webster	2.04	2.00	2.09
RL	Static	0.05	-0.01	0.10
RL	Webster	0.44	0.39	0.48
Static	Webster	0.38	0.33	0.43



(a) Travel time metric.

(b) Speed metric.

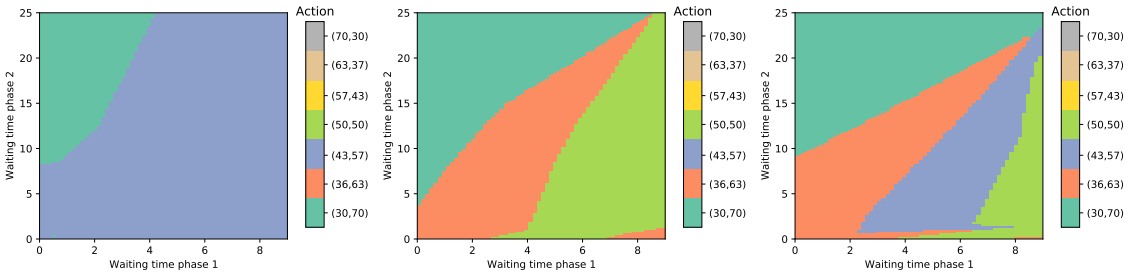
Figure 4.5: Kernel density estimation of the travel time and speed metrics, aggregated per vehicle trip.

provide an analysis of the policies obtained by the RL algorithms in our running example.

4.4.3 Policy analysis

Being able to explain the decisions taken by the agent, as well as predict its behaviour under different situations is fundamental when developing a RL system in the context of TLC, a domain where the control over real-world TSCs must only be conceded to agents that exhibit a responsible and trustable behaviour. While a comprehensive study of this important topic is outside the scope of the present work, some of the efforts taken in order to better understand the behaviour of the agents are presented below.

Due to the low dimensional state space definition, a simple yet effective way to understand the behaviour of the agent is to directly plot its policy. Figure 4.6 displays three different policies that resulted from the same training procedure, only differing on the simulation seeds. As it can be seen, despite the fact that all three policies exhibit reasonable performance, they choose slightly different actions for a comprehensive part of the state space. Nevertheless, all policies privilege phase 2, a sensible choice given the intersection's topology. As it can be seen, the qualitative analysis of the resulting policies unveils additional findings with respect to the agent's behaviour.



(a) Policy 1 (Travel time: 25.04). **(b)** Policy 2 (Travel time: 24.88). **(c)** Policy 3 (Travel time: 24.82).

Figure 4.6: Illustration of three policies that resulted from the training procedure. The plot displays, color-encoded, the action that yields the maximum Q-value for each possible point in the state space domain.

The present chapter introduced a methodology for the development of intelligent TSCs that provides a standardization of all the steps required to train and evaluate RL-based TSCs, allowing a fair and meaningful comparison between different approaches to TLC. The following chapter partly concretizes the presented methodology by extracting a set of road networks and defining a set of demand types, as well as implementing a set of MDP formulations and RL controllers.

5

Implementation

Contents

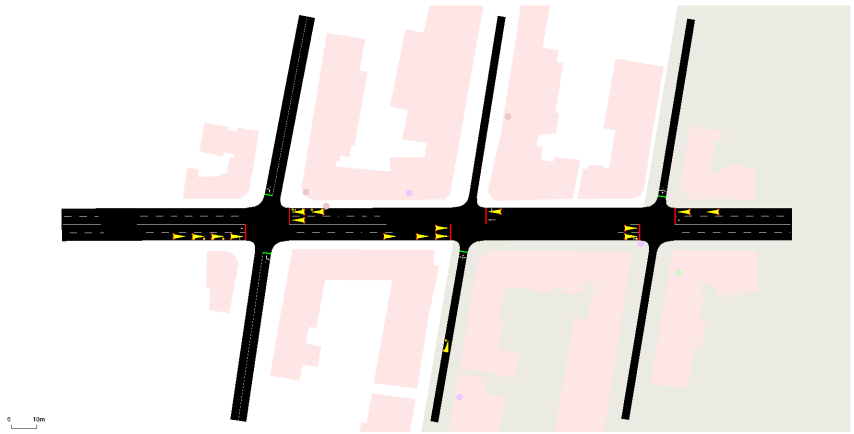


Figure 5.1: SUMO screenshot of the *arterial* network, composed of four streets. On the horizontal direction (major street): Rua do Conde de Redondo. On the vertical direction (from the left to the right): Rua Luciano Cordeiro, Rua Bernardo Lima and Rua Ferreira Lapa. The network is composed of three two-phased intersections. Phase 1 allows the movement of vehicles in the vertical direction, whereas phase 2 allows the movement in the horizontal direction.

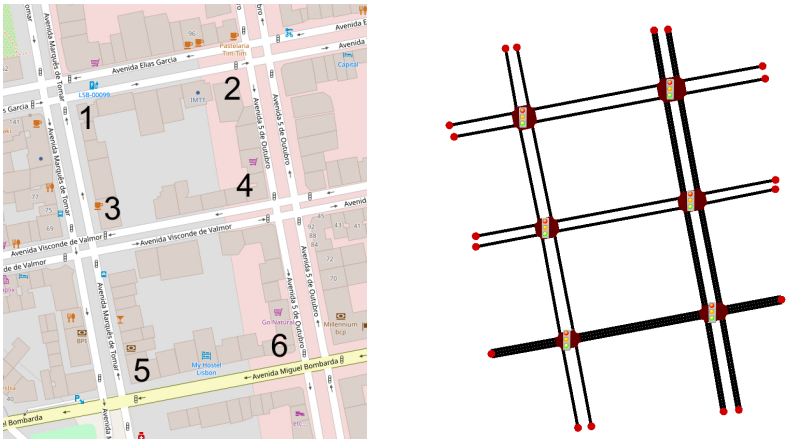
This chapter details the different implemented components that allow to compare different RL-based approaches to TLC, namely the extracted network topologies, the types of demands considered, and the set of MDP formulations and RL algorithms used. The extraction/implementation of the different components closely followed the methodology described in the previous chapter.

5.1 Simulation setup

All considered road network topologies are extracted from the city of Lisbon following the steps described in the previous chapter. Three networks, each representative of a different topology type are considered: an *isolated* intersection, an *arterial* network, and a *grid* network. The first two networks are extracted from an area near Marquês de Pombal square. The simplest network (*intersection* network) was introduced in Chapter 4 (Fig. 4.2) and comprises a single isolated two-phased intersection. Figure 5.1 displays the second considered network (*arterial* network) from the same downtown area, comprising three two-phased intersections. The third network (*grid* network) is extracted from an area between Saldanha square and Campo Pequeno square. It exhibits a modern, grid-like topology commonly observed in cities throughout the world. Figure 5.2 displays two screenshot of the network from the aforementioned area. Figure 5.3 displays a set of screenshots of the intersections that compose the *grid* traffic network.

With respect to the demand types, the simplest considered definition, named *constant* demand, comprises a time-constant demand, i.e., the insertion probabilities mapping is kept constant throughout the simulation. This type of demand was introduced in Section 4.1.3.

A more interesting type of demand, named *variable* demand, incorporates a variation of the insertion



(a) OSM screenshot. **(b)** SUMO screenshot.

Figure 5.2: *Grid* network, extracted from Lisbon's metropolitan area near Saldanha square and Campo Pequeno square. Noteworthy streets include Av. 5 de Outubro and Av. Miguel Bombarda. For all intersections, phase 1 allows the movement of vehicles in the vertical direction, whereas phase 2 allows the movement in the horizontal direction.

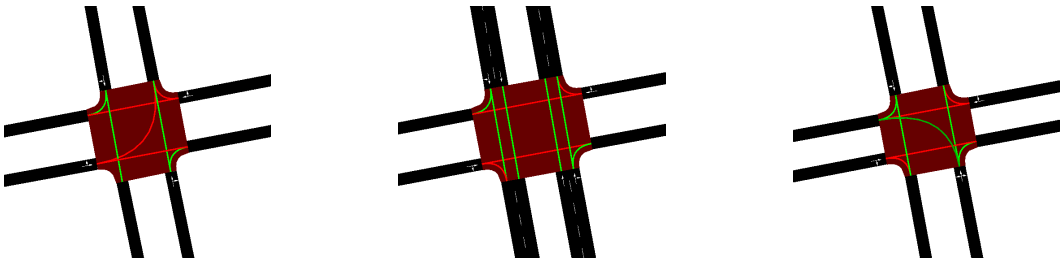
probabilities throughout the day, resembling real-world traffic patterns such as rush-hours and free-flow periods. Depending on the hour of the day, the constant insertion probabilities mapping is, thus, further rescaled by a scalar bounded between zero and one. Figure 5.4 displays the implemented rescaling function.

The third considered demand, named *cyclical* demand, rescales the constant insertion probabilities at unequal rates for two disjoint sets of source edges defined by the user. This allows to create rich simulation scenarios where the main direction of traffic changes throughout the day. Formally, let $\mathcal{E}$ be the set of edges at the boundaries of the network which act as sources. Let $\mathcal{U}$ and $\mathcal{V}$ be a disjoint partition of $\mathcal{E}$. Then, the insertion probabilities associated to the edges in $\mathcal{U}$, $U(t)$, and the insertion probabilities associated to the edges in $\mathcal{V}$, $V(t)$, can be defined as

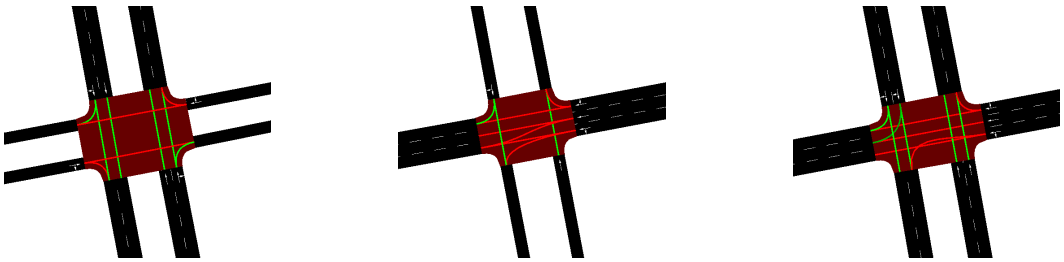
$$U(t) = A(t)\left[1 + 0.5cos\left(\frac{2\pi}{T_u}t + \phi_u\right)\right], \tag{5.1}$$

$$V(t) = A(t)\left[1 + 0.5sin\left(\frac{2\pi}{T_v}t + \phi_v\right)\right], \tag{5.2}$$

where T_u is the period parameter, in seconds, associated with the set of edges $\mathcal{U}$, T_v is the period parameter associated with the set of edges $\mathcal{V}$, ϕ_u and ϕ_v are phase parameters, and $A(t)$ represents the base demand factor given by a constant demand mapping. Figure 5.5 displays the function parameterization used for the *cyclical* demand.



(a) Intersection 1. **(b)** Intersection 2. **(c)** Intersection 3.



(d) Intersection 4. **(e)** Intersection 5. **(f)** Intersection 6.

Figure 5.3: *Grid* network intersections (close-up screenshots). The intersections' numbering is displayed in Figure 5.2(a).

5.2 Markov decision process formulations

We consider a fully decentralised approach to TLC. Each agent is responsible for the control of the traffic lights of a given intersection: it perceives the traffic state at the intersection level and, through the choice of its actions, aims to optimize a local reward function. The action space definition is fixed to the *set phase split*. For algorithms with a discrete action space support, the agents are able to pick, at the end of each cycle, the signal plan to be executed throughout the next cycle. Seven signals plans, i.e., seven actions, are considered (Table 4.1). With respect to RL algorithms with continuous action space

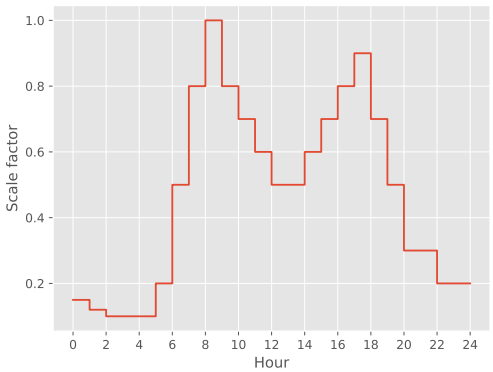


Figure 5.4: Rescaling factor used in the *variable* demand type.

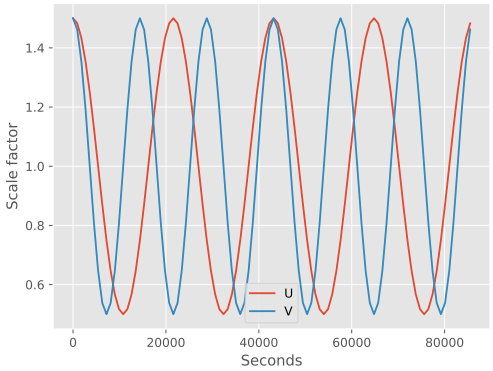


Figure 5.5: Rescaling factor used in the *cyclical* demand with parameters $T_u = 21600$, $T_v = 14400$, $\phi_u = 0$, and $\phi_v = 0$. The effect of $A(t)$ is neglected. As it can be seen, the different periods of the trigonometric functions contribute to create rich simulations with periods of positive-positive, negative-negative, positive-negative and negative-positive contributions. For the implementation, the function is further discretized into intervals of fifteen minutes.

support, such as the DDPG algorithm, the agent is able to directly pick the percentage of the cycle length allocated for each of the phases, however, a minimum green time is ensured for all phases.

We implement different MDP formulations so their performance can be fairly assessed and compared under the different simulation scenarios presented in the previous section. The design of the different formulations is greatly inspired by the works described in Section 3.3. However, the formulations are usually adapted so that they can fit our action space definition. Below, we describe the different MDP formulations used in our work, specifically the state space and reward function definitions.[1]

The cycle length is fixed to 60 seconds, optimized via grid-search, and the yellow time to six seconds. The discount factor is set to 0.98.

5.2.1 Preliminaries

In the remainder of this chapter, K represents the cycle length, L_{p_i} denotes the set of lanes associated with phase p_i, V_l^k denotes the set of vehicles traveling on lane l at the timestep k, and $\text{speed}(v, k)$ gives the normalized velocity of vehicle v at timestep k. The number of vehicles associated with phase p_i, at timestep k, can be calculated as follows

$$N_{p_i}^k = \sum_{l \in L_{p_i}} |V_l^k|. \tag{5.3}$$

[1] In order to keep the presentation concise, the described formulations consider two-phased intersections. However, they can be easily adapted to accommodate an arbitrary number of phases.

Thus, the average volume $v_c^{p_i}$, associated with phase p_i during a certain cycle c, is given by

$$v_c^{p_i} = \frac{1}{K} \sum_{k=0}^{K-1} N_{p_i}^k. \tag{5.4}$$

5.2.2 Speed-based formulations

A speed-based formulation is focused on the optimization of the vehicles' average velocity. It can be either aiming at the direct maximization of vehicles' velocity, or at the minimization of some sort of distance between the maximum allowed speed and the actual velocity of the vehicles. Intuitively, the higher the average speed of the vehicles, the lower the average travel time. Our speed-based formulations, below presented, are inspired by the works of Casas [55] and Wei et al. [56].

The *minimize speed delta* definition seeks to minimize a weighted average of the distances between the maximum speed of the road and the actual velocity of the vehicles. The number of vehicles travelling at each timestep is used as a weight factor. Thus, the agent is penalized if a high number of vehicles is moving at low speeds. For a given intersection, at a certain cycle c, the state consists of a tuple $(v_c^{p_1}, v_c^{p_2}, s_c^{p_1}, s_c^{p_2})$ containing the observed average volume and speed for each of the phases, p_1 and p_2, where

$$s_c^{p_i} = \frac{1}{\sum_{k=0}^{K-1} N_{p_i}^k} \sum_{k=0}^{K-1} \sum_{l \in L_{p_i}} \sum_{v \in V_l^k} \Big(1 - \text{speed}(v,k)\Big), \tag{5.5}$$

and the reward is defined as

$$r_c = -\sum_{i=1}^{2} v_c^{p_i} s_c^{p_i}. \tag{5.6}$$

The *minimize delay* formulation considers a cost function that penalizes vehicles that are moving at slow speeds. For a given intersection, at a certain cycle c, the state consists of a tuple $(d_c^{p_1}, d_c^{p_2})$, where

$$d_c^{p_i} = \frac{1}{K} \sum_{k=0}^{K-1} \sum_{l \in L_{p_i}} \sum_{v \in V_l^k} \text{delay}(v,k), \tag{5.7}$$

$$\text{delay}(v,k) = \exp\{-5 \cdot \text{speed}(v,k)\}, \tag{5.8}$$

and the reward is defined as

$$r_c = -\sum_{i=1}^{2} d_c^{p_i}. \tag{5.9}$$

Finally, the *maximize delay reduction* aims to maximize the delay reduction between two consecutive cycles. The state space of the *minimize delay* definition is extended to accommodate the delay of the vehicles for the present cycle, as well as the delay of the vehicles for the previous cycle, consisting of a tuple $(d_c^{p_1}, d_c^{p_2}, d_{c-1}^{p_1}, d_{c-1}^{p_2})$. The reward is thus defined, at cycle c, as

$$r_c = \sum_{i=1}^{2} \left(d_{c-1}^{p_i} - d_c^{p_i} \right). \tag{5.10}$$

5.2.3 Waiting time-based formulations

A waiting time-based formulation is focused on the minimization of the time vehicles spend in queue, i.e., stopped or travelling at low speeds. The *minimize waiting time* definition, similar to Nishi et al. [59], was already considered in the running example from Chapter 4. The state consists of a tuple $(w_c^{p_1}, w_c^{p_2})$ containing the observed waiting times for each of the phases, p_1 and p_2, at a certain cycle c, where

$$w_c^{p_i} = \frac{1}{K} \sum_{k=0}^{K-1} \sum_{l \in L_{p_i}} \sum_{v \in V_l^k} \text{stopped}(v, k), \tag{5.11}$$

$$\text{stopped}(v, k) = \begin{cases} 1 & \text{speed}(v, k) < 0.1 \\ 0 & \text{otherwise,} \end{cases} \tag{5.12}$$

and the reward is defined as

$$r_c = -\sum_{i=1}^{2} w_c^{p_i}. \tag{5.13}$$

5.2.4 Queue-based formulations

The *minimize queue* definition, similarly to Abdoos et al. [95], is focused on the minimization of the average queue length for all incoming approaches. For a given intersection, at a certain cycle c, the state consists of a tuple $(q_c^{p_1}, q_c^{p_2})$, where

$$q_c^{p_i} = \max_{k \in K} \left\{ \max_{l \in L_{p_i}} \left(\sum_{v \in V_l^k} \text{stopped}(v, k) \right) \right\}, \tag{5.14}$$

and $\text{stopped}(v, k)$ indicates whether vehicle v is stopped at timestep k (defined in Eq. 5.12). The reward is defined as

$$r_c = -\sum_{i=1}^{2} q_c^{p_i}. \tag{5.15}$$

The *maximize queue reduction* definition, similarly to El-Tantawy et al. [11], aims to maximize the reduction in queue length between consecutive cycles, for both phases. The state space of the *minimize queue* definition is extended to accommodate the queue-related information for the present cycle, as well as the queue-related information for the previous cycle, consisting of a tuple $(q_c^{p_1}, q_c^{p_2}, q_{c-1}^{p_1}, q_{c-1}^{p_2})$. The reward is defined as

$$r_c = \sum_{i=1}^{2} \left(q_{c-1}^{p_i}\right)^2 - \sum_{i=1}^{2} \left(q_c^{p_i}\right)^2. \tag{5.16}$$

5.2.5 Pressure-based formulations

The *minimize pressure* formulation, similarly to Wei et al. [61], seeks to minimize the intersection's pressure (the pressure concept was introduced in Section 3.1). For a given intersection, at a certain cycle c, the state consists of a tuple $(p_c^{p_1}, p_c^{p_2})$, where

$$p_c^{p_i} = \frac{1}{K} \sum_{k=0}^{K-1} \Big(\sum_{l \in L_{p_i}} |V_l^k| - \sum_{o \in O_{p_i}} |V_o^k| \Big), \tag{5.17}$$

and O_{p_i} denotes the set of all outgoing lanes of phase p_i. The reward is defined as

$$r_c = -\sum_{i=1}^{2} p_c^{p_i}. \tag{5.18}$$

5.3 Reinforcement learning methods

We consider three different classes of RL algorithms: Q-learning, a tabular method with discrete action support, deep Q-network, a function approximation method with discrete action support, and DDPG, a deterministic actor-critic method with continuous action support. All considered algorithms are used in an off-policy setting, a choice majorly justified by the need of improved efficiency due to the low frequency of interaction with the environment. Below, we provide an overview of the main implementation choices for the different algorithms, and refer to Appendix A for a complete list of hyperparameters used.

5.3.1 Q-learning

The Q-learning algorithm holds a table containing the estimated Q-value for each state-action pair. The procedure used to discretize each of the state space features into a set of six categories/bins consists of the following steps: (a) a set of trajectories is gathered by performing some simulations with Webster TSCs; (b) the collected data is grouped by intersection and phase; (c) for each intersection and phase, a distribution of the observed values for the given state feature is constructed; and (d) the values of the

bins used for discretization equal the [0.1,0.3,0.5,0.7,0.9] percentiles of the distributions. We empirically observed that it is of extreme importance for the bins to be calculated independently for each feature, intersection, and phase, otherwise it is likely that the bins will be misaligned, impacting performance.

In order to improve sample-efficiency, we store the observed trajectories in a replay buffer and, at each iteration, perform multiple Q-learning updates using transitions sampled from the replay memory. Finally, we use an ϵ-greedy policy in order to ensure an adequate exploration-exploitation balance, with $\epsilon_t = 1/(1 + N_t(s))$, where $N_t(s)$ denotes the number of times state s was visited up to timestep t.

5.3.2 Deep Q-network

The considered DQN algorithm consists of a fully connected neural network with three hidden layers, that receives a state as input and outputs the Q-values for each corresponding action. Further improvements over the standard DQN algorithm are considered, including double Q-learning, a dueling network architecture, prioritized replay, and N-step transitions. In order to properly balance exploration and exploitation, we use an ϵ-greedy policy, with ϵ being linearly decayed from 1.0 to 0.01 throughout the train duration.

5.3.3 Deep deterministic policy gradient

The DDPG algorithm is able to directly pick the portion of the cycle length allocated for each of the phases. More precisely, the agent picks the allocation of 80% of the total green time of the cycle, while the remainder 20% are equally split among all phases. By doing this, we ensure that all phases receive a minimum green time.

The critic consists of a fully connected neural network (composed of three hidden layers), where the final layer maps the hidden activations into a scalar representing the Q-value for the input state-action pair. The actor consists of a fully connected neural network, composed of three hidden layers, where the final layer is a softmax function mapping the hidden activations into the allocations for each of the phases. In order to ensure adequate exploration, we introduce gaussian noise $\mathcal{N} \sim N(0, \sigma^2)$ to the behavior network, with σ^2 being linearly decayed throughout the train duration. Figure 5.6 displays an illustration of the actor network architecture.

Finally, as pointed out by previous works [8], the DDPG algorithm is sensible to different scales in the state input features. Therefore, all state variables are standardized before being inputted into the neural networks. This revealed to be crucial to stabilize the learning procedure.

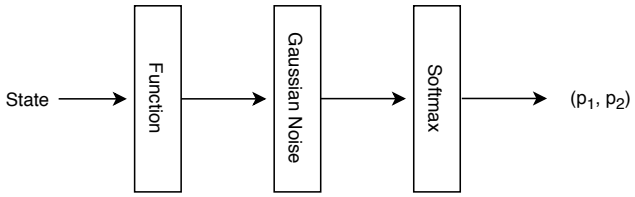


Figure 5.6: DDPG actor network illustration. The output contains the allocation for each of the phases. The output layer size can be adjusted to an arbitrary number of phases.

5.4 Baseline controllers

We use as baselines the Webster and max-pressure methods introduced in Section 3.1, as well as an actuated controller[2] that dynamically extends the current phase, up to a maximum value, if a continuous stream of incoming vehicles is detected. We implement an additional controller, named "adaptive-Webster". It features a fixed cycle length, but dynamically allocates, every five minutes, the phase splits proportionally to the counts of vehicles that arrived at each of the intersection's phases. Therefore, the adaptive-Webster controller is capable of adapting to time-changing demands as the timings are dynamically adjusted throughout the simulation. The Webster, adaptive-Webster, and RL controllers are *periodic*, since they feature a fixed cycle length; the two remainder controllers, max-pressure and actuated, are *non-periodic*, because they allow for variable cycle lengths.

5.5 Software implementation

The software was developed in a Python environment, using two additional open-source frameworks: FLOW [85], which is a computational framework for the development of RL-based TLC systems, and the RL framework ACME [96, 97]. The traffic simulation backend is held by the SUMO simulator [80], and the computational graphs' computations by the Tensorflow framework [98].

[2] https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html#based_on_time_gaps

6

Experimental results

Contents

This chapter presents and discusses the experimental results obtained in different scenarios, starting with the simpler *intersection* all the way to the *grid* network. Our analysis follows the methodology discussed in Chapter 4. However, we present and discuss only a representative subset of all obtained results. We refer to Appendix B for a complete listing of all gathered experimental data, comprising the report of all selected performance metrics, as well as detailed demands' definition.

6.1 *Intersection* network

The first studied traffic network is the *intersection* network, a single and isolated intersection. We assess the performance of the different controllers under three different types of demands: a time-*constant* demand, a time-*variable* demand, and a phase-uneven (*cyclical*) demand. Results for the RL controllers are obtained from 30 independent training runs, each of which evaluated using three rollouts of 24 hours. Non-trainable methods are evaluated with a set of 30 rollouts of 24 hours.

6.1.1 *Constant* demands

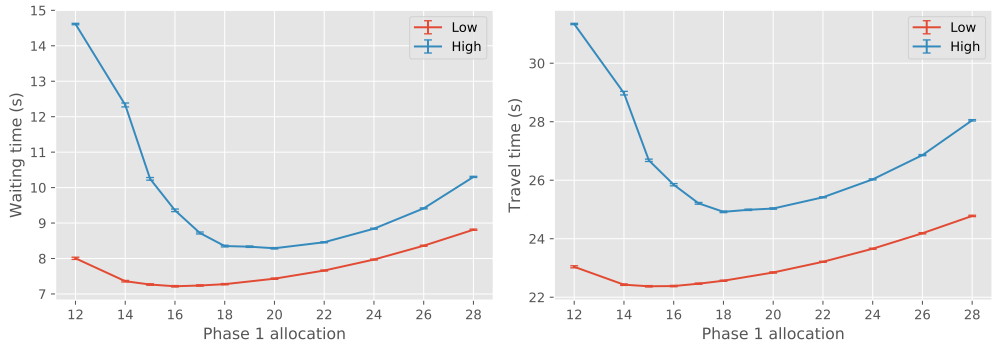
The simplest experimental setup comprises the study of two time-*constant* demands:

- *High constant* demand: the intersection is congested and, during some cycles, the maximum queue length is achieved.
- *Low constant* demand: vehicles freely flow through the intersection and queues barely emerge.

We carried a grid-search to determine the best fixed phase allocations for the present experimental setup. Figure 6.1 displays the obtained performance metrics for the two demand types. As it can be seen, for the *high constant* demand, higher average waiting and travel times are observed in comparison to the *low constant* demand, for every signal plan. Furthermore, it is noticeable that the difference in performance between the static signal plans is higher for the *high* demand in comparison to the *low* demand, i.e., changes in the phase allocations impact more performance for the *high* demand in comparison to the *low* demand.

With respect to the average travel time (Fig. 6.1(b)), the optimal static signal plan for the *high* demand allocates around 18 seconds for phase 1, whereas for the *low* demand the optimal value decreases to around 15 seconds.

Regarding the baseline controllers, for both the *high* (Tab. 6.1) and *low* (Tab. 6.2) *constant* demands, the static and Webster controllers are outperformed by the max-pressure controller. Despite the fact that the actuated controller achieved a similar average travel time in comparison to the static controller for the *high constant* demand, it was capable of outperforming the periodic baselines with respect to the *low*



(a) Waiting time metric.

(b) Travel time metric.

Figure 6.1: (*Intersection* network) Performance metrics for static signal plans with different phase 1 allocations. Displayed values are calculated by averaging over 30 simulations of 24 hours. Error bars represent the 95% bootstrapped mean confidence interval.

Table 6.1: (*Intersection* network, *high constant* demand) Baselines performance metrics. The static controller corresponds to the static signal plan that achieved the lowest average travel time, optimized via grid-search.

Controller	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Static	8.3	25.0	24.9
Webster	8.5	25.4	25.3
Max-pressure	6.0	23.4	23.3
Actuated	8.0	24.9	24.7

constant demand. For the two demand types, it is important to notice that the non-periodic controllers achieve a lower average waiting time in comparison to the periodic controllers.

Since non-periodic controllers do not have a fixed cycle length, they are able to cycle through the phases faster, thus minimizing the time each vehicle needs to wait before crossing the intersection. Given the fact that the present network topology consists of an isolated intersection, this explains why non-periodic controllers usually achieve improved average travel times: the delay vehicles experience in their trip is caused by the time they wait to cross the intersection, thus, a method that is successful in reducing the waiting time of the vehicles will directly contribute to the reduction of the travel time (a strong correlation is visible between the two plots displayed in Fig. 6.1). However, the performance gain of the non-periodic baselines over the periodic baselines is higher for the *low* demand type (Tab. 6.2) in comparison to the *high* demand type (Tab. 6.1). This shows that for congested/saturated conditions, no big improvement over static timings is usually achieved with the use of adaptive, more reactive TSC systems, whereas for the less saturated regime, more adaptable systems, namely non-periodic controllers, exhibit a higher performance gain.

Regarding the Q-learning controller, under the *high constant* demand (Tab. 6.3), obtained results for all MDPs exhibit performances significantly better than the random policy, for all reported metrics; as an example, the average travel time is at least ≈2.6 seconds lower. However, the *min. speed delta*, *max. delay reduction*, and *max. queue reduction* MDPs achieved worse results in comparison to the

Table 6.2: (*Intersection* network, *low constant* demand) Baselines performance metrics. The static controller corresponds to the static signal plan that yielded the lowest average travel time, optimized via grid-search.

Controller	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Static	7.2	22.4	22.2
Webster	7.7	23.2	23.1
Max-pressure	4.7	20.2	20.1
Actuated	5.1	20.8	20.7

Table 6.3: (*Intersection* network, *high constant* demand) RL controllers performance metrics. Columns denoted with WT, TT, and TT-3 correspond, respectively, to the average waiting time (s), the average travel time (s), and the average travel time (s) of the top-3 policies.

	Q-learning			DQN			DDPG		
	WT	TT	TT-3	WT	TT	TT-3	WT	TT	TT-3
Random	11.4	29.1	28.6	11.4	29.1	28.6	13.3	31.3	30.6
Min. speed delta	9.2	26.3	26.0	8.5	25.1	24.9	8.6	25.4	24.9
Min. delay	8.4	25.0	24.9	8.4	25.1	24.8	8.3	25.1	24.8
Max. delay reduction	9.4	26.5	26.3	9.4	26.1	25.0	8.4	25.3	24.9
Min. waiting time	8.4	25.0	24.9	8.3	25.0	24.9	8.3	25.2	24.9
Min. queue	8.7	25.4	25.0	8.3	25.0	24.8	8.5	25.2	24.9
Max. queue reduction	9.4	26.3	26.0	8.5	25.3	24.9	8.7	25.4	25.1
Min. pressure	8.4	25.1	24.9	8.6	25.2	24.8	8.3	25.1	24.8

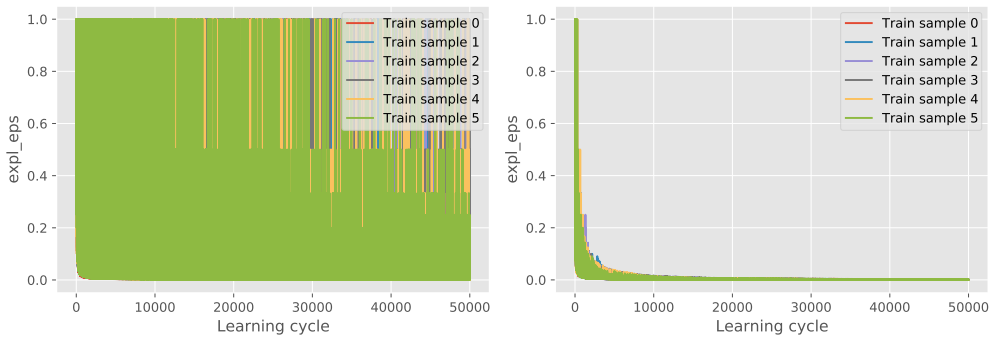
remainder formulations, yielding an average travel time approximately one second higher in comparison to the other MDPs. A similar trend is observed when comparing the performances of the three policies that yielded the lowest average travel times for each of the MDPs.

With respect to the Q-learning controller, under the *low constant* demand (Tab. 6.4), the *min. delay*, *min. waiting time*, and *min. pressure* MDPs achieve improved performance over the remainder formulations. On the other hand, the *min. speed delta*, *max. delay red.*, *min. queue* and *max. queue red.* MDPs recorded the highest average waiting and travel times, sometimes only marginally outperforming the random policy.

The fact that some MDPs, namely the *min. speed delta*, *max. delay red.*, and *max. queue red.* formulations, are outperformed by the remainder formulations with respect to the Q-learning algorithm can be explained due to underlying convergence problems. Despite the fact that additional efforts were

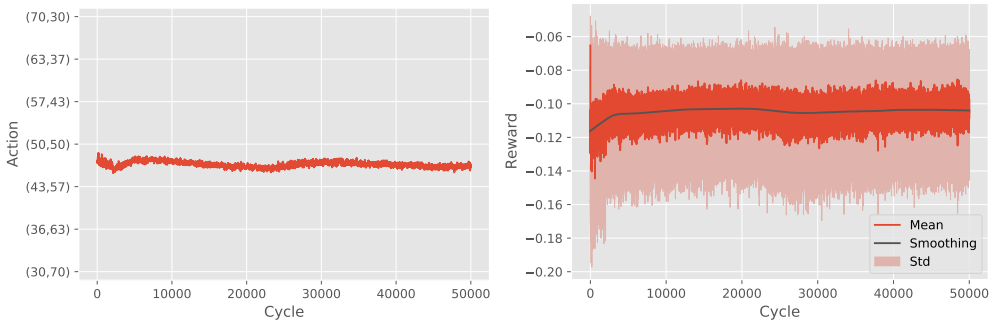
Table 6.4: (*Intersection* network, *low constant* demand) RL controllers performance metrics. Columns denoted with WT, TT, and TT-3 correspond, respectively, to the average waiting time (s), the average travel time (s), and the average travel time (s) of the top-3 policies.

	Q-learning			DQN			DDPG		
	WT	TT	TT-3	WT	TT	TT-3	WT	TT	TT-3
Random	8.6	24.4	24.2	8.6	24.4	24.2	9.4	25.2	24.9
Min. speed delta	8.1	23.7	23.5	7.2	22.4	22.3	7.3	22.4	22.3
Min. delay	7.4	22.6	22.4	7.2	22.4	22.3	7.3	22.6	22.4
Max. delay reduction	8.1	23.7	23.6	8.2	23.8	22.4	7.3	22.4	22.3
Min. waiting time	7.4	22.6	22.4	7.2	22.4	22.3	7.4	22.6	22.3
Min. queue	8.7	24.4	23.1	8.7	24.5	22.5	8.5	24.3	22.4
Max. queue reduction	8.3	24.0	23.7	8.7	24.4	22.4	9.3	25.3	23.2
Min. pressure	7.4	22.6	22.4	7.2	22.4	22.3	7.3	22.6	22.3



(a) *Minimize speed delta* MDP. **(b)** *Minimize delay* MDP.

Figure 6.2: (*Intersection* network, *high constant* demand) Q-learning ϵ exploration parameter throughout training. The curves are plotted for five randomly sampled train runs.

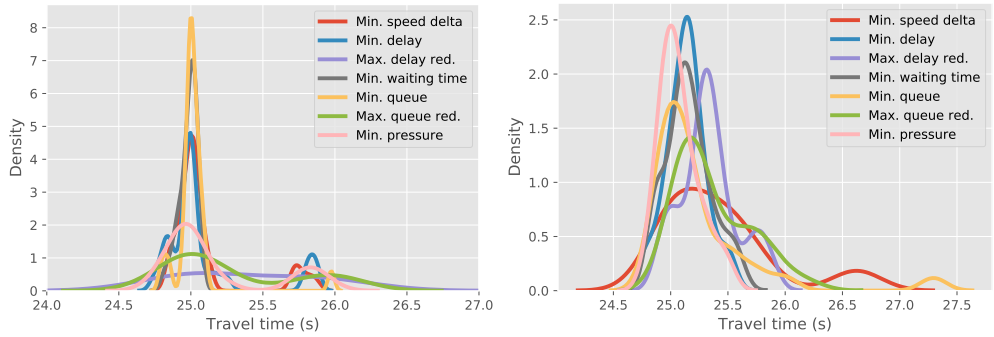


(a) Average action. **(b)** Instantaneous rewards.

Figure 6.3: (*Intersection* network, *high constant* demand) Training curves observed for the Q-learning agent with the *min. speed delta* MDP.

taken in order to speed up the learning process of the algorithm, such as the use of an additional replay memory, the aforementioned MDPs would still require more steps in order for the training process to fully converge since their state is composed of four components, whereas the state of all the remainder MDPs is composed of only two components. Figure 6.2 displays the ϵ exploration parameter throughout training (as previously stated in Section 5.3.1, ϵ is decayed depending on the number of times each state was visited). As it can be seen, for the *min. speed delta* MDP, new states are still being visited at the end of the learning procedure. This comes as no surprise since the Q-table grows exponentially in the number of state components. Therefore, as noticeable in Figure 6.3, the agent fails to steadily converge to signal plans that prioritize phase 2 in 50 000 training cycles, approximately equivalent to one month in simulation time.

With respect to the DQN controller, under the *high constant* demand (Tab. 6.3), all MDPs outperformed the random policy by a significant margin. However, the *max. delay reduction* MDP features a



(a) DQN agent. **(b)** DDPG agent.

Figure 6.4: (*Intersection* network, *high constant* demand) Kernel density estimation of the travel time means distribution.

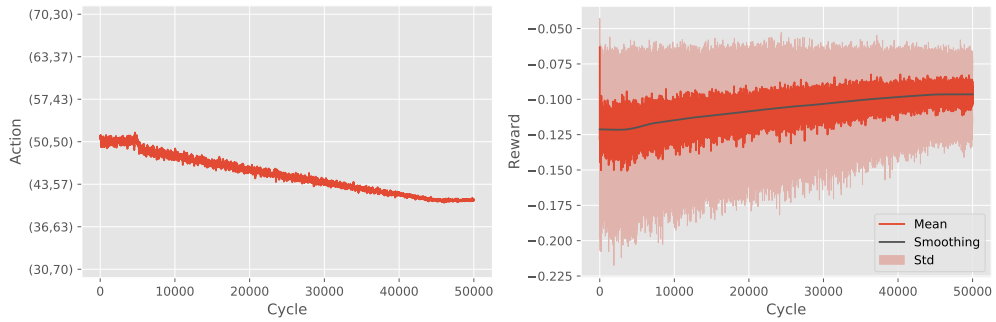
higher average travel time in comparison to the remainder formulations, despite the fact that the performance of the top-3 policies is generally similar among all MDPs. Figure 6.4(a) displays the travel time means distribution for the different MDPs. It is noticeable the fact that the distributions for the majority of the MDPs are multi-modal, however, both the *max. delay reduction* and the *max. queue reduction* MDPs distributions exhibit a higher degree of spread.

For the *low constant* demand (Tab. 6.4), the DQN controller outperforms the random policy for all MDPs except the ones which state calculation depends on queue-related information, despite the fact that all MDPs show improved performance over the random policy with respect to the top-3 travel times. For the remainder, non queue-based MDPs, the *max. delay reduction* MDP performed the worst, achieving an average travel time ≈1.4 seconds higher, on average, comparing to the remainder formulations.

With respect to the DDPG controller, under the *high constant* demand (Tab. 6.3), all MDPs performed better than the random policy, for all performance metrics. As seen in Figure 6.4(b), the *min. speed delta* and *min. queue* MDPs exhibit a slightly higher spread in terms of mean travel times.

Finally, regarding the DDPG controller, under the *low constant* demand (Tab. 6.4), the two queue-based MDPs exhibit the highest travel times, while the remainder are roughly similar in terms of performance. The top-3 travel times are similar for all formulations except for the *max. queue reduction* MDP, which top-3 travel time is at least ≈0.8 seconds higher in comparison to the other MDPs.

In contrary to the Q-learning algorithm, both the DQN and DDPG controllers are able to learn useful behavior independently of the size of the state space. As an example, while the Q-learning agent is unable to properly converge for the *min. speed delta* MDP, the DQN agent is able to properly generalize from the gathered experience, steadily converging towards signal plans that prioritize phase 2 (Fig. 6.5). Generally, both the DQN and DDPG agents achieved improved average travel times in comparison to the Q-learning agent, as seen in Table 6.5. The biggest improvement in performance occurred for the



(a) Average action.

(b) Instantaneous rewards.

Figure 6.5: (*Intersection* network, *high constant* demand) Training curves observed for the DQN agent with the *min. speed delta* MDP.

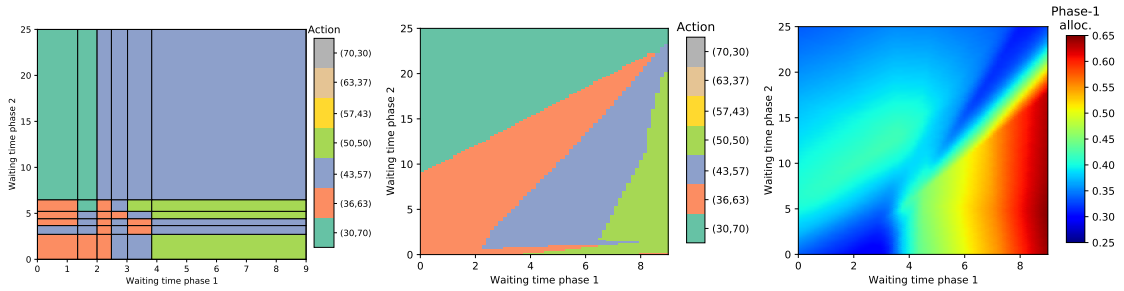
Table 6.5: (*Intersection* network) RL algorithms performance comparison (averaged for all MDPs).

	Travel time (*high*)	**Travel time (*low*)**
Q-learning	25.65	23.37
DQN	25.26	23.18
DDPG	25.24	23.17

three MDPs that feature a larger state space, but the improvement was not just restricted to that class of MDPs. However, the improvement in performance was not one-directional, with a minority of MDPs showing no improvement, or even a decrement in performance.

The use of a continuous action space (DDPG) does not provide, according to the experimental data, a substantial improvement in performance over a discrete action space (DQN). Such observation can be justified by the fact that, for a considerable interval around the optimal phase allocation, no significant difference in performance is noted. Figure 6.1 supports this argument. As it can be seen, the optimal phase 1 allocation sits on a plateau, i.e., a coarse approximation of the optimal value achieves a similar performance than the exact optimal allocation. Therefore, no great loss in performance occurs by discretizing the action space into a set of signal plans, as it is the case with the DQN agent.

Figure 6.6 displays three policies, sampled among the best performing policies that resulted from the training procedure, for each of the RL algorithms with the *min. waiting time* MDP. As it can be seen, all three policies prioritize phase 2 by allocating a larger portion of the cycle length to this phase, a sensible choice given the intersection's topology. However, depending on the observed average waiting time for each of the phases, RL agents exhibit fine-tuned behavior. As an example, whenever the average waiting time of phase 2 is high and the average waiting time of phase 1 is low, RL agents allocate almost the entirety of the cycle length to phase 2. However, as the average waiting time of phase 1 increases, a larger portion is allocated to this phase. In the limit, the cycle length is roughly equally split for both phases. Despite the fact that all three policies achieve similar average travel times under the current experimental setup, it can be seen that the DQN and DDPG algorithms display improved



(a) Q-learning policy (Travel time: 24.95).

(b) DQN policy (Travel time: 24.82).

(c) DDPG policy (Travel time: 24.87).

Figure 6.6: (*Intersection* network, *high constant* demand) Illustration of three sampled policies that resulted from the training procedure for each of the RL algorithms with the *min. waiting time* MDP.

Table 6.6: (*Intersection* network) Average travel time for two disjoint sets of MDPs, calculated using the performances obtained by the DQN and DDPG algorithms.

	Travel time (*high*)	**Travel time (*low*)**
Max. delay red. and *max. queue red.* MDPs	25.53	23.98
Min. delay and *min. queue* MDPs	25.10	23.45

generalization in comparison to the Q-learning agent: the discretization procedure limits the adaptability of the Q-learning agent for the parts of the state space where the bins are more sparse. On the other hand, the DQN and DDPG algorithms display decision boundaries that smoothly extend to the entirety of the state space, which can help the agents to perform more robustly under traffic demand changes. Moreover, it appears that tile-coding may not be the most suitable discretization technique for the present experimental setup given the fact that both the DQN and DDPG policies suggest decision boundaries that are not parallel to the state features axis. We observe a similar trend among all policies.

Regarding the different MDPs, a noticeable difference in performance resides in the fact that the formulations which reward consists in a difference/variation between adjacent decision steps appear to perform worse in comparison to the remainder MDPs, which define the reward as a point value at a given decision step, independently of the RL algorithm. As an example, for both the DQN and DDPG agents, the *max. delay reduction* and *max. queue reduction* MDPs (variation/difference-based MDPs) are, respectively, outperformed by their closest variants, the *min. delay* and *min. queue* MDPs, which reward is defined as an absolute value in time. Table 6.6 summarizes the differences in performance between the two categories of MDPs.

Secondly, queue-based MDPs appear to be inappropriate under mid/low traffic demands since the state information is unable to capture the presence of vehicles at the intersection due to the lack of queues. The results obtained by both the DQN and DDPG agents under the *low constant* demand (Tab. 6.4) support this finding. As it can be seen, the queue-based MDPs, i.e., the *min. queue* and *max. queue red.* formulations, achieved the highest average travel times, exhibiting inconsistent per-

Table 6.7: (*Intersection* network, *variable* demand) Baselines performance metrics.

Controller	Travel time free-flow (s)	Travel time congested (s)	Travel time (s)	Top-3 travel time (s)
Webster	22.1	25.1	23.7	23.5
Max-pressure	18.2	23.3	21.0	20.9
Actuated	19.8	24.8	22.0	21.9

formances. Despite the poor performance recorded under low congestion, the fact that the *min. queue* definition achieves, with respect to the *high constant* demand (Tab. 6.3) and for both the DQN and DDPG controllers, a similar average travel time than the best performing MDPs is a good indicator that, indeed, queue information reveals to be less useful only under mid/low-congested situations.

In summary, the present experimental setup shows that: (i) the relative performance between the different controllers is dependant on the demand intensity; (ii) the DQN and DDPG controllers generally outperform the Q-learning controller, however, no significant difference is observed between continuous and discrete control; and (iii) queue-based MDP formulations are not suitable for low-demand settings, and the reward function is best formulated as an absolute value at a given the decision point. Finally, given the time-*constant* demands that comprise this scenario, as well as the reactiveness and time-scale adaptability of the chosen action space definition, it is not expected that the periodic RL controllers outperform the non-periodic baselines. Nevertheless, as it can be seen, the performance of the RL controllers is generally on par with that of the best known static timing, even equalling, in some cases, the performance of the actuated controller for highly congested settings. Importantly, the majority of the RL controllers outperformed the well-known Webster method for static timings calculation, showing that RL can, indeed, learn useful strategies for TLC.

6.1.2 *Variable* demand

Similarly to the previous scenario, both the max-pressure and actuated controllers outperform the Webster method, as seen in Table 6.7. This improvement in performance is most noticeable during the free-flow regime, similarly the previous section.

Being the reward intrinsically dependent on the traffic volume at the intersection, we studied whether it is useful to include additional information in the state, such as the hour of the day. This kind of contextualizing information, named *time variable*, helps the agent to decouple if a certain reward gain/loss was due to the choices of its actions, or due to external changes in the traffic volume induced by the simulator. Figure 6.7 displays a barchart with the obtained cumulative rewards for a subset of MDPs, with and without additional contextualizing information in the state space. For both the *min. speed delta* and *min. waiting time* MDPs, a clear improvement in the cumulative reward is noticeable. However, for the *max. queue red.* MDP no significant difference is noted. Such observation seems reasonable since the reward definition of the *max. queue red.* MDP is defined as a difference between decision steps,

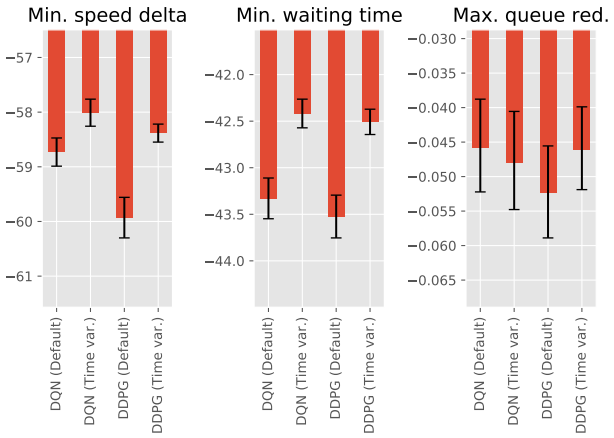


Figure 6.7: (*Intersection* network, *variable* demand) Cumulative reward for different MDPs with and without including the additional *time variable* in the agent's state. Error bars represent the 95% bootstrapped mean confidence interval.

Table 6.8: (*Intersection* network, *variable* demand) DQN controller performance metrics (with the additional *time variable*).

	Travel time free-flow (s)	**Travel time congested (s)**	**Travel time (s)**	**Top-3 travel time (s)**
Random	23.4	28.9	25.8	25.5
Min. speed delta	21.5	25.6	23.4	23.1
Min. delay	21.7	25.4	23.5	23.1
Max. delay reduction	22.5	28.3	25.0	23.1
Min. waiting time	21.6	25.5	23.4	23.1
Min. queue	22.1	26.2	24.0	23.1
Max. queue reduction	22.6	29.6	26.0	23.1
Min. pressure	21.8	25.5	23.4	23.1

therefore already contributing to rule out the influence of the traffic volume variation throughout the day on the reward. All formulations which reward is defined as an absolute value at a given decision point steadily improved their cumulative reward with the inclusion of the *time variable*, thus revealing that such additional contextualizing information is beneficial to learning under time-varying demands. With respect to the Q-learning agent, the inclusion of additional information to the state revealed to be prohibitive due to convergence problems.

Table 6.8 shows the performance metrics obtained for the DQN agent. The other RL algorithms, Q-learning and DDPG, behave in a way that is qualitatively similar to that reported in the previous section, and we omit their discussion for brevity. The corresponding results are included in Appendix B. Regarding the DQN agent, all MDPs, except the *max. queue reduction* definition, outperformed the random policy, achieving a lower average travel time for all demand regimes. Three MDPs, namely the *max. delay reduction*, *min. queue* and *max. queue reduction* formulations are outperformed by the remainder definitions, despite the fact that all formulations achieved the same performance with respect to the top-3 travel time metric.

Similarly to Section 6.1.1, the relative performance between the different controllers is dependant

Table 6.9: (*Intersection* network, *variable* demand) RL algorithms performance comparison excluding the *max. delay reduction*, *min. queue* and *max. queue reduction* MDPs.

	Travel time
Q-learning (w/o *time var.*)	23.87
DQN (w/ *time var.*)	23.42
DDPG (w/ *time var.*)	23.40

on the demand regime. As an example, the DQN agent with the *min. speed delta* MDP attains an average travel time ≈0.3 seconds lower than the Webster controller, in spite of the fact that it exhibits a higher average travel time during the congested regime, as seen in Tables 6.7 and 6.8. Therefore, while time-variable demands contribute to richer simulations, it is generally more challenging to analyze and compare the performances of the different controllers as, most of the time, the improvement/decline in performance is not uni-directional for all traffic regimes.

With respect to the MDP definitions, as previously mentioned, three formulations, namely the *max. delay reduction*, the *min. queue* and the *max. queue reduction* formulations, are generally outperformed by the remainder definitions. Given the discussion in the previous section, this comes as no surprise: the MDPs that achieve the worst performances are those which reward is defined as a difference between consecutive timesteps, or reside on queues' information for state/reward computation.

The Q-learning controller revealed to be more unstable in comparison to the other RL algorithms, mostly due to convergence problems, while both the DQN and DDPG controllers achieved, on average, similar performances as seen in Table 6.9.

Finally, the performance of the RL controllers is generally well aligned with that of the periodic baselines, despite the fact that under the congested regime higher average travel times are sometimes recorded in comparison to the Webster method. Given the present topology, RL methods are generally outperformed by the non-periodic controllers for the reasons previously discussed.

In summary, the obtained results show that: (i) the relative performance of the controllers is demand-dependant; (ii) the inclusion of a *time variable* in the agent's state space is beneficial to the learning procedure; (iii) the Q-learning controller is normally outperformed by the DQN and DDPG algorithms, however, no significant difference is noticeable between discrete and continuous control; and (iv) queue-based MDPs, and the formulations which reward is defined as a difference between adjacent decision steps are outperformed by the remainder definitions.

6.1.3 *Cyclical* demand

The third experimental setup comprises a single intersection with time-varying traffic volumes that cyclically put uneven pressure in the intersection's phases. Table 6.10 shows the obtained results for the baselines. As it can be seen, both periodic methods are outperformed by the non-periodic baselines. Moreover, the adaptive-Webster controller is able to achieve improved adaptability to the time-changing

Table 6.10: (*Intersection* network, *cyclical* demand) Baselines performance metrics.

Controller	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Webster	8.4	24.8	24.6
Adaptive-Webster	7.9	24.1	24.0
Max-pressure	5.5	21.9	21.8
Actuated	6.5	22.8	22.6

Table 6.11: (*Intersection* network, *cyclical* demand) DQN controller performance metrics.

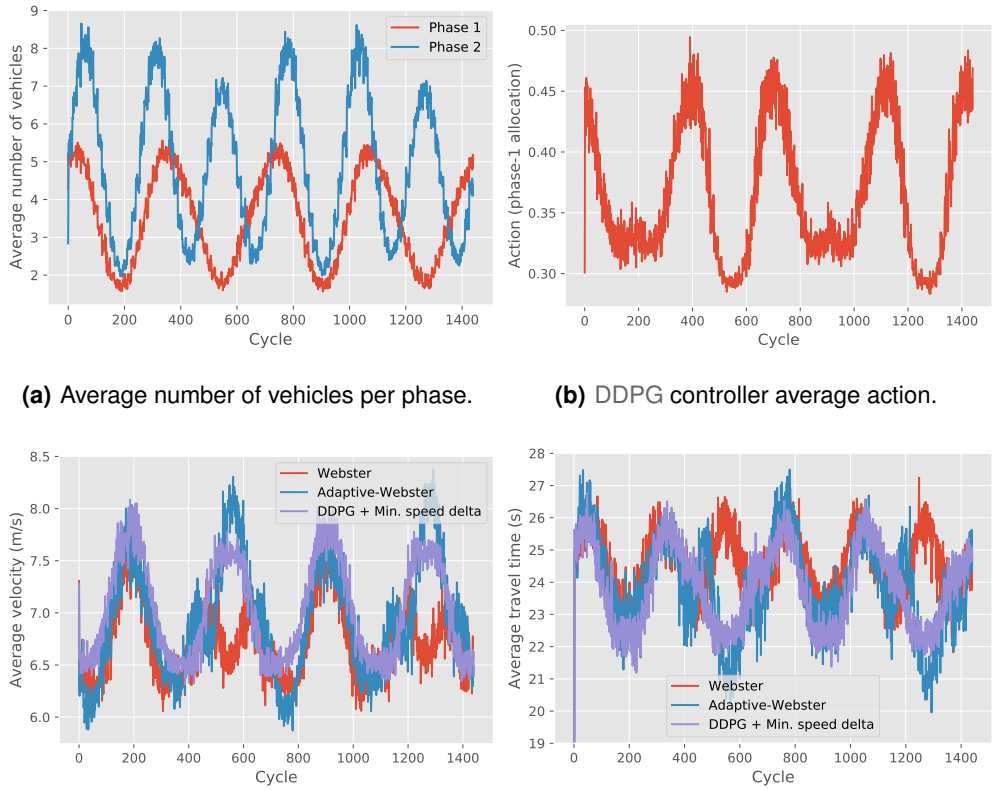
	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Random	10.2	26.9	26.6
Min. speed delta	8.1	24.0	23.8
Min. delay	8.2	24.2	24.0
Max. delay reduction	9.6	26.1	24.3
Min. waiting time	8.2	24.3	23.9
Min. queue	8.9	25.2	24.2
Max. queue reduction	9.2	25.7	24.3

demands, exhibiting a lower average travel time, as well as average waiting time, in comparison to the default Webster method.

Regarding the DQN controller, Table 6.11 displays the results for a set of formulations. All MDPs outperformed the random policy with respect to all reported metrics. However, the *max. delay reduction*, *min. queue* and *max. queue reduction* MDPs exhibit higher average travel times in comparison to the remainder definitions. Similarly to the previous sections, the MDPs that define the reward function as a difference between consecutive decision points, as well as the queue-based definitions, are outperformed by the other formulations. The *min. speed delta* formulation yielded the lowest travel time overall. The other RL algorithms, Q-learning and DDPG, behave in a way that is qualitatively similar to that reported in the previous sections, and we omit their discussion for brevity. The corresponding results are included in Appendix B.

Given the demand considered in the present scenario, it is expected that methods that dynamically adjust to the phase-uneven, time-changing demands, achieve improved performance over static timings. This is indeed the case with the baselines, as the adaptive-Webster controller is capable of outperforming the static timings calculated using the Webster method. Once again, as expected given the current network topology, both non-periodic methods outperform the periodic baselines.

Obtained results for the RL-based controllers outperform the static Webster timings, showing that, indeed, RL is able to learn useful behavior in the context of TLC. Moreover, some RL algorithm and MDP combinations, such as the case with the DQN agent with the *min. speed delta* MDP, are also able to slightly outperform the adaptive-Webster method with respect to the travel time metrics. Figure 6.8 displays a set of plots that allow to take a closer look into the adaptability of different controllers to the time-changing demands. As it can be seen in Figures 6.8(c) and 6.8(d), both the RL controller and the adaptive-Webster method are able to achieve reduced average travel times, as well as increased average speeds, throughout the entire day in comparison to the default Webster method. The static



(a) Average number of vehicles per phase.

(b) DDPG controller average action.

(c) Average velocity.

(d) Average travel time.

Figure 6.8: (*Intersection* network, *cyclical* demand) Curves showing the adaptability of the DDPG agent (which achieved an average travel time of 24.0 seconds), and other baseline methods, to the *cyclical* demand.

timings calculated using the Webster method reveal to be inefficient at handling the time-changing demands (Fig. 6.8(a)) during certain periods of the day. On the other hand, both the RL controller and the adaptive-Webster method are able to dynamically adapt the phase splits depending on the perceived state of traffic congestion, as seen in Figure 6.8(b). Such online calibration of the phase-splits allows the adaptive methods to more consistently achieve reduced average travel times throughout the evaluation rollouts (Fig. 6.8(d)). Finally, it is noticeable from the same set of plots that the RL controller achieves a more consistent performance in comparison to the adaptive-Webster controller. As an example, the adaptive-Webster method records the highest and lowest average speeds among all controllers during the evaluation rollouts, while the RL controller performs in a more consistent manner (Fig. 6.8(c)).

In summary, the present experimental setup shows that: (i) queue-based MDP formulations, and the MDPs which reward is defined as a difference between adjacent decision steps are outperformed by the other definitions; (ii) the Q-learning controller is normally outperformed by the DQN and DDPG algorithms, however, no significant difference is noticeable between discrete and continuous control; and (iii) RL controllers are capable of clearly outperforming the static signal plans calculated with the Webster method, achieving a similar performance with that of the adaptive-Webster controller.

6.1.4 Takeaways

Obtained experimental results show that, for the present network topology and independently of the demand type, non-periodic methods, i.e., the max-pressure and actuated controllers, outperform the periodic controllers, i.e., the RL-based, Webster and adaptive-Webster controllers. This observation is, as previously discussed, majorly justified by the fact that the non-periodic controllers exhibit a higher degree of flexibility and reactiveness in comparison to the periodic controllers. Non-periodic methods are able to dynamically adjust the cycle length, allowing the controller to more timely and efficiently allocate the green time based on current demands. As the obtained experimental results suggest, the improved adaptability over periodic controllers is more evident in low/mid congested scenarios since the controller is able to skip phases as soon as no cars are detected. Such trend comes as no surprise given the fact that fully-actuated non-periodic controllers are recommended, among the traffic engineering field, to be used for control of isolated intersections: "*Research has shown that the best form of isolated [intersection] operation occurs when fully-actuated controllers are used. Actuated controllers operate most effectively when timed in a manner that permits them to respond rapidly to fluctuations in vehicle demand.*" - U.S. Department of Transportation, Traffic Signal Timing Manual, p. 5-17 [5].

Despite the fact that the present study considers a rather constrained action space definition, making it harder for the RL-based controllers to outperform the non-periodic baselines for the current network topology, RL revealed to learn useful strategies for TLC. As our results suggest, the performance obtained by the RL-based controllers is well inline with that of the periodic baselines, for all tested types of demands. Especially under the *cyclical* demand, RL methods displayed improved performance, being able to opportunely adapt the executed signal plans in such a way that the average travel time is minimized throughout the entire simulation, clearly outperforming the computed static timings. For different experimental setups, RL-based controllers were capable of outperforming the commonly used Webster method for static timings calculation.

Among the three tested RL methods, the Q-learning agent was, in general, outperformed by both the DQN and the DDPG agents. The use of higher dimensional state spaces revealed to be prohibitive due to the slow convergence of the algorithm. Furthermore, the discretization process revealed to be cumbersome: the bins need to be independently calculated for each component of the state, and it is generally hard to pick the right number of bins. No significant difference in performance is noticeable between discrete and continuous control.

With respect to the different assessed MDP formulations, under all demand types, queue-based MDPs and the formulations which reward is defined as a difference between adjacent decision steps are outperformed by the other definitions. Given the fact that this observation is consistent for all experimental setups up to this point, it is considered that, indeed, these formulations are the less suitable to be used in the context of this study and, therefore, their results will be omitted from now on.

Table 6.12: (*Arterial* network, *constant* demand) Baselines performance metrics.

Controller	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Webster	10.8	34.4	34.3
Max-pressure	8.6	35.6	35.2
Actuated	10.0	37.9	37.7

Table 6.13: (*Arterial* network, *constant* demand) DQN controller performance metrics.

	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Random	15.4	40.8	40.5
Min. speed delta	10.7	34.3	34.2
Min. delay	10.7	34.3	34.1
Min. waiting time	10.7	34.3	34.1
Min. pressure	14.8	40.8	35.9

6.2 *Arterial* network

The present section assesses and discusses the performance of different TSCs using an *arterial* traffic network under different demands. For the purposes of brevity, we include the results for the constant demand only - the other demand types lead to similar conclusions and are provided in Appendix B. Results for the RL controllers are obtained from 15 independent training runs, each of which evaluated using three rollouts of 24 hours. Non-trainable methods are evaluated with a set of 15 rollouts of 24 hours.

6.2.1 *Constant* demand

Table 6.12 presents the obtained results for the different baselines. As it can be seen, the periodic controller, i.e., the Webster method, is capable of achieving lower average travel times in comparison to the non-periodic baselines, despite the fact that both the max-pressure and actuated controllers outperform the Webster method with respect to the average waiting time metric.

Regarding both the DQN and DDPG controllers (Tabs 6.13 and 6.14), all MDPs except the *min. pressure* formulation outperformed the random policy. Table 6.13 displays the obtained experimental results for the DQN controller under a subset of formulations. All definitions, expect the *minimize pressure* MDP, exhibit a similar performance for all reported metrics. With respect to the DDPG agents (Tab. 6.14), a similar trend is observed, however, the registered average travel times are higher, except for the *min. pressure* definition, in comparison to the DQN agents.

In contrast to the results obtained for the *intersection* network, both non-periodic baselines are outperformed by the Webster method in the context of the *arterial* network. Despite the fact that both the max-pressure and actuated controllers are capable of reducing the average waiting time locally to each intersection, as seen in Tab. 6.12, the miscoordination between the signal timings along the major street leads to an overall increase in the average travel time of the vehicles. An additional evidence

Table 6.14: (*Arterial* network, *constant* demand) DDPG controller performance metrics.

	Waiting time (s)	**Travel time (s)**	**Top-3 travel time (s)**
Random	18.0	44.5	44.1
Min. speed delta	10.9	34.6	34.4
Min. delay	11.0	34.7	34.4
Min. waiting time	10.9	34.7	34.4
Min. pressure	11.8	36.3	34.3

Table 6.15: (*Arterial* network, *constant* demand) Controllers average speed metric, calculated, for each trip, as an average of the instantaneous velocity of the vehicle at each simulation second.

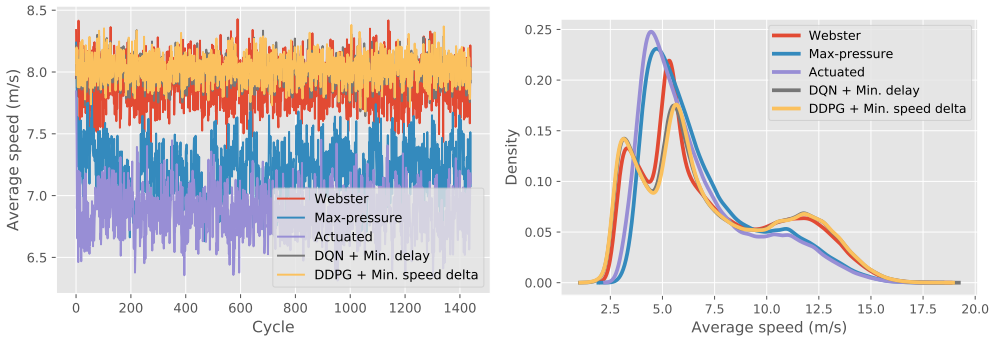
Controller	**Speed (m/s)**
Webster	6.78
Max-pressure	6.49
Actuated	6.20
DQN + *min. delay*	6.82
DDPG + *min. speed delta*	6.79

that supports this claim can be seen in Table 6.15, which presents the average vehicles' speed for a set of methods. As shown in the table, both non-periodic baselines exhibit a lower average speed as to the remainder controllers; on the other hand, the two RL methods and the Webster controller achieve improved average speeds. The fact that the non-periodic methods do not have a fixed cycle length and are highly reactive to the arrival of vehicles contributes to the desynchronization of the signal plans of the intersections along the major street, not allowing the free-flow of vehicles. On the other hand, due to the fact that the RL methods have a fixed cycle length and the cycles between adjacent intersections are in sync, a green wave[1] occurs, allowing the vehicles to more freely travel across the major street.

Figure 6.9 provides complementary speed-related information, allowing to take a more in-depth look with respect to this metric. It is clear from Figure 6.9(a) that both RL methods and the Webster controller achieve an improved average speed in comparison to the remainder methods throughout all the simulation (24 hours). Moreover, despite the fact that the DQN and DDPG controllers perform roughly equivalent to the Webster method with respect to the scalar average speed metric (Tab. 6.15), it can be seen that the RL methods exhibit, positively, a slightly more consistent average speed than the Webster controller. The gap in performance between the non-periodic and periodic controllers is also visible in the kernel density estimation of the trips' average speed (Fig. 6.9(b)).

In summary, according to experimental observations: (i) periodic controllers achieve improved performance over the non-periodic methods; (ii) the *min. pressure* MDP is outperformed by the other studied definitions; and (iii) the DQN algorithm achieved lower average travel times in comparison to the DDPG algorithm.

[1] A green wave occurs when a series of traffic lights are coordinated to allow continuous traffic flow over several intersections in one main direction.



(a) Average velocity of the vehicles during the evaluation rollouts.

(b) Kernel density estimation of the average speed distributions.

Figure 6.9: (*Arterial* network, *constant* demand) Controllers average speed metric plots.

6.2.2 Takeaways

Experimental results show that, if no explicit coordination mechanisms are considered in the context of non-periodic methods, coordination problems arise, harming the efficiency of the traffic network. On the other hand, periodic methods, which can be easily synchronized between adjacent intersections, showed improved performance with respect to the average travel time metric.

In contrast to the results reported for the previous network, the more restrictive and less flexible action space definition revealed to suit well the current *arterial* network. While in the context of the *intersection* network, the use of a fixed cycle length limits the adaptability of the RL agents, for the current *arterial* topology, it facilitates the coordination of adjacent intersections, a key feature to improved global network performance: "*For intersections located along arterial streets, isolated operation can often be improved by considering coordination of the major street movements along the arterial. Common cycle lengths are often employed to facilitate this coordination.*" - U.S. Department of Transportation, Traffic Signal Timing Manual, p. 3-2 [5]. Obtained experimental data supports the arguments given in Section 3.3.3 that justified the chosen action space definition, revealing it to be suitable to be used under a fully decentralized approach to TLC, where no coordination mechanisms are explicitly considered among the RL controllers that compose the multi-agent system.

Generally, the performance of the RL controllers is well inline with that of the best performing baselines, i.e., the Webster and adaptive-Webster controllers. Three MDPs, namely the *min. speed delta*, *min. delay* and *min. waiting time* formulations, outperformed the *min. pressure* MDP for all performance metrics and experimental setups. Finally, no significant improvement was registered with the use of continuous control over discrete control; on the contrary, the DQN agent achieved the lowest registered travel times.

Table 6.16: (*Grid* network, *constant* demand) Baselines performance metrics.

Controller	Waiting time (s)	Travel time (s)	Top-1 travel time (s)
Webster	22.2	65.9	65.7
Max-pressure	12.7	59.5	59.2
Actuated	13.9	61.0	60.7

Table 6.17: (*Grid* network, *constant* demand) DQN controller performance metrics.

	Waiting time (s)	Travel time (s)	Top-1 travel time (s)
Random	27.4	72.9	72.6
Min. speed delta	22.5	66.2	65.1
Min. delay	22.5	66.1	65.6
Min. waiting time	22.8	66.4	65.4
Min. pressure	27.2	72.2	71.8

6.3 *Grid* network

The present section assesses and discusses the performance of different TSCs using a *grid* traffic network under different demands. Results for the RL controllers are obtained from six independent training runs, each of which evaluated using three rollouts of 24 hours. Non-trainable methods are evaluated with a set of six rollouts of 24 hours.

6.3.1 *Constant* demand

Table 6.16 presents the obtained results for the different baselines. As it can be seen, the Webster method is outperformed by the non-periodic baselines with respect to all reported metrics. In turn, the max-pressure method achieves lower average waiting and travel times in comparison to the actuated controller.

Regarding the RL agents, Tables 6.17 and 6.18 present, respectively, the obtained results for the DQN and DDPG controllers. It is clear from both tables that the *min. pressure* MDP is the less suitable for the present experimental setup, exhibiting the highest recorded average travel and waiting times. Furthermore, it can be seen that the DDPG controller achieves a lower average travel time in comparison to the DQN controller for three out of the four MDPs, achieving the best overall average travel times with the *min. delay* and *min. waiting time* formulations. Figure 6.10 complementarily compares the DQN and DDPG algorithms with respect to the obtained cumulative reward during the evaluation rollouts. As noticeable, the DDPG algorithm is able to achieve a significantly higher cumulative reward for the *min. delay* and *min. waiting time* MDPs.

Given the fact that the presently considered demands are time-constant, it is expected that the different agents, each responsible for the control of a single intersection, converge to a relatively stable strategy. This is indeed the case, as seen in Figure 6.11(b): the agents, on average, exhibit a time-invariant behavior. Nevertheless, the displayed behavior is fine-tuned depending on the assigned intersection.

Table 6.18: (*Grid* network, *constant* demand) DDPG controller performance metrics.

	Waiting time (s)	Travel time (s)	Top-1 travel time (s)
Random	31.8	78.4	78.0
Min. speed delta	22.7	66.5	66.2
Min. delay	21.4	64.7	63.1
Min. waiting time	21.0	64.1	63.4
Min. pressure	23.6	67.6	67.2

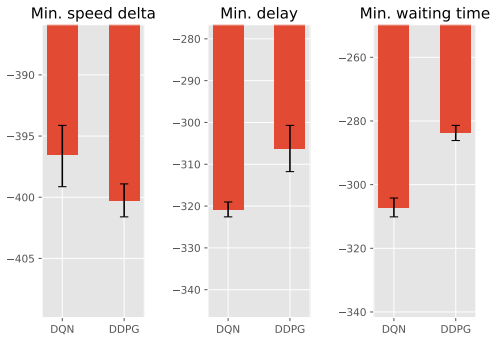
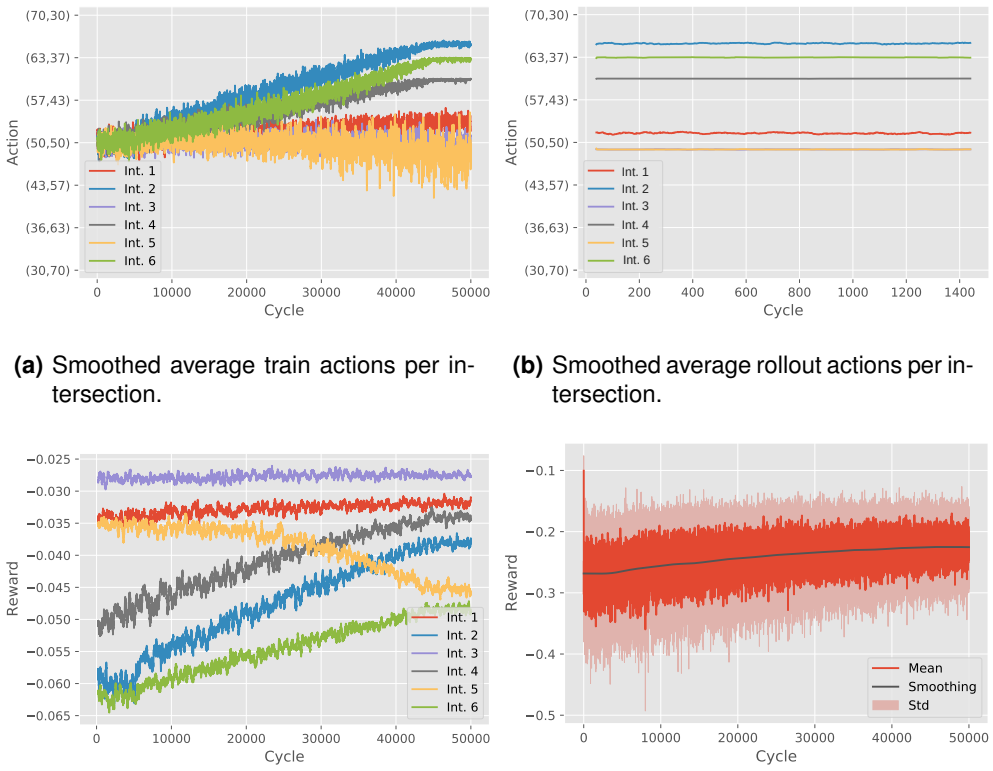


Figure 6.10: (*Grid* network, *constant* demand) Cumulative reward obtained by the DQN and DDPG algorithms for three MDPs. Error bars represent the 95% bootstrapped mean confidence interval.

As an example, the agents along Av. 5 de Outubro, assigned to intersections 2, 4 and 6, prioritize the movement of vehicles along the avenue by allocating a larger fraction of the cycle length to phase 1, associated with the movement of vehicles in the vertical direction.

As previously discussed, the problem of RL-based TLC can be seen as an inherently cooperative game between different agents. In its simpler form, a RL controller is developed for each of the intersections and no explicit coordination mechanisms are considered between agents, as it is the case with the present study. As noticeable in Figure 6.11(d), a clear improvement in the global instantaneous reward (sum of the rewards of all agents) obtained throughout the training procedure occurs. However, Figure 6.11(c) unveils the fact that, despite the general improvement in performance of the global traffic network, one of the agents actually obtains progressively lower instantaneous rewards as the training proceeds. Such observation is usually associated with badly-convergent policies. One possible explanation to this observation is related to the fact that intersection 5 (Figs. 5.2(b) and 5.3(e)) is the only intersection that has a phase (phase 2, horizontal direction) where the traffic volume is entirely dependent on the behavior of another agent (in this case, the agent assigned to intersection 6). For all the other intersections (Fig. 5.3), each phase has at least one incoming approach which volume is not regulated by another agent but it is instead dictated by the simulator (boundary incoming approaches, also known as source edges). Therefore, as the training proceeds and the agent assigned with intersection 6 progressively changes its behavior, it creates non-stationary conditions for agent 5 that hinder its learning process. Whether the inclusion of explicit coordination mechanisms between adjacent agents contributes to the stabilization of the learning process, and such improvement in turn promotes an increase in performance



(a) Smoothed average train actions per intersection.

(b) Smoothed average rollout actions per intersection.

(c) Smoothed average instantaneous rewards per intersection.

(d) Global instantaneous rewards.

Figure 6.11: (*Grid* network, *constant* demand) Curves for the DQN agents with the *minimize delay* MDP.

of the global network is an open question of great interest. However, its study is outside the scope of the present work.

Finally, it is interesting to notice that, despite the steady improvement in reward throughout the training process, the agent assigned to intersection 6, the intersection with the highest number of incoming lanes (Fig. 5.3), is the most penalized with respect to the obtained rewards.

Despite the fact that we detected a minority of badly convergent policies, RL-based TSCs were capable of learning efficient TLC strategies. RL policies, especially with respect to the DDPG algorithm, were capable of outperforming the periodic baseline method. As an example, the DDPG controller with the *min. waiting time* MDP achieved an average travel time ≈1.8 seconds lower in comparison to the Webster method.

In summary, according to experimental observations: (i) non-periodic controllers achieve improved performance over the periodic methods; (ii) the *min. pressure* MDP is outperformed by the other studied definitions; and (iii) the DDPG algorithm, with the *min. speed delta* and *min. waiting time* MDPs, achieved the lowest recorded average travel times and average waiting times among the RL controllers.

Table 6.19: (*Grid* network, *variable* demand) Baselines performance metrics.

Controller	Travel time free-flow (s)	Travel time congested (s)	Travel time (s)	Top-1 travel time (s)
Webster	57.9	65.7	61.5	61.4
Max-pressure	46.0	61.8	55.6	55.3
Actuated	55.0	63.5	58.5	58.3

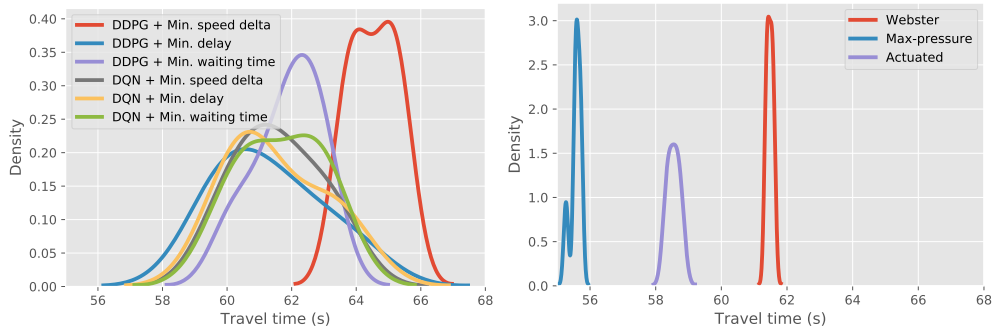
6.3.2 *Variable* demand

Similarly to the previous demand type, both the max-pressure and actuated controllers outperform the Webster method, as seen in Table 6.19. This improvement in performance is most noticeable during the free-flow regime.

Regarding the RL controllers, Figure 6.12 shows an estimation of the distributions of the travel time means for a subset of MDPs. As noticeable in Figure 6.12(a), the DQN agents generally achieve lower average travel times in comparison to the DDPG agents, especially with respect to to the *min. speed delta* and *min. waiting time* MDPs. Figure 6.12(b) displays an estimation of the distributions of the travel time means obtained by the baseline controllers. It can be seen that, on average, the majority of the RL controllers attain a similar average travel time than the Webster controller. Unfortunately, RL controllers exhibit more inconsistent results in comparison to the baseline controllers (Figs. 6.12(a) and 6.12(b)). As an example, the difference between the highest and lowest average travel times for the Webster controller is of $\approx$0.2 seconds. On the other hand, for the DDPG algorithm with the *min. delay* MDP, the difference between the highest and lowest average travel times for the different policies sets that resulted from the training step is of $\approx$4.3 seconds. In other words, while some training runs generate sets of policies that clearly outperform the Webster method, this is not always the case.

The undesirable increased variability featured by the RL controllers should be carefully addressed, especially in cases where limited data is available for learning. However, such high variance in the obtained performances is common while applying RL methods in different domains [91], especially when the use of complex non-linear function approximators is considered. Therefore, it is frequent among the RL-related literature to only report results for the k best performing policies [91], as it is also the case with several works in the TLC domain. Figure 6.13 displays the average travel time of the k best performing policies for the DDPG controller with the *minimize delay* MDP, for different values of k. As it can be seen, a gap of $\approx$1.6 seconds exists between the best performing set of policies (top-1), and the average of all sets of policies (top-12). One could be tempted to only report that the DDPG controller achieves an average travel time $\approx$1.8 seconds lower than the Webster method, closing the gap to the actuated controller. While this is, indeed, the case for one of the policies set, this value may not be well representative of the underlying quality of the RL method.

The increased variability observed for the current six-intersection topology should come as no surprise given the fact that, even for the single agent setting (Section 6.1), RL controllers already exhibited



(a) RL controllers. **(b)** Baselines.

Figure 6.12: (*Grid* network, *variable* demand) Kernel density estimation of the distributions of travel time means for different controllers with the time-*variable* demand.

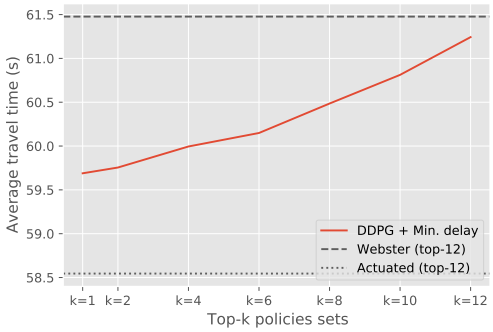


Figure 6.13: (*Grid* network, *variable* demand) Average travel time of the top-k policies sets that achieved the lowest average travel time, for different values of k (DDPG controller with the *minimize delay* MDP).

some degree of variability in the reported results, as seen in Figure 6.4: the variance in performance of each of the independent learners contributes to an increased overall variance in the performance of the network. Whether this problem is also being potentiated by the lack of explicit coordination mechanisms between the agents is an important question, however, its study is outside the scope of the present thesis. Nevertheless, while such variance in the observed performances for the RL-based controllers should be properly assessed and reported, it does not make RL agents less suitable for TLC in comparison to the baselines, since at deployment-time, the best performing policy can be picked.

Despite the increased performance variance exhibited by the RL controllers in comparison to the baselines, it can be seen that, only with time-*variable* demands, RL methods can achieve a reduction in the average travel time of up to $\approx$1.8 seconds in comparison to the Webster method.

In summary, according to experimental observations: (i) non-periodic controllers outperformed the periodic methods; and (ii) the DQN algorithm achieved more consistent and generally lower average travel times with respect to the studied MDPs. However, the best performing algorithm is dependent on the MDP.

Table 6.20: (*Grid* network, *cyclical* demand) Baselines performance metrics.

Controller	Waiting time (s)	Travel time (s)	Top-1 travel time (s)
Webster	23.1	67.5	67.1
Adaptive-Webster	22.1	66.0	65.8
Max-pressure	13.1	60.7	60.5
Actuated	14.9	62.3	62.1

Table 6.21: (*Grid* network, *cyclical* demand) DQN controller performance metrics.

	Waiting time (s)	Travel time (s)	Top-1 travel time (s)
Random	30.3	77.3	76.9
Min. speed delta	23.1	67.2	66.4
Min. delay	22.1	66.0	65.1
Min. waiting time	22.7	66.8	66.3
Min. pressure	27.2	72.7	71.2

6.3.3 *Cyclical* demand

The last demand considered in the context of the *grid* network is the *cyclical* demand. The demand is set up in such a way that the network's roads along the horizontal and vertical axis are unevenly pressured through time.

Table 6.20 presents the obtained results for the different baselines. As it can be seen, both the Webster and adaptive-Webster methods are outperformed by the non-periodic baselines with respect to all reported metrics. As expected given the *cyclical* demand, the adaptive-Webster tops the static timings calculated using the Webster method with respect to the average travel time, as well as the average waiting time.

Regarding the RL agents, Tables 6.21 and 6.22 present, respectively, the obtained results for the DQN and DDPG controllers. It is clear from both tables, and similarly to the previous scenarios, that the *min. pressure* formulation is the less suitable MDP, exhibiting the highest recorded average travel and waiting times. Furthermore, for both RL algorithms, the *min. delay* and *min. waiting time* MDPs outperformed the *min. speed delta* formulation with respect to all performance metrics. The lowest average top-1 travel times are achieved by the DDPG controller for the *min. delay* and *min. waiting time* MDPs.

As the previously presented experimental results suggest, RL methods are able to learn useful TLC strategies that can cope with the time-variable demands. Both the *min. pressure* and *min. speed delta* revealed to be the less suitable MDPs, with the later achieving a similar average travel time as the

Table 6.22: (*Grid* network, *cyclical* demand) DDPG controller performance metrics.

	Waiting time (s)	Travel time (s)	Top-1 travel time (s)
Random	36.3	85.0	83.9
Min. speed delta	23.3	67.6	66.6
Min. delay	22.4	66.4	64.6
Min. waiting time	22.1	66.1	64.9
Min. pressure	25.5	70.3	68.3

default Webster method. On the other hand, for the *min. delay* and *min. waiting time* MDPs, the two RL algorithms were able to achieve an average travel time up to ≈1.5 seconds lower in comparison to the default Webster method, and a similar performance than the adaptive-Webster method. However, a careful analysis of the performance with respect to the top-1 travel time metric unveils two important findings: (i) similarly to what was observed in Section 6.3.2, RL controllers exhibit a higher variability in performance in comparison to the baselines; and (ii) RL methods are capable of achieving the lowest recorded average travel times among the periodic controllers. As an example, the DDPG controller with the *min. delay* MDP outperforms the adaptive-Webster baseline by ≈1.2 seconds.

6.3.4 Takeaways

Obtained experimental results show that, for the *grid* network and independently of the demand type, non-periodic methods outperform the periodic controllers. This observation is majorly justified by the fact the present network does not possess a major direction of traffic flow, as opposed to the *arterial* network. Despite the fact that two streets feature a higher number of lanes and increased traffic (concretely, Av. 5 de Outubro and Av. Miguel Bombarda), they are perpendicular between each other. Therefore, as already thoroughly discussed in the previous sections, non-periodic controllers are able to achieve reduced average travel times over the periodic methods because they are able to successfully minimize the waiting time of the vehicles by efficiently allocating the green time based on current demands.

Despite the fact that the present study considers a rather constrained action space definition, making it harder for the RL-based controllers to outperform the non-periodic baselines for the current network topology, RL revealed to learn useful strategies for TLC. Each agent exhibited intersection-dependant behavior, prioritizing the movement of vehicles along the major streets. As the reported results suggest, the performance obtained by the RL-based controllers is well inline with that of the periodic baselines, for all tested types of demands. Despite the noted increase in performance variability in comparison to the baseline methods, for some experimental setups, RL methods were capable of outperforming the Webster method for both the *constant* and *variable* demands. However, the most noticeable improvement in performance occurred for the last studied demand type, i.e., the *cyclical* demand. For this demand type, RL methods displayed improved performance, being able to opportunely adapt the executed signal plans in such a way that the average travel time is minimized throughout the entire simulation, clearly outperforming the computed static timings. For different experimental setups, RL-based controllers also outperformed the adaptive-Webster method, achieving the lowest recorded average travel times among the periodic baselines.

With respect to the different assessed MDP formulations, under all demand types, the *min. pressure* definition achieved higher average travel times in comparison to the *min. speed delta*, *min. delay* and *min. waiting time* MDPs. Among the speed-based formulations, no significant improvement was noted

with the use of the more complex *min. speed delta* MDP over the *min. delay* MDP; on the contrary, the *min. delay* formulation achieved the most consistent, and generally lower, average waiting and travel times for the three demands and two RL algorithms. The *min. delay* and *min. waiting time* MDPs revealed to be the most suitable formulations to be used in the context of TLC, with no unidirectional improvement being noticed with the use of one MDP over the other, despite the fact that the *min. delay* definition achieved the lowest top-1 average travel times for the three demands.

Finally, the DDPG algorithm records the lowest top-1 average travel times for the three demands, however, such improvement becomes less substantial with respect to the other reported metrics.

7

Conclusion

Contents

The present thesis provides a thorough study of different RL-based TSC design parameters under different road network topologies and demand types. We consider a fully decentralized approach to TLC with a rather constrained action space definition: each agent is responsible for adjusting the phase-splits of a given intersection, and no explicit coordination mechanisms are considered among the agents. We provide a comparison between several state and reward function definitions, as well as three different classes of RL algorithms: Q-learning, a tabular method with discrete action support, DQN, a function approximation method with discrete action support, and DDPG, a deterministic actor-critic method with continuous action support.

Firstly, in order to consistently and fairly assess the performance impact of different RL-based TSC design parameters, we provide a novel methodology for the development intelligent TSCs. Such methodology contributes one step further towards a wider application of RL in TLC, by ensuring some level of standardization at the different stages of the experimental process: simulation setup, TSC design – MDP formulation and selection of the RL method – as well as performance estimation and comparison. We discuss good practices that contribute to a better interpretation of the experimental results and mitigate reproducibility issues. In particular, we propose the use of statistical testing as a tool to infer robust conclusions from experimental data.

Secondly, while following the proposed methodology, we assess the performance of different RL-based TSCs under three traffic networks: a single *intersection*, an *arterial* network, and a *grid* network. Experimental results reveal that the performance of the different TSCs is dependent on both the traffic network, as well as the traffic demand. With respect to the studied state and reward definitions, obtained results show that queue-related formulations are not suitable to be used under low traffic and, given the chosen action space definition, the reward is best formulated as an absolute value at the decision step, and not as a difference between consecutive decision steps. The speed-based and waiting time-based formulations revealed to be the most suitable to be used under the TLC problem. As experimental results suggest, the Q-learning algorithm is less suitable to handle the inherently continuous state spaces in comparison to the DQN and DDPG algorithms, exhibiting, in general, higher average travel times than the later algorithms. No significant and consistent improvement in performance is noted with the use of continuous over discrete control, or vice versa.

Despite the fact that we adopted a rather constrained action space definition, making it harder for the RL-based controllers to outperform the non-periodic baselines for the *intersection* and *grid* networks, the use of such action space revealed to possess some advantageous features over the more commonly used, highly flexible action space definitions. Due to its easiness of synchronization among adjacent agents, RL controllers achieved the lowest recorded average travel times for the *arterial* network, outperforming, for some experimental setups, all baseline controllers. Moreover, due to its simplicity, the chosen action space definition allows for the improved interpretability of the learned policies, especially

important in the context of TLC, a domain where the control over real-world TSCs must only be conceded to agents that exhibit a responsible and trustable behaviour.

We show that RL is able to find policies that are on par with classical controllers, which benefit from both human supervision and from decades-old literature from the transportation engineering domain. For all studied networks and demands, the performance obtained by the RL-based controllers is well inline, on average, with that of the periodic baselines. For different experimental setups, RL-based controllers outperformed the commonly used Webster method for static timings calculation. Specifically under phase-uneven, time-varying traffic volumes, RL methods demonstrated to be able to timely react to demand changes in such a way that the average travel time is minimized.

Finally, it is worth noticing that, while intelligent TSCs may greatly improve urban mobility, they are just an important piece in the transition to an efficient, reliable and environmentally sustainable transportation system. On the extreme case scenario, the development of efficient TSCs may lead to increased demand as more people start to commute using their private vehicles. Therefore, further development and investment in the public transportation sector, as well as the prioritization of collective transportation systems and carbon-free means of transport, are equally important for a successful transition into future urban mobility.

7.1 Future Work

Future work could comprise a progressive flexibilization of the studied action space definition. Specifically, it is interesting to extend the current action space definition by giving the RL agent(s) the possibility to, also, dynamically adjust the cycle length. Such additional degree of flexibility could allow RL methods to learn even more efficient strategies to TLC, outperforming the non-periodic baselines, while still preserving advantageous properties such as the improved interpretability of the resulting policies. However, with such change, there is no longer a guarantee that a certain degree of synchronization exists among the agents. Therefore, it might be important to include the study of explicit coordination mechanisms. Similarly to the present work, it is hard to infer from the literature which coordination mechanisms are the most suitable under the TLC problem, thus, a comparison between different approaches should be carried on.

Another important future line of research is related to the performance assessment of the developed RL-based TSCs under unexpected events that cause abrupt disruptions in the traffic flow, such as accidents or lanes shutdown, since it is important for TSCs to reliably perform in highly dynamic urban environments, being able to adapt to unpredictable changes in traffic patterns. Furthermore, the robustness of the different RL algorithms should be tested with respect to state-related partial observability issues, such as noisy or missing sensory information.

Bibliography

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[2] Y. Wang, D. Zhang, Y. Liu, B. Dai, and L. Lee, "Enhancing transportation systems via deep learning: A survey," *Transportation Research Part C: Emerging Technologies*, vol. 99, pp. 144 – 163, 2019.

[3] Y. Wang, X. Yang, H. Liang, and Y. Liu, "A review of the self-adaptive traffic signal control system based on future traffic environment," *Journal of Advanced Transportation*, vol. 2018, pp. 1–12, 2018.

[4] H. Wei, G. Zheng, V. Gayah, and Z. Li, "A survey on traffic signal control methods," *CoRR*, vol. abs/1904.08117, 2019.

[5] P. Koonce, L. Rodegerdts, K. Lee, S. Quayle, S. Beaird, C. Braud, J. Bonneson, P. Tarnoff, and T. Urbanik, "Traffic signal timing manual," U.S. Department of Transportation, Tech. Rep. FHWA-HOP-08-024, 2008.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[8] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2016.

[9] B. Abdulhai and L. Kattan, "Reinforcement learning: Introduction to theory and potential for transport applications," *Canadian Journal of Civil Engineering*, vol. 30, no. 6, pp. 981–991, 2003.

[10] W. Genders and S. Razavi, "An open-source framework for adaptive traffic signal control," *CoRR*, vol. abs/1909.00395, 2019.

[11] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, “Design of reinforcement learning parameters for seamless application of adaptive traffic signal control,” *Journal of Intelligent Transportation Systems*, vol. 18, no. 3, pp. 227–245, 2014.

[12] M. Aslani, S. Seipel, M. Mesgari, and M. Wiering, “Traffic signal optimization through discrete and continuous reinforcement learning with robustness analysis in downtown Tehran,” *Advanced Engineering Informatics*, vol. 38, pp. 639 – 655, 2018.

[13] A. Stevanovic, “Adaptive traffic control systems: Domestic and foreign state of practice,” Transportation Research Board and National Academies of Sciences, Engineering and Medicine, Tech. Rep. 20-05/Topic 40-03, 2010.

[14] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–44, 2015.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[16] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proc. 14th International Conference Artificial Intelligence and Statistics*, vol. 15, 2011, pp. 315–323.

[17] D. Rumelhart, G. Hinton, and R. Williams, “Learning representations by back-propagating errors,” in *Neurocomputing: Foundations of Research*. MIT Press, 1988, pp. 696–699.

[18] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2015.

[19] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[20] C. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[21] H. Hasselt, “Double Q-learning,” in *Advances in Neural Information Processing Systems*, vol. 23, 2010, pp. 2613–2621.

[22] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *CoRR*, vol. abs/1511.05952, 2016.

[23] Z. Wang, N. de Freitas, and M. Lanctot, “Dueling network architectures for deep reinforcement learning,” *CoRR*, vol. abs/1511.06581, 2015.

[24] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” *CoRR*, vol. abs/1710.02298, 2017.

[25] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proc. of Machine Learning Research*, vol. 32, no. 1, 2014, pp. 387–395.

[26] W. Genders and S. N. Razavi, “Using a deep reinforcement learning agent for traffic signal control,” *CoRR*, vol. abs/1611.01142, 2016.

[27] S. Touhbi, M. Babram, T. Nguyen-Huu, N. Marilleau, M. Hbid, C. Cambier, and S. Stinckwich, “Adaptive traffic signal control : Exploring reward definition for reinforcement learning,” *Procedia Computer Science*, vol. 109, pp. 513 – 520, 2017.

[28] M. Aslani, M. 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.

[29] M. Eom and B.-I. Kim, “The traffic signal control problem for intersections: a review,” *European Transport Research Review*, vol. 12, no. 1, p. 50, 2020.

[30] F. Webster, “Traffic signal settings,” British road res. Lab., Tech. Rep. 39, 1958.

[31] J. Little, M. Kelson, and N. Gartner, “MAXBAND : A versatile program for setting signals on arteries and triangular networks,” Massachusetts Institute of Technology, Tech. Rep. 1185-81., 1981.

[32] P. Varaiya, “The max-pressure controller for arbitrary networks of signalized intersections,” in *Advances in Dynamic Network Modeling in Complex Transportation Systems*. Springer, 2013, pp. 27–66.

[33] P. Lowrie, “Scats, sydney co-ordinated adaptive traffic system: A traffic responsive method of controlling urban traffic,” *Roads and Traffic Authority NSW, Darlinghurst*, 1990.

[34] R. Bretherton, “Scoot urban traffic control system—philosophy and evaluation,” in *IFAC Proc. Volumes*, vol. 23, no. 2, 1990, pp. 237 – 239.

[35] P. Mirchandani and L. Head, “A real-time traffic signal control system: Architecture, algorithms, and analysis,” *Transportation Research Part C: Emerging Technologies*, vol. 9, pp. 415–432, 2001.

[36] F. Martinez, C. Toh, J.-C. Cano, C. Calafate, and P. Manzoni, “A survey and comparative study of simulators for vehicular ad hoc networks,” *Wireless Communications and Mobile Computing*, vol. 11, no. 7, pp. 813–828, 2011.

[37] R. Roess, E. Prassas, and W. McShane, *Traffic engineering*, 4th ed. Prentice Hall, 2011.

[38] B. Gokulan and D. Srinivasan, “Distributed geometric fuzzy multiagent urban traffic signal control,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 714–727, 2010.

[39] Y. Bi, X. Lu, D. Srinivasan, Z. Sun, and Z. Sun, "Optimal type-2 fuzzy system for arterial traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 9, pp. 3009–3027, 2018.

[40] M. Shirvani and H. Maleki, "Maximum green time settings for traffic actuated signal control at isolated intersections using fuzzy logic," *International Journal of Fuzzy Systems*, vol. 19, no. 1, pp. 247–256, 2017.

[41] W. Yang, L. Zhang, Z. He, and L. Zhuang, "Optimized two-stage fuzzy control for urban traffic signals at isolated intersection and paramics simulation," in *15th International IEEE Conference on Intelligent Transportation Systems*, 2012, pp. 391–396.

[42] M. Khooban and A. Liaghat, "A time-varying strategy for urban traffic network control: A fuzzy logic control based on an improved black hole algorithm," *International Journal of Bio-Inspired Computation*, vol. 10, p. 33, 2017.

[43] B. Park, C. Messer, and T. II, "Traffic signal optimization program for oversaturated conditions: genetic algorithm approach," *Transportation Research Record*, vol. 1683, pp. 133–142, 1999.

[44] K. Teo, W. Kow, and Y. Chin, "Optimization of traffic flow within an urban traffic light intersection with genetic algorithm," in *2nd Int. Conf. on Computational Intelligence, Modelling and Simulation*, 2010, pp. 172–177.

[45] Y. Zhang and Y. Zhou, "Distributed coordination control of traffic network flow using adaptive genetic algorithm based on cloud computing," *Journal of Network and Computer Applications*, vol. 119, pp. 110 – 120, 2018.

[46] H. Asadi, R. Moghaddam, N. Pour, and E. Najafi, "A new nondominated sorting genetic algorithm based to the regression line for fuzzy traffic signal optimization problem," *Scientia Iranica*, vol. 25, no. 3, pp. 1712–1723, 2018.

[47] M. Simona, L. Dupont, and M. Camargo, "Multi-objective traffic signal optimization using 3d mesoscopic simulation and evolutionary algorithms," *Simulation Modelling Practice and Theory*, vol. 86, pp. 120 – 138, 2018.

[48] D. Teodorovic, "Swarm intelligence systems for transportation engineering: Principles and applications," *Transportation Research Part C: Emerging Technologies*, vol. 16, pp. 651–667, 2008.

[49] J. Garcia-Nieto, E. Alba, and A. Olivera, "Swarm intelligence for traffic light scheduling: Application to real urban areas," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 2, pp. 274–283, 2012.

[50] K.-H. Chao, R.-H. Lee, and M.-H. Wang, “An intelligent traffic light control based on extension neural network,” in *Knowledge-Based Intelligent Information and Engineering Systems*, 2008, pp. 17–24.

[51] G. Shen and X. Kong, “Study on road network traffic coordination control technique with bus priority,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 39, pp. 343–351, 2009.

[52] L. Cao, B. Hu, X. Dong, G. Xiong, F. Zhu, Z. Shen, D. Shen, and Y. Liu, “Two intersections traffic signal control method based on adhdp,” in *IEEE International Conference on Vehicular Electronics and Safety*, 2016, pp. 1–5.

[53] L. Jácome, L. Benavides, D. Jara, G. Riofrio, F. Alvarado, and M. Pesantez, “A survey on intelligent traffic lights,” in *IEEE International Conference on Automation*, 2018, pp. 1–6.

[54] K. Yau, J. Qadir, H. Khoo, M. Ling, and P. Komisarczuk, “A survey on reinforcement learning models and algorithms for traffic signal control,” *ACM Computing Surveys*, vol. 50, pp. 34:1–34:38, 2017.

[55] N. Casas, “Deep deterministic policy gradient for urban traffic light control,” *CoRR*, vol. abs/1703.09035, 2017.

[56] H. Wei, H. Yao, G. Zheng, and Z. Li, “Intellilight: A reinforcement learning approach for intelligent traffic light control,” in *Proc. of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 2496–2505.

[57] F. Rodrigues and C. L. Azevedo, “Towards robust deep reinforcement learning for traffic signal control: Demand surges, incidents and sensor failures,” in *IEEE Intelligent Transportation Systems Conference*, 2019, p. 3559–3566.

[58] W. Genders and S. Razavi, “Evaluating reinforcement learning state representations for adaptive traffic signal control,” *Procedia Computer Science*, vol. 130, no. C, pp. 26–33, 2018.

[59] T. Nishi, K. Otaki, K. Hayakawa, and T. Yoshimura, “Traffic signal control based on reinforcement learning with graph convolutional neural nets,” in *21st International Conference on Intelligent Transportation Systems*, 2018, pp. 877–883.

[60] L. Nunes, D. de Oliveira, A. Bazzan, B. da Silva, E. Basso, R. Rossetti, E. Oliveira, R. da Silva, and L. Lamb, “Reinforcement learning-based control of traffic lights in non-stationary environments: A case study in a microscopic simulator,” in *CEUR Workshop Proc.*, 2006, pp. 31–42.

[61] H. Wei, C. Chen, G. Zheng, K. Wu, V. Gayah, K. Xu, and Z. Li, “Presslight: Learning max pressure control to coordinate traffic signals in arterial network,” in *Proc. of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, p. 1290–1298.

[62] M. Wiering, “Multi-agent reinforcement learning for traffic light control,” in *Proc. of the 17th International Conference on Machine Learning*, 2000, pp. 1151–1158.

[63] M. Wiering, J. van Veenen, J. Vreeken, and A. Koopman, “Intelligent traffic light control,” Dept. of Information and Computing Sciences. Utrecht University, Tech. Rep., 2004.

[64] D. Houli, L. Zhiheng, and Z. Yi, “Multiobjective reinforcement learning for traffic signal control using vehicular ad hoc network,” *EURASIP Journal on Advances in Signal Processing*, no. 1, p. 724035, 2010.

[65] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis, “Multiagent reinforcement learning for urban traffic control using coordination graphs,” in *Machine Learning and Knowledge Discovery in Databases*, 2008, pp. 656–671.

[66] B. Abdulhai, R. Pringle, and G. Karakoulas, “Reinforcement learning for true adaptive traffic signal control,” *Journal of Transportation Engineering*, vol. 129, no. 3, pp. 278–285, 2003.

[67] T. Thorpe and C. Anderson, “Traffic light control using SARSA with three state representations,” IBM Corporation, Tech. Rep., 1996.

[68] X. Zhou, F. Zhu, Q. Liu, Y. Fu, and W. Huang, “A sarsa(λ)-based control model for real-time traffic light coordination,” *The Scientific World Journal*, vol. 2014, p. 759097, 2014.

[69] M. Aslani, S. Seipel, and M. Wiering, “Continuous residual reinforcement learning for traffic signal control optimization,” *Canadian Journal of Civil Engineering*, vol. 45, no. 8, pp. 690–702, 2018.

[70] M. Abdoos, N. Mozayani, and A. Bazzan, “Hierarchical control of traffic signals using Q-learning with tile coding,” *Applied Intelligence*, vol. 40, no. 2, pp. 201–213, 2014.

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

[72] C. Choe, S. Baek, B. Woon, and S. Kong, “Deep Q learning with LSTM for traffic light control,” in *24th Asia-Pacific Conference on Communications*, 2018, pp. 331–336.

[73] G. Zheng, Y. Xiong, X. Zang, J. Feng, H. Wei, H. Zhang, Y. Li, K. Xu, and Z. Li, “Learning phase competition for traffic signal control,” *CoRR*, vol. abs/1905.04722, 2019.

[74] J. Zeng, J. Hu, and Y. Zhang, “Adaptive traffic signal control with deep recurrent Q-learning,” in *IEEE Intelligent Vehicles Symposium*, 2018, pp. 1215–1220.

[75] E. van der Pol and F. Oliehoek, “Coordinated deep reinforcement learners for traffic light control,” in *NIPS Workshop on Learning, Inference and Control of Multi-Agent Systems*, 2016.

[76] S. Shabestary and B. Abdulhai, “Deep learning vs. discrete reinforcement learning for adaptive traffic signal control,” in *21st International Conference on Intelligent Transportation Systems*, 2018, pp. 286–293.

[77] L. Prashanth and S. Bhatnagar, “Reinforcement learning with average cost for adaptive control of traffic lights at intersections,” in *14th International IEEE Conference on Intelligent Transportation Systems*, 2011, pp. 1640–1645.

[78] Y. Lin, X. Dai, L. Li, and F. Wang, “An efficient deep reinforcement learning model for urban traffic control,” *CoRR*, vol. abs/1808.01876, 2018.

[79] J.-T. Girault, V. Gayah, S. Guler, and M. Menendez, “Exploratory analysis of signal coordination impacts on macroscopic fundamental diagram,” *Journal of the Transportation Research Board*, vol. 2560, pp. 36–46, 2016.

[80] M. Behrisch, L. Bieker-Walz, J. Erdmann, and D. Krajzewicz, “Sumo – simulation of urban mobility: An overview,” in *Proc. of SIMUL*, 2011.

[81] J. Casas, J. Ferrer, D. Garcia, J. Perarnau, and A. Torday, “Traffic simulation with aimsun,” in *Fundamentals of Traffic Simulation*. Springer, 2010, pp. 173–232.

[82] G. Cameron and G. Duncan, “Paramics—parallel microscopic simulation of road traffic,” *The Journal of Supercomputing*, vol. 10, no. 1, pp. 25–53, 1996.

[83] M. Saidallah, A. El Fergougui, and A. El Alaoui, “A comparative study of urban road traffic simulators,” in *MATEC Web of Conferences*, vol. 81, 2016, p. 05002.

[84] N. Ghariani, S. Elkosantini, S. Darmoul, and L. Ben Said, “A survey of simulation platforms for the assessment of public transport control systems,” in *International Conference on Advanced Logistics and Transport*, 2014, pp. 85–90.

[85] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. Bayen, “Flow: Architecture and benchmarking for reinforcement learning in traffic control,” *CoRR*, vol. abs/1710.05465, 2017.

[86] L. Codeca, R. Frank, S. Faye, and T. Engel, “Luxembourg sumo traffic scenario: Traffic demand evaluation,” *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 2, pp. 52–63, 2017.

[87] X. Lu, “Deliver a set of tools for resolving bad inductive loops and correcting bad data,” California PATH, ITS, University of California, Berkeley, Tech. Rep., 2012.

[88] C. Daganzo, “Urban gridlock: Macroscopic modeling and mitigation approaches,” *Transportation Research Part B: Methodological*, vol. 41, no. 1, pp. 49–62, 2007.

[89] S. Whiteson, B. Tanner, M. E. Taylor, and P. Stone, “Protecting against evaluation overfitting in empirical reinforcement learning,” in *Proc. IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2011, pp. 120–127.

[90] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, “Reproducibility of benchmarked deep reinforcement learning tasks for continuous control,” *CoRR*, vol. abs/1708.04133, 2017.

[91] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” *CoRR*, vol. abs/1709.06560, 2019.

[92] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, “A study on overfitting in deep reinforcement learning,” *CoRR*, vol. abs/1804.06893, 2018.

[93] S. Shapiro and M. Wilk, “An analysis of variance test for normality,” *Biometrika*, vol. 52, no. 3-4, pp. 591–611, 1965.

[94] I. Olkin, *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*. Stanford University Press, 1960.

[95] M. Abdoos, N. Mozayani, and A. Bazzan, “Traffic light control in non-stationary environments based on multi agent q-learning,” in *14th International IEEE Conference on Intelligent Transportation Systems*, 2011, pp. 1580–1585.

[96] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, A. Cowie, Z. Wang, B. Piot, and N. de Freitas, “Acme: A research framework for distributed reinforcement learning,” 2020.

[97] A. Cassirer, G. Barth-Maron, T. Sottiaux, M. Kroiss, and E. Brevdo, “Reverb: An efficient data storage and transport system for ml research,” 2020.

[98] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015.

Algorithms hyperparameters

Tables below display the hyperparameters used for each of the RL algorithms. Commonly to all RL methods, the number of training steps is set to 50 000, the discount factor (γ) to 0.98, and the reward rescale coefficient to 0.01. Tables A.1, A.2, and A.3 display, respectively, the hyperparameters for the DQN, Q-learning, and DDPG algorithms.

Table A.1: Complete list of the hyperparameters used with the DQN algorithm, including double Q-learning, prioritized replay and N-step transitions. In order to properly balance exploration and exploitation, an ϵ-greedy policy is used. The neural network consists of a three-layer feed-forward ANN with an [8,16,8] duelling network architecture (torso layers=[8, 16], dueling head layers=[8]).

Hyperparameter	Value(s)
Optimizer	Adam
Learning rate	1e-3
N-step	5
Replay buffer min. size	5 000
Replay buffer max. size	50 000
Replay buffer batch size	128
Target network update period	100
Prioritized replay (ω)	0.9
Prioritized replay (β)	0.6
Exploration initial ϵ	1.0
Exploration final ϵ	0.01
Exploration ϵ schedule	45 000

Table A.2: Complete list of the hyperparameters used with the Q-learning algorithm. The observed trajectories are stored in a replay buffer and, at each decision step, multiple Q-learning updates are performed using transitions sampled from the replay memory. An ϵ-greedy policy is used, with $\epsilon_t = 1/(1+N_t(s))$, where $N_t(s)$ denotes the number of times state s was visited up to timestep t.

Hyperparameter	Value(s)
Learning rate (α)	0.05
Discretization bins (percentiles)	[0.1,0.3,0.5,0.7,0.9]
Exploration strategy	ϵ-greedy
Replay buffer min. size	2 000
Replay buffer max. size	20 000
Replay buffer batch size	128

Table A.3: Complete list of the hyperparameters used with the DDPG algorithm, including N-step transitions. The critic consists of a fully connected neural network (with architecture [16,32,16]), which final layer maps the hidden activations into a scalar representing the Q-value for the inputted (state, action) pair. The actor consists of a fully connected neural network (with architecture [16,32,16]), which final layer is a softmax function mapping the hidden activations into the allocations for each of the phases. In order to ensure adequate exploration, gaussian noise $\mathcal{N} \sim N(0, \sigma^2)$ is introduced to the behaviour network. Figure 5.6 displays an illustration of the actor network architecture.

Critic optimizer	Adam
Actor optimizer	Adam
Critic learning rate	1e-3
Actor learning rate	1e-4
Replay buffer min. size	5 000
Replay buffer max. size	50 000
Replay buffer batch size	128
Target network update period	100
Exploration initial σ^2	0.2
Exploration final σ^2	0.01
Exploration σ^2 schedule	45 000
Gradients clipping	True

B

Complete experimental results

Table B.1: Complete set of performance metrics for the *intersection* network under the *high constant* demand, defined with the mapping $\{d(1) = 0.10, d(2) = 0.22\}$. For trainable algorithms, reported performance metrics are calculated using 30 training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of 30 evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Static	N/A	N/A	0.47	(8.3, 10.2)	(25.0, 12.5)	24.9
Webster	N/A	N/A	0.49	(8.5, 10.0)	(25.4, 12.4)	25.3
Max-pressure	N/A	N/A	0.48	(6.0, 6.8)	(23.4, 9.0)	23.3
Actuated	N/A	N/A	0.46	(8.0, 10.5)	(24.9, 12.8)	24.7
QL (Random)	N/A	N/A	0.56	(11.4, 13.2)	(29.1, 16.5)	28.6
QL	*Min. speed delta*	-148.97	0.50	(9.2, 11.1)	(26.3, 13.7)	26.0
QL	*Min. delay*	-107.06	0.46	(8.4, 10.6)	(25.0, 12.8)	24.9
QL	*Max. delay red.*	-0.08	0.50	(9.4, 11.4)	(26.5, 13.9)	26.3
QL	*Min. waiting time*	-99.25	0.46	(8.4, 10.6)	(25.0, 12.8)	24.9
QL	*Min. queue*	-138.18	0.47	(8.7, 11.1)	(25.4, 13.3)	25.0
QL	*Max. queue red.*	-0.50	0.49	(9.4, 11.8)	(26.3, 14.2)	26.0
QL	*Min. pressure*	-177.55	0.46	(8.4, 10.9)	(25.0, 13.0)	24.9
DQN (Random)	N/A	N/A	0.56	(11.4, 13.2)	(29.1, 16.5)	28.6
DQN	*Min. speed delta*	-138.34	0.46	(8.5, 10.9)	(25.1, 13.0)	24.9
DQN	*Min. delay*	-107.83	0.47	(8.4, 10.7)	(25.1, 12.9)	24.8
DQN	*Max. delay red.*	-0.08	0.48	(9.4, 12.7)	(26.1, 14.9)	25.0
DQN	*Min. waiting time*	-98.70	0.47	(8.3, 10.3)	(25.0, 12.6)	24.9
DQN	*Min. queue*	-137.10	0.47	(8.3, 10.3)	(25.0, 12.6)	24.8
DQN	*Max. queue red.*	-0.49	0.48	(8.5, 10.6)	(25.3, 12.8)	24.9
DQN	*Min. pressure*	-178.81	0.46	(8.6, 11.1)	(25.2, 13.2)	24.8
DDPG (Random)	N/A	N/A	0.61	(13.3, 16.3)	(31.3, 20.2)	30.6
DDPG	*Min. speed delta*	-140.96	0.48	(8.6, 10.7)	(25.4, 13.0)	24.9
DDPG	*Min. delay*	-107.31	0.48	(8.3, 10.1)	(25.1, 12.5)	24.8
DDPG	*Max. delay red.*	-0.07	0.48	(8.4, 10.1)	(25.3, 12.5)	24.9
DDPG	*Min. waiting time*	-99.51	0.48	(8.3, 10.1)	(25.2, 12.5)	24.9
DDPG	*Min. queue*	-137.33	0.47	(8.5, 10.8)	(25.2, 13.0)	24.9
DDPG	*Max. queue red.*	-0.50	0.47	(8.7, 11.1)	(25.4, 13.2)	25.1
DDPG	*Min. pressure*	-177.72	0.47	(8.3, 10.2)	(25.1, 12.5)	24.8

Table B.2: Complete set of performance metrics for the *intersection* network under the *low constant* demand, defined with the mapping $\{d(1) = 0.05, d(2) = 0.10\}$. For trainable algorithms, reported performance metrics are calculated using 30 training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of 30 evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Static	N/A	N/A	0.41	(7.2, 10.2)	(22.4, 12.4)	22.2
Webster	N/A	N/A	0.45	(7.7, 9.7)	(23.2, 12.3)	23.1
Max-pressure	N/A	N/A	0.39	(4.7, 6.2)	(20.2, 8.8)	20.1
Actuated	N/A	N/A	0.41	(5.1, 6.6)	(20.8, 9.2)	20.7
QL (Random)	N/A	N/A	0.46	(8.6, 11.1)	(24.4, 13.8)	24.2
QL	*Min. speed delta*	-58.73	0.45	(8.1, 10.6)	(23.7, 13.2)	23.5
QL	*Min. delay*	-42.76	0.42	(7.4, 10.4)	(22.6, 12.6)	22.4
QL	*Max. delay red.*	-0.03	0.45	(8.1, 10.7)	(23.7, 13.3)	23.6
QL	*Min. waiting time*	-39.97	0.42	(7.4, 10.3)	(22.6, 12.6)	22.4
QL	*Min. queue*	-75.88	0.46	(8.7, 11.3)	(24.4, 14.0)	23.1
QL	*Max. queue red.*	-0.15	0.46	(8.3, 10.8)	(24.0, 13.5)	23.7
QL	*Min. pressure*	-73.10	0.42	(7.4, 10.5)	(22.6, 12.7)	22.4
DQN (Random)	N/A	N/A	0.46	(8.6, 11.1)	(24.4, 13.8)	24.2
DQN	*Min. speed delta*	-53.48	0.41	(7.2, 10.2)	(22.4, 12.4)	22.3
DQN	*Min. delay*	-42.05	0.41	(7.2, 10.2)	(22.4, 12.4)	22.3
DQN	*Max. delay red.*	-0.03	0.45	(8.2, 10.9)	(23.8, 13.5)	22.4
DQN	*Min. waiting time*	-39.29	0.41	(7.2, 10.2)	(22.4, 12.4)	22.3
DQN	*Min. queue*	-75.69	0.47	(8.7, 10.9)	(24.5, 13.7)	22.5
DQN	*Max. queue red.*	-0.15	0.47	(8.7, 11.0)	(24.4, 13.8)	22.4
DQN	*Min. pressure*	-72.12	0.41	(7.2, 10.2)	(22.4, 12.4)	22.3
DDPG (Random)	N/A	N/A	0.48	(9.4, 12.4)	(25.2, 15.2)	24.9
DDPG	*Min. speed delta*	-53.54	0.41	(7.3, 10.5)	(22.4, 12.6)	22.3
DDPG	*Min. delay*	-42.78	0.42	(7.3, 10.1)	(22.6, 12.4)	22.4
DDPG	*Max. delay red.*	-0.03	0.41	(7.3, 10.4)	(22.4, 12.6)	22.3
DDPG	*Min. waiting time*	-40.06	0.42	(7.4, 10.1)	(22.6, 12.4)	22.3
DDPG	*Min. queue*	-75.74	0.46	(8.5, 10.9)	(24.3, 13.6)	22.4
DDPG	*Max. queue red.*	-0.14	0.48	(9.3, 11.6)	(25.3, 14.5)	23.2
DDPG	*Min. pressure*	-72.88	0.42	(7.3, 10.1)	(22.6, 12.4)	22.3

Table B.3: Complete set of performance metrics for the *intersection* network under the *variable* demand (+ denotes the inclusion of the *time variable*). The base demand is defined with the mapping $\{d(1) = 0.10, d(2) = 0.22\}$. For trainable algorithms, reported performance metrics are calculated using 30 training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of 30 evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value. The free-flow period corresponds to the 22h00-23h00 interval, and the congested period to the 08h00-09h00 interval.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time free-flow (s)	Travel time congested (s)	Travel time (s)	Top-3 travel time (s)
Webster	N/A	N/A	0.45	(7.7, 9.7)	(22.1, 12.3)	(25.1, 12.4)	(23.7, 12.3)	23.5
Max-pressure	N/A	N/A	0.41	(4.9, 6.3)	(18.2, 8.5)	(23.3, 9.0)	(21.0, 8.9)	20.9
Actuated	N/A	N/A	0.43	(5.9, 7.7)	(19.8, 8.9)	(24.8, 12.5)	(22.0, 10.2)	21.9
QL (Random)	N/A	N/A	0.49	(9.4, 11.8)	(23.4, 13.9)	(28.9, 16.4)	(25.8, 14.7)	25.5
QL	*Min. speed delta*	-63.10	0.46	(8.5, 10.9)	(22.7, 13.3)	(26.5, 14.1)	(24.6, 13.5)	24.3
QL	*Min. delay*	-46.34	0.43	(7.9, 10.9)	(21.6, 12.6)	(25.8, 14.4)	(23.6, 13.2)	23.1
QL	*Max. delay red.*	-0.01	0.46	(8.3, 10.7)	(22.7, 13.4)	(25.7, 13.3)	(24.2, 13.3)	24.0
QL	*Min. waiting time*	-43.11	0.43	(7.8, 10.8)	(21.6, 12.6)	(25.8, 14.3)	(23.5, 13.1)	23.2
QL	*Min. queue*	-74.19	0.45	(8.5, 11.5)	(22.2, 13.0)	(26.7, 15.5)	(24.4, 14.0)	23.5
QL	*Max. queue red.*	-0.04	0.46	(8.5, 11.0)	(23.9, 14.2)	(26.2, 14.0)	(24.6, 13.6)	24.2
QL	*Min. pressure*	-78.83	0.44	(8.1, 11.4)	(21.6, 12.6)	(26.5, 15.7)	(23.8, 13.7)	23.2
DQN (Random)	N/A	N/A	0.49	(9.4, 11.8)	(23.4, 13.9)	(28.9, 16.4)	(25.8, 14.7)	25.5
DQN	*Min. speed delta*	-58.73	0.44	(7.8, 10.5)	(22.3, 13.0)	(25.4, 13.5)	(23.6, 12.9)	23.1
DQN	*Min. speed delta +*	-58.02	0.43	(7.7, 10.5)	(21.5, 12.3)	(25.6, 13.7)	(23.4, 12.8)	23.1
DQN	*Min. delay*	-46.73	0.44	(7.9, 10.5)	(22.8, 13.3)	(25.3, 13.2)	(23.7, 12.9)	23.1
DQN	*Min. delay +*	-45.88	0.44	(7.7, 10.5)	(21.7, 12.4)	(25.4, 13.4)	(23.5, 12.8)	23.1
DQN	*Max. delay red.*	-0.01	0.54	(11.7, 16.9)	(23.5, 13.9)	(34.0, 26.4)	(28.5, 20.8)	23.2
DQN	*Max. delay red. +*	-0.01	0.46	(9.0, 12.5)	(22.5, 13.2)	(28.3, 18.0)	(25.0, 15.0)	23.1
DQN	*Min. waiting time*	-43.33	0.44	(7.8, 10.3)	(22.3, 12.9)	(25.3, 13.1)	(23.7, 12.7)	23.2
DQN	*Min. waiting time +*	-42.42	0.43	(7.7, 10.4)	(21.6, 12.3)	(25.5, 13.5)	(23.4, 12.7)	23.1
DQN	*Min. queue*	-74.21	0.47	(8.8, 11.5)	(23.2, 13.6)	(27.6, 16.3)	(24.9, 14.2)	23.2
DQN	*Min. queue +*	-73.46	0.45	(8.0, 10.4)	(22.1, 12.6)	(26.2, 14.1)	(24.0, 12.9)	23.1
DQN	*Max. queue red.*	-0.05	0.46	(8.7, 11.8)	(22.5, 13.0)	(27.3, 16.2)	(24.8, 14.3)	23.1
DQN	*Max. queue red. +*	-0.05	0.49	(9.8, 14.4)	(22.6, 13.1)	(29.6, 21.5)	(26.0, 17.7)	23.1
DQN	*Min. pressure +*	-77.05	0.43	(7.7, 10.5)	(21.8, 12.5)	(25.5, 13.7)	(23.4, 12.8)	23.1
DDPG (Random)	N/A	N/A	0.52	(10.7, 13.9)	(23.6, 14.3)	(31.0, 20.2)	(27.3, 17.2)	26.8
DDPG	*Min. speed delta*	-59.94	0.44	(8.1, 11.0)	(22.2, 13.0)	(26.4, 14.8)	(23.8, 13.3)	23.3
DDPG	*Min. speed delta +*	-58.39	0.42	(7.9, 11.2)	(21.5, 12.4)	(26.4, 15.3)	(23.4, 13.3)	23.2
DDPG	*Min. delay*	-46.52	0.44	(7.8, 10.3)	(22.1, 12.6)	(25.5, 13.3)	(23.7, 12.7)	23.2
DDPG	*Min. delay +*	-45.55	0.43	(7.7, 10.4)	(21.7, 12.3)	(25.5, 13.4)	(23.4, 12.7)	23.1
DDPG	*Max. delay red. +*	-0.01	0.45	(8.2, 10.8)	(23.0, 13.5)	(26.0, 14.2)	(24.0, 13.3)	23.3
DDPG	*Min. waiting time*	-43.52	0.45	(7.9, 10.3)	(22.2, 12.6)	(25.5, 13.4)	(23.7, 12.7)	23.1
DDPG	*Min. waiting time +*	-42.51	0.44	(7.7, 10.4)	(21.8, 12.3)	(25.5, 13.4)	(23.4, 12.7)	23.1
DDPG	*Min. queue*	-73.39	0.45	(8.2, 10.9)	(23.1, 13.3)	(26.2, 14.8)	(24.1, 13.3)	23.4
DDPG	*Max. queue red.*	-0.05	0.47	(8.7, 11.2)	(23.3, 13.9)	(26.7, 14.7)	(24.8, 13.8)	23.3
DDPG	*Max. queue red. +*	-0.05	0.45	(8.3, 11.2)	(23.4, 13.9)	(26.4, 14.8)	(24.2, 13.6)	23.3
DDPG	*Min. pressure +*	-76.97	0.43	(7.7, 10.6)	(21.6, 12.3)	(25.5, 13.9)	(23.4, 12.8)	23.0

Table B.4: Complete set of performance metrics for the *intersection* network under the *cyclical* demand. The base demand is defined with the mapping $\{d(1) = 0.08,\ d(2) = 0.15\}$. For trainable algorithms, reported performance metrics are calculated using 30 training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of 30 evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Webster	N/A	N/A	0.48	(8.4, 10.0)	(24.8, 12.6)	24.6
Adaptive-Webster	N/A	N/A	0.46	(7.9, 10.1)	(24.1, 12.8)	24.0
Max-pressure	N/A	N/A	0.44	(5.5, 6.7)	(21.9, 9.1)	21.8
Actuated	N/A	N/A	0.44	(6.5, 8.7)	(22.8, 11.2)	22.6
QL (Random)	N/A	N/A	0.52	(10.2, 12.5)	(26.9, 15.5)	26.6
QL	*Min. speed delta*	-101.34	0.48	(8.9, 11.2)	(25.3, 13.9)	25.1
QL	*Min. delay*	-75.54	0.45	(8.4, 11.1)	(24.5, 13.5)	24.2
QL	*Max. delay red.*	-0.06	0.48	(8.9, 11.1)	(25.3, 13.7)	25.1
QL	*Min. waiting time*	-69.98	0.45	(8.3, 11.0)	(24.4, 13.4)	24.1
QL	*Min. queue*	-108.43	0.48	(9.3, 11.8)	(25.7, 14.6)	24.9
QL	*Max. queue red.*	-0.39	0.48	(8.9, 11.2)	(25.2, 13.9)	24.9
QL	*Min. pressure*	-123.85	0.45	(8.4, 11.3)	(24.4, 13.7)	24.0
DQN (Random)	N/A	N/A	0.52	(10.2, 12.5)	(26.9, 15.5)	26.6
DQN	*Min. speed delta*	-93.01	0.45	(8.1, 10.7)	(24.0, 13.1)	23.8
DQN	*Min. delay*	-73.94	0.45	(8.2, 10.6)	(24.2, 12.9)	24.0
DQN	*Max. delay red.*	-0.06	0.49	(9.6, 12.2)	(26.1, 14.9)	24.3
DQN	*Min. waiting time*	-68.99	0.46	(8.2, 10.5)	(24.3, 12.9)	23.9
DQN	*Min. queue*	-107.48	0.47	(8.9, 11.2)	(25.2, 13.7)	24.2
DQN	*Max. queue red.*	-0.40	0.49	(9.2, 11.7)	(25.7, 14.4)	24.3
DQN	*Min. pressure*	-123.49	0.45	(8.4, 11.2)	(24.4, 13.6)	23.8
DDPG (Random)	N/A	N/A	0.55	(11.7, 14.9)	(28.7, 18.5)	28.1
DDPG	*Min. speed delta*	-93.18	0.44	(8.1, 10.7)	(24.0, 13.0)	23.8
DDPG	*Min. delay*	-73.24	0.46	(8.0, 10.3)	(24.1, 12.7)	23.8
DDPG	*Min. waiting time*	-68.15	0.46	(8.0, 10.2)	(24.1, 12.7)	23.8
DDPG	*Max. queue red.*	-0.39	0.48	(8.8, 11.2)	(25.2, 13.8)	24.2

Table B.5: Complete set of performance metrics for the *arterial* network under the *constant* demand, defined with the mapping $\{d(1) = 0.05, d(2) = 0.10\}$. For trainable algorithms, reported performance metrics are calculated using 15 training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of 15 evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Webster	N/A	N/A	0.60	(10.8, 11.2)	(34.4, 15.1)	34.3
Max-pressure	N/A	N/A	0.72	(8.6, 8.7)	(35.6, 16.9)	35.2
Actuated	N/A	N/A	0.81	(10.0, 10.1)	(37.9, 18.8)	37.7
DQN (Random)	N/A	N/A	0.73	(15.3, 16.1)	(40.8, 22.7)	40.5
DQN	*Min. speed delta*	-103.76	0.59	(10.7, 11.7)	(34.3, 14.9)	34.1
DQN	*Min. delay*	-81.14	0.59	(10.7, 11.7)	(34.3, 15.0)	34.1
DQN	*Min. waiting time*	-76.11	0.59	(10.7, 11.6)	(34.3, 15.1)	34.1
DQN	*Min. pressure*	-22.55	0.79	(14.8, 14.7)	(40.8, 20.4)	35.4
DDPG (Random)	N/A	N/A	0.82	(18.0, 19.3)	(44.5, 26.9)	44.1
DDPG	*Min. speed delta*	-105.27	0.59	(10.9, 11.7)	(34.6, 15.3)	34.4
DDPG	*Min. delay*	-82.70	0.60	(11.0, 11.5)	(34.7, 15.7)	34.4
DDPG	*Min. waiting time*	-77.54	0.60	(10.9, 11.5)	(34.7, 15.7)	34.4
DDPG	*Min. pressure*	-45.52	0.62	(11.8, 12.5)	(36.3, 17.2)	34.3

Table B.6: Complete set of performance metrics for the *arterial* network under the *variable* demand (+ denotes the inclusion of the *time variable*). The base demand is defined with the mapping $\{d(1) = 0.10, d(2) = 0.22\}$. For trainable algorithms, reported performance metrics are calculated using 15 training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of 15 evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value. The free-flow period corresponds to the 22h00-23h00 interval, and the congested period to the 08h00-09h00 interval.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time free-flow (s)	Travel time congested (s)	Travel time (s)	Top-3 travel time (s)
Webster	N/A	N/A	0.62	(11.3, 11.2)	(33.2, 15.1)	(38.6, 14.8)	(35.7, 14.9)	35.5
Max-pressure	N/A	N/A	0.75	(9.3, 9.1)	(32.9, 16.5)	(42.4, 18.7)	(37.5, 17.4)	37.1
Actuated	N/A	N/A	0.85	(11.3, 11.2)	(36.0, 17.7)	(44.8, 21.5)	(40.2, 19.7)	39.9
DQN (Random)	N/A	N/A	0.84	(18.4, 18.5)	(37.8, 20.9)	(56.4, 33.3)	(45.9, 26.7)	45.1
DQN	*Min. speed delta +*	-114.08	0.63	(11.7, 12.0)	(33.6, 15.5)	(41.0, 17.3)	(36.4, 15.8)	35.9
DQN	*Min. delay +*	-88.01	0.62	(11.6, 11.8)	(33.4, 15.4)	(40.2, 16.7)	(36.2, 15.7)	35.6
DQN	*Max. delay red. +*	-0.02	0.72	(15.3, 16.7)	(34.9, 17.2)	(51.4, 31.1)	(41.4, 22.8)	36.0
DQN	*Min. waiting time +*	-83.31	0.63	(11.7, 12.0)	(33.5, 15.5)	(40.8, 17.6)	(36.3, 15.9)	35.8
DQN	*Max. queue red. +*	-0.10	0.77	(17.1, 19.4)	(35.9, 19.0)	(53.6, 37.3)	(43.3, 27.7)	36.5
DQN	*Min. pressure +*	-33.65	0.81	(16.1, 17.1)	(35.9, 17.7)	(48.4, 24.3)	(43.0, 23.4)	37.0
DDPG (Random)	N/A	N/A	0.99	(22.6, 23.8)	(40.6, 24.2)	(65.8, 42.6)	(51.6, 33.6)	50.9
DDPG	*Min. speed delta +*	-114.84	0.62	(11.8, 12.4)	(33.5, 15.5)	(41.5, 18.3)	(36.6, 16.1)	36.0
DDPG	*Min. delay +*	-89.61	0.63	(11.8, 12.1)	(33.4, 15.7)	(42.2, 20.2)	(36.6, 16.6)	36.0
DDPG	*Min. waiting time +*	-82.69	0.62	(11.6, 11.9)	(33.4, 15.6)	(40.7, 17.6)	(36.4, 16.1)	35.9
DDPG	*Min. pressure +*	-32.92	0.72	(14.2, 14.1)	(34.5, 16.4)	(46.2, 22.6)	(40.6, 20.0)	37.4

Table B.7: Complete set of performance metrics for the *arterial* network under the *cyclical* demand. The base demand is defined with the mapping $\{d(1) = 0.05, d(2) = 0.10\}$. For trainable algorithms, reported performance metrics are calculated using 15 training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of 15 evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time (s)	Top-3 travel time (s)
Webster	N/A	N/A	0.66	(12.5, 11.5)	(37.4, 15.6)	37.3
Adaptive-Webster	N/A	N/A	0.64	(12.3, 12.5)	(37.1, 17.4)	36.9
Max-pressure	N/A	N/A	0.80	(10.3, 9.6)	(39.0, 17.7)	38.7
Actuated	N/A	N/A	0.84	(12.1, 12.3)	(41.0, 20.6)	40.7
DQN (Random)	N/A	N/A	0.92	(20.4, 20.0)	(48.9, 29.0)	48.1
DQN	*Min. speed delta*	-190.32	0.65	(12.4, 12.4)	(37.6, 16.6)	37.0
DQN	*Min. delay*	-148.12	0.66	(12.4, 12.0)	(37.6, 16.3)	37.2
DQN	*Min. waiting time*	-138.76	0.66	(12.4, 12.1)	(37.7, 16.4)	37.1
DQN	*Min. pressure*	-45.15	0.85	(17.4, 17.6)	(45.2, 25.0)	42.5
DDPG (Random)	N/A	N/A	0.86	(19.0, 20.1)	(46.1, 28.2)	45.8
DDPG	*Min. speed delta*	-188.19	0.65	(12.3, 11.9)	(37.4, 16.1)	36.9
DDPG	*Min. delay*	-149.86	0.66	(12.6, 12.2)	(37.8, 16.9)	37.0
DDPG	*Min. waiting time*	-138.79	0.66	(12.5, 12.0)	(37.6, 16.7)	37.2
DDPG	*Min. pressure*	-50.22	0.75	(15.5, 15.1)	(42.1, 22.0)	38.9

Table B.8: Complete set of performance metrics for the *grid* network under the *constant* demand, defined with the mapping $\{d(1) = 0.05, d(2) = 0.15, d(3) = 0.20\}$. For trainable algorithms, reported performance metrics are calculated using six training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of six evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time (s)	Top-1 travel time (s)
Webster	N/A	N/A	1.08	(22.2, 17.7)	(65.9, 29.6)	65.7
Max-pressure	N/A	N/A	1.02	(12.7, 11.3)	(59.5, 24.1)	59.2
Actuated	N/A	N/A	1.07	(13.9, 12.4)	(61.0, 24.7)	60.7
DQN (Random)	N/A	N/A	1.21	(27.4, 21.1)	(72.9, 32.7)	72.6
DQN	*Min. speed delta*	-396.56	1.09	(22.5, 17.5)	(66.2, 28.3)	65.1
DQN	*Min. delay*	-320.90	1.08	(22.5, 17.6)	(66.1, 28.4)	65.6
DQN	*Min. waiting time*	-307.22	1.08	(22.8, 18.2)	(66.4, 28.9)	65.4
DQN	*Min. pressure*	162.09	1.22	(27.2, 21.9)	(72.2, 31.9)	71.8
DDPG (Random)	N/A	N/A	1.32	(31.8, 25.8)	(78.4, 37.8)	78.0
DDPG	*Min. speed delta*	-400.27	1.10	(22.7, 16.7)	(66.5, 26.9)	66.2
DDPG	*Min. delay*	-306.35	1.04	(21.4, 16.8)	(64.7, 27.9)	63.1
DDPG	*Min. waiting time*	-283.78	1.03	(21.0, 16.2)	(64.1, 27.5)	63.4
DDPG	*Min. pressure*	147.02	1.12	(23.6, 18.2)	(67.6, 28.9)	67.2

Table B.9: Complete set of performance metrics for the *grid* network under the *variable* demand (+ denotes the inclusion of the *time variable*). The base demand is defined with the mapping $\{d(1) = 0.05, d(2) = 0.20, d(3) = 0.30\}$. For trainable algorithms, reported performance metrics are calculated using six training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of six evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value. The free-flow period corresponds to the 22h00-23h00 interval, and the congested period to the 08h00-09h00 interval.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time free-flow (s)	Travel time congested (s)	Travel time (s)	Top-1 travel time (s)
Webster	N/A	N/A	0.98	(19.8, 17.9)	(57.9, 28.3)	(65.7, 30.2)	(61.5, 29.2)	61.4
Max-pressure	N/A	N/A	0.88	(10.9, 10.6)	(46.0, 18.9)	(61.8, 23.8)	(55.6, 22.8)	55.3
Actuated	N/A	N/A	1.04	(13.1, 11.9)	(55.0, 22.9)	(63.5, 26.3)	(58.5, 24.1)	58.3
DQN (R)	N/A	N/A	1.14	(25.9, 20.8)	(61.7, 29.0)	(79.1, 35.5)	(69.6, 32.0)	69.2
DQN	*Min. speed delta +*	-208.83	0.99	(19.9, 17.3)	(56.4, 26.7)	(66.6, 29.5)	(61.6, 27.8)	59.8
DQN	*Min. delay +*	-168.45	1.00	(19.8, 17.2)	(57.1, 27.2)	(65.9, 28.3)	(61.5, 27.5)	60.1
DQN	*Min. waiting time +*	-160.50	0.99	(20.0, 17.2)	(58.0, 27.8)	(66.7, 28.8)	(61.7, 27.7)	60.0
DQN	*Min. pressure +*	88.83	1.18	(26.2, 22.0)	(65.1, 31.2)	(74.0, 33.1)	(69.6, 31.6)	68.2
DDPG (R)	N/A	N/A	1.25	(30.1, 25.5)	(64.5, 31.8)	(88.3, 44.3)	(75.0, 37.0)	73.8
DDPG	*Min. speed delta +*	-228.52	1.07	(22.1, 18.2)	(58.6, 28.0)	(69.7, 29.2)	(64.5, 28.2)	63.7
DDPG	*Min. delay +*	-167.17	0.99	(19.7, 16.3)	(56.4, 26.8)	(66.0, 27.4)	(61.3, 26.8)	59.7
DDPG	*Min. waiting time +*	-161.49	1.01	(20.1, 16.7)	(56.0, 26.6)	(66.8, 27.9)	(61.9, 27.1)	60.2
DDPG	*Min. pressure +*	78.68	1.09	(23.2, 18.9)	(60.0, 29.8)	(70.7, 29.1)	(65.7, 29.0)	65.0

Table B.10: Complete set of performance metrics for the *grid* network under the *cyclical* demand. The base demand is defined with the mapping $\{d(1) = 0.05, d(2) = 0.15, d(3) = 0.20\}$. For trainable algorithms, reported performance metrics are calculated using six training runs, each evaluated with a set of three evaluation rollouts of 24 hours; non-trainable methods are evaluated with a set of six evaluation rollouts of 24 hours. For tuple entries, the first and second positions correspond, respectively, to the mean and standard deviation values; non-tuple entries display the mean value.

Controller	MDP	Cum. reward	Number of stops	Waiting time (s)	Travel time (s)	Top-1 travel time (s)
Webster	N/A	N/A	1.12	(23.1, 18.0)	(67.5, 30.2)	67.1
Adaptive-Webster	N/A	N/A	1.08	(22.1, 18.3)	(66.0, 30.4)	65.8
Max-pressure	N/A	N/A	1.03	(13.1, 11.7)	(60.7, 24.7)	60.5
Actuated	N/A	N/A	1.09	(14.9, 13.7)	(62.3, 25.9)	62.1
DQN (Random)	N/A	N/A	1.31	(30.3, 23.9)	(77.3, 36.8)	76.9
DQN	*Min. speed delta*	-407.61	1.10	(23.1, 18.6)	(67.2, 30.0)	66.4
DQN	*Min. delay*	-317.13	1.08	(22.1, 16.9)	(66.0, 28.4)	65.1
DQN	*Min. waiting time*	-307.12	1.10	(22.7, 17.7)	(66.8, 29.4)	66.3
DQN	*Min. pressure*	154.27	1.24	(27.2, 21.2)	(72.7, 32.1)	71.2
DDPG (Random)	N/A	N/A	1.47	(36.3, 31.2)	(85.0, 45.0)	83.9
DDPG	*Min. speed delta*	-411.52	1.12	(23.3, 18.2)	(67.6, 29.5)	66.6
DDPG	*Min. delay*	-321.03	1.08	(22.4, 17.0)	(66.4, 28.5)	64.6
DDPG	*Min. waiting time*	-300.45	1.09	(22.1, 16.3)	(66.1, 27.8)	64.9
DDPG	*Min. pressure*	148.98	1.18	(25.5, 21.6)	(70.3, 33.0)	68.3