TÉCNICO LISBOA

Critical Review of Reinforcement Learning Based Adaptive Traffic Signal Control

Miguel Ângelo Mendes Coelho

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor(s): Prof. Francisco António Chaves Saraiva de Melo
Prof. José Alberto Rodrigues Pereira Sardinha

Examination Committee

Chairperson: FILL IN LATER
Supervisor: FILL IN LATER
Member of the Committee: FILL IN LATER

November 2021

Acknowledgments

I would like to express my sincere gratitude to my supervisors, Prof. Francisco Melo and Prof. Alberto Sardinha, for all the knowledge and guidance they provided during the entirety of my thesis. They were always available when I needed any sort of help and showed me the right path when I had any doubts.

This thesis wouldn't be possible without the help of Pedro Santos, who built the work this thesis leans on and provided me with help, specially during this thesis' project.

I especially thank Guilherme Varela, who worked with me side-by-side during the entire duration of the development of this thesis and helped me with is knowledge and previous works.

I thank my family for always being there throughout the development of my thesis and my life and showing me love when I needed it the most. In particular, my parents, who helped me in every step and provided the financial support that enabled me to take this degree.

I also thank the GAIPS lab managers for maintaining the machines where this thesis experiments were executed.

Finally, I would like to thank my friends who kept me company during my academic journey, especially, Daniel Teixeira, Francisco Lopes, Bernardo Esteves, Ricardo Silva and João Gonçalves.

Thank you all for your support.

Resumo

O controlo de sinais tráfego adaptativo é uma tarefa de desempenho crítico em sistemas de transporte inteligentes: Pelo uso de sensores avançados, os controladores podem responder a condições reais de tráfego nas intersecções e produzir políticas capazes de mitigar impedimentos de tráfego e evitar engarrafamentos. Os sistemas multi-agentes baseados na aprendizagem por reforço (MARL) fornecem uma estrutura natural para lidar com a tarefa. Em MARL, cada agente controla uma intersecção e pode se coordenar com seus vizinhos para obter uma política óptima em toda a rede. No entanto, é ainda uma tarefa difícil comparar abordagens, pois há muitas composições possíveis. Esta tese faz uma comparação crítica entre diferentes controladores de tráfego encontrados na literatura. Mostramos que o uso de aproximadores não lineares, mecanismos de coordenação e aumento da observabilidade em cada intersecção são os principais factores de desempenho.

Palavras-chave: Sistemas de transporte inteligentes, Aprendizagem por reforço, Aprendizagem por reforço multi-agente, Aprendizagem de máquina

Abstract

Adaptive traffic signal control is a performance-critical task in intelligent transportation systems: Through the use of advanced sensors, controllers can respond to actual traffic conditions at intersections and produce policies capable of mitigating bottlenecks and preventing gridlocks. Reinforcement learning-based multi-agent systems (MARL) provide a natural framework to address the task. In MARL, each agent controls one intersection and can coordinate with its neighbors to attain an optimal network-wide policy. However, it remains a daunting task to compare approaches as there are many possible compositions. This thesis makes a critical comparison across different traffic controllers found in the literature. We show that using non-linear approximators, coordination mechanisms and increasing the observability at each intersection are key performance drivers.

Keywords: Intelligent transportation systems, Reinforcement learning, Multi-agent reinforcement learning, Machine learning

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The most recent growth in population led to an increase in social and economic activities that then resulted in a higher demand for transportation. The adoption of modern transportation systems allows city planners to cope with this escalating demand from transportation users while preventing the ever-increasing infrastructure expenses. The effectiveness of current transportation systems is crucial to ensure short travel times of individuals and goods while also having minimal impact on the ecological environment. This high transportation demand has outclassed the existing traffic infrastructures that are now incapable of handling everyday traffic congestions. The present traffic infrastructure uses traffic lights, signaling and transit planning to minimize these congestions, where traffic lights are the most typical control mechanism.

One way of increasing the traffic capacity is to expand the current road infrastructure. This is normally avoided due to its monetary cost, extensive time commitment and environmental constraints.

Another approach is to create Intelligent Transportation Systems (ITS). The overarching goal of Intelligent Transportation Systems (ITS) is to reduce congestion and provide safety for transportation users [1]. Adaptive traffic signal control (ATSC) in ITS has been consistently studied for more than 60 years now, starting from the Webster method [2]. The field benefits from developments in computation power, cloud infrastructure, and enhanced sensor capacity [3]. It plays a pivotal role in alleviating bottlenecks at intersections and mitigating traffic congestions. Traffic congestions have a detrimental effect on the economy since they waste valuable public and private resources, lower living standards for the population, and harm the environment through increased emission of pollution [4].

Classic traffic signal control systems such as the Webster [2] method have three main limitations: They use manual labor for data collection, historical data to design signal plans, and make some assumptions, such as , the traffic patterns will hold until the next re-calibration. In contrast, adaptive traffic signal controllers can react to instantaneous traffic conditions using sensors, without manual input or any traffic assumption. For instance, the Max-Pressure [5] controller selects the phase with the maximum difference between vehicles in the incoming approaches and vehicles in the outgoing approaches (pressure). This controller performs a rule-based local optimization, but it does not consider past traffic patterns or optimize for network-wide traffic.

Another approach to developing ATSC systems is to combine machine learning algorithms into available traffic data sensors. The advantage is threefold: The system incorporates historic traffic patterns and reacts, taking the best long-term action into account. The system can adapt automatically to new traffic patterns as they emerge, *i.e.*, *online* learning. And can optimize for plans for more than one intersection [6].

Reinforcement learning (RL) is a machine learning paradigm, usually modeled by *Markov decision processes* (MDP) [7]. MDPs describe sequential decision-making problems where an agent interacts with an environment. The agent observes the environmental state, makes a decision based on its policy and acts. In response, the environment transitions to another state and emits a reward signal based on the relative improvement of the system. The agent's objective is to search for a policy, the mapping from state to actions, that maximizes the average cumulative reward over many rounds of interactions with the environment.

The flexibility of the MDP framework comes with a drawback; there are many different ways to express a reinforcement learning agent system. The length of the queue can represent the environment's state [8], or the volume [9] in the incoming lanes, by the pressure on the intersection [5], among others. Similarly, many approaches to the action scheme, reward design, and learning algorithm are beyond the current work's scope. Wei, *et al.* [10] provides a comprehensive survey on all the state formulations and action schemes used in the literature.

A multi-agent reinforcement learning (MARL) system is a group of autonomous, interacting entities sharing an environment, which they perceive with sensors and act with actuators. MARL provides an alternative to a centralized system controlled by a single agent, when achieving centralized control might be either costly or non-viable. In the particular case of reinforcement learning-based adaptive traffic signal control (RLATSC), centralized control entails controlling hundreds or thousands of intersections, leading to a high computation cost due to the scale of the state and action spaces. Besides, the potential benefits from MARL include, according to [11]: Computational speed-up due to parallel processing. Enhanced robustness as there is the possibility of neighbor agents assuming the task of a malfunctioning agent. More effective policies due to the use of coordination mechanisms, allowing to maximize a common goal without having a centralized system. As MARL provides a way to decompose a task, the system can handle larger problem sets by adding more agents and scale quickly. Finally, learning agents benefit from decreased learning cycles due to communication and information sharing.

The straightforward way to approach MARL is building many independent learners (ILs) who learn oblivious to their neighbors. But, the dynamics of the environment may become unstable, *i.e.*, causing the *non-stationary effect*. Agents cannot differentiate environment changes that are caused by their actions, by other agents' actions, or even intrinsic changes from the environment itself. This phenomenon can prevent learning from happening altogether since the policy of an agent changes as their neighbors' policy changes. In RLATSC, coordination strategies allow increasing road network sizes while mitigating the *non-stationarity effect*.

As with the single-agent case, there are many ways to define multi-agent systems, including not only the MDP formulation but also other factors, such as, the type of discretization used, if coordination

methods are applied and how much information is given to each system during training and execution. Unless the many works in the literature are compared along the same axes, researchers are left unable to assess the aspects that make multi-agent reinforcement learning systems work.

Thus rises the main question of this work: How do different methods of discretization, coordination and observability affect traffic controllers when run on a homogeneous simulation environment?

1.1 Contributions

The major contributions of the current work are to create a stable environment and defined problem formulation to compare different traffic control strategies.

Specifically, this work aims to:

- Prepare the simulator that will be used to train and evaluate the traffic controllers and create a set of realistic scenarios for them to be tested on. In particular, a 1x3 arterial network and a 3x2 grid network.
- Create an effective Markov Decision Problem (MDP) formulation that allows for a meaningful comparison between the different traffic control methods.
- Develop baseline traffic controllers that will be used to compare with the other reinforcement learning-based controllers, in particular: a random controller, a fixed-time controller, a Webster controller and a Max-pressure controller.
- Develop reinforcement learning-based traffic controllers, in particular: an Actor-Critic (ACAT) model that uses tile coding [9], a Deep Q-Network (DQN) model [12] and three different implementations of the MARLIN algorithm [8], based on Fictitious Play.
- Analyse how do different methods of discretization, coordination and observability affect the baseline and reinforcement learning-based traffic controllers, using a set of evaluations metrics to access the performance of each controller in each of these 3 axis.

1.2 Thesis Outline

The next chapter covers some traffic light control terms and objectives as well as single and multi-agent reinforcement learning theory that will be relevant for this thesis. Chapter 3 explores the different approaches that have been used to solve this problem, namely, fixed timing and RL based controllers. Chapter 4 details the methodology used to set up the simulator, the MDP formulation used, what traffic controllers were implemented and how these were trained, tuned and tested. Chapter 5 defines the concrete implementation and parameters of each algorithm used as well as simulation specific information. Chapter 6 compiles and analyses some of the results obtained in this work, and finally, Chapter 7 recaps the problem and solution presented in this work and explores possible avenues for future research.

Chapter 2

Background

This Section covers some traffic control concepts that will be used in latter chapters, followed by the background on single-agent reinforcement learning that is modeled as a Markov Decision Process. It also presents an extension of this model for multi-agent systems which will be relevant for the multi-agent RL traffic controllers presented in this work.

2.1 Traffic control

In this Section, some traffic light control terms and objectives are defined. Mostly adapted from Wei, *et al.* [10].

2.1.1 Traffic terms definition

A traffic network can be represented by a graph where *junctions* are the nodes and *roads* are the edges, where each *road* contains a certain amount of *lanes*. An *intersection* is a type of *junction* where two *roads* intersect and where traffic light controllers are usually placed. The edges of a *road* that meet the intersection are labeled *approaches* and can be defined as *incoming*, when they lead traffic into the intersection, or *outgoing*, if the traffic in that lane is leaving the intersection. Figure 2.1 represents a traffic intersection with two incoming and two outgoing north-south approaches, each with two lanes, and two incoming and two outgoing east-west approaches, each with three lanes.

A *traffic movement* consists of vehicles moving from an incoming approach to an outgoing approach. These are usually defined as right-turn, through, left-turn or combinations of the previous three.

A *movement signal* is defined as a traffic movement that is allowed by the traffic controller. Figure 2.1 shows three valid movement signals, right-turn/through, through and left-turn. For example, in the left-turn movement, vehicles can only go to the outgoing south approach.

A *phase* defines a combination of non-conflicting traffic movements—those that do not result in vehicle collisions. The three movements represented in figure 2.1 are a valid phase.

A *signal plan* is a sequence of phases and their respective duration, or equivalently, their start time. The time that takes to cycle through every phase in a signal plan is defined as *cycle length*.

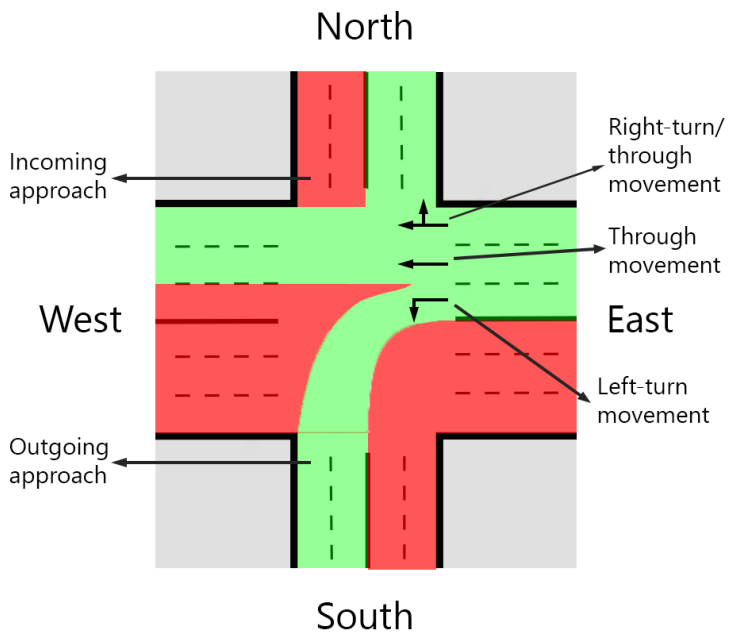


Figure 2.1: A traffic intersection with labeled terminology.

2.1.2 Traffic control objectives

The ultimate objective in a traffic network is to make vehicle movement more efficient and safe. Safety is mostly insured though phases with valid movements. Efficiency is usually quantified by different metrics, such as:

- Travel time: The average travel time of all vehicles in the network. In other words, the average time since each vehicle entered and exited the network.
- Vehicle Speed: The speed of the vehicles on a given phase, usually normalized by the speed limit of the respective lane.
- Number of stops: The average number of stops for all vehicles in the network.
- Throughput: The number of vehicles that traverse one or more intersections in a certain period of time, for example, in the last phase.
- Cumulative Delay: The sum of the times a vehicle as been stopped in any intersection.
- Queue Length: The sum of the number of waiting vehicles in a phase or set of lanes.

Thus, the objective is to minimize one or more of the previous metrics. The most important and most used metric is the travel time since it is strongly related to congestion, has the same meaning in all transportation modes and even those not familiar with traffic control terms can comprehend the concept. To evaluate the traffic controllers developed in this work, the travel time and speed of vehicles will be used. A more detailed explanation of this evaluation is done in Section 4.5.

2.2 Single-agent reinforcement Learning

As mentioned in the previous chapter, the reinforcement learning problem, introduced by [7] can be described as a *Markov decision process* (MDP). An MDP is a sequential decision making model, where an agent iteratively observes the environment, decides on an action to take and analyzes the reward for the selected action.

We can formally define a MDP with the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$,:

- $\mathcal{S} = \{s_1, ..., s_n\}$ is the finite set of environmental states.
- $\mathcal{A} = \{A(s),\ s \in \mathcal{S}\}$ where $A(s)$ is the finite set of all possible actions in the state s.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a stochastic transition probability function, where $\mathcal{P}(s, a, s')$ defines the probability of moving to state s' after choosing action a in state s.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $\mathcal{R}(s, a)$ defines the expected reward the agent will receive after choosing action a in state s.
- $\gamma \in [0, 1]$ is the discount factor, that represents how much future rewards are valued when compared to present rewards.

At each time step, t, the agent observes the current state of the environment $s_t \in \mathcal{S}$, and chooses an action $a_t \in A(s_t)$. This action causes a change in the environment state, defined by the transition probability function:

$$\mathcal{P}(s'|s, a) = \mathbb{P}\left[s_{t+1} = s' \mid s_t = s, a_t = a\right]. \tag{2.1}$$

Upon executing an action a_t in state s_t, the agent receives a reward with expectation given by $\mathcal{R}(s, a)$. The goal of the agent is to select its actions so as to maximize the *expected total discounted reward* (TDR):

$$\mathrm{TDR} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]. \tag{2.2}$$

The long-term *value* of an action a in a state s is captured by the optimal Q-value $Q^*(s, a)$, which can be computed using, for example, the *Q-learning algorithm* [13]. The Q-learning algorithm estimates the optimal Q-values as the agent interacts with the environment. Given a transition (s, a, r, s') experienced by the agent, Q-learning performs the update:

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha\big(r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s', a') - \hat{Q}(s, a)\big), \tag{2.3}$$

where α is a step size. Upon computing Q^*, the agent can act optimally by selecting, in each state s,

the optimal action given by the policy: $\pi^*(s) = \arg\max_a Q^*(s, a)$. This policy that maps each state s to the corresponding optimal action a is known as the *optimal policy* for the MDP.

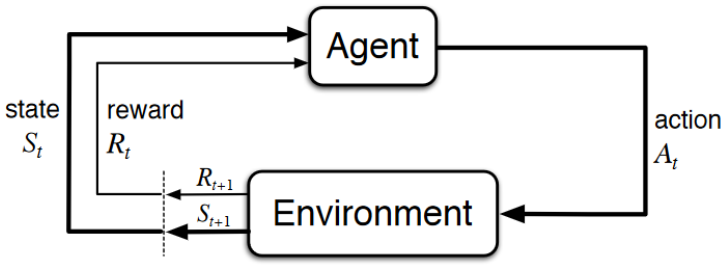


Figure 2.2: The interaction loop between the environment and the agent [7].

2.2.1 Q-function approximation

For large real-world problems, where the state and actions can be continuous, building a real-valued function that summarizes the long-term goodness of an action into a single number can be infeasible due to the scale of these state and action spaces.

One way to build reinforcement learning models in these complex environments is by using function approximation methods, these allow the models to scale by sacrificing representation power.

A function approximation can be defined as a feature map that transforms a discrete or continuous input into a smaller, usually discrete, output. For example, a feature map $\phi(s)$ can associate a state $s \in \mathcal{S}$ to some meaningful features. In the present domain, these features can be captured from the traffic sensors, including, delay, number of vehicles, queue, or pressure. Since discrete and non-linear Q-function approximation carry slightly different notations, they are presented separately.

Discrete approximation

For discrete, tabular methods, such as Q-learning, using a naive state space may lead to having a large combination state space that impedes the agent's ability to learn. For instance, in an intersection with $D = 4$ incoming lanes, each with capacity for twenty vehicles, the Q-learning algorithm must account for $20^4 = 160,000$ combinations. However, using a poor discretization method can result to losing the model's generalization, the ability of the model to perform well on unseen data.

It is possible to define a Q-function approximator $\Phi(\vec{x}) : \mathbb{N}^D \to \mathbb{N}^d, d \leq D$ and $range(\Phi(\vec{x})) < range(\vec{x})$, such as, tile coding [7]. Tile coding generalizes the state space into partitions called tilings. Each tiling is composed of a set of non-overlapping tiles that are slightly offset in the state space.

Using the previous example with 4 lanes, applying tile coding with 5 tiles reduces the state space to $5^4 = 625$ combinations.

Non-linear approximation

For non-linear Q-function approximation, an effective way to reduce the complexity of the system is to use artificial neural networks. Let $Q(s, a; \theta)$ be Q-function with an artificial neural network as function

approximator with parameters θ. Suppose $\vec{x} \in \{0, 1, \cdots, 40\}^2$ are the number of vehicles in the horizontal and vertical lanes in the incoming approaches of the intersection on Fig. 2.1. Then $\theta = \{W_1, \vec{b}_1, W_2, \vec{b}_2\}$ where $W_1 \in \mathbb{R}^{2\times u}$, $b_1 \in \mathbb{R}^u$, $W_2 \in \mathbb{R}^{u\times 2}$, $b_2 \in \mathbb{R}^2$ fully defines a feed-forward neural network [14] with some activation function and a certain number of actions as output. The forward pass is given with:

First layer propagation

$$\vec{h} = \sigma(W_1\vec{x} + \vec{b_1})$$

Second layer propagation

$$\vec{z} = W_2\vec{h} + \vec{b_2}$$

Softmax activation

$$(\vec{q})_i = \frac{e^{z_i}}{\sum_{z_j} e^{z_j}},$$

where the first layer propagation also includes an activation function σ and $(\vec{q})_i$ denotes the i-th component of vector $\vec{q}$. The output action can be recovered by performing an $a = \arg\max_i (\vec{q})_i$. This creates a non-linear function approximator that can scale to large, continuous state spaces.

2.2.2 Actor-critic

An Actor-critic algorithm has two modules: A critic that estimates the action-value function $Q(s, a)$ or the state-value function $V(s)$, following a given policy π, while starting from state s. This function is a measure of how good a policy is on the long run. The actor's role is to select an action a, given the state s. Every time step, after the actor selects an action, the environment transitions to a new state and the critic evaluates the quality of the action using the temporal difference (TD) error:

$$TD = \delta_{t+1} = r_{t+1} + \gamma V(s_{t+1}) - V(s_t),\ 0 \leq \gamma \leq 1. \tag{2.4}$$

Since past states also affect the TD error, eligibility traces can be added to the actor-critic algorithm so that the error of previous time steps can be weighted in the value function updates, while only requiring a single trace vector:

$$\begin{gathered} V(s) = V(s) + \alpha\delta_{t+1}e_{t+1}(s),\ s \in \mathcal{S} \\ e_{t+1}(s) = \begin{cases} \gamma\lambda e_t(s) & \text{if } s \neq s_t \\ \gamma\lambda e_t(s) + 1 & \text{if } s = s_t \end{cases} \quad 0 \leq \gamma, \lambda \leq 1. \end{gathered} \tag{2.5}$$

After the actor selects an action, the values of state-action pairs are updated:

$$P(s, a) = P(s, a) + \beta\delta_{t+1}e_{t+1}(s, a),\ s \in \mathcal{S}, a \in \mathcal{A}, \tag{2.6}$$

where β is the actor's learning rate.

2.2.3 DQN

Deep Q-network [12] is a well known RL method that approximates the Q-values with a neural network $\hat{Q}(s, a; \theta)$, where θ denotes the parameters of the model. At each step, the agent adds a transition (s, a, r, s') to a replay memory buffer, from which batches of transitions are sampled in order to optimize the parameters of the model such that the following loss in minimized:

$$\mathcal{L}(\theta) = \left(r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s', a'; \theta^-) - \hat{Q}(s, a; \theta) \right)^2. \tag{2.7}$$

The gradient of the loss is back-propagated only into the *behaviour network*, $\hat{Q}(s, a; \theta)$, which is used to select actions. The term θ^- represents the parameters of the *target network*, a periodic copy of the behaviour network. This algorithm has been popularized due to its impressive performance when playing Atari games [12].

2.3 Multi-agent Reinforcement Learning

In Adaptive Traffic Signal Control, the MDP formulation needs to be extended for a multi-agent setting, where agents do not have access to the global environment state. This extension is called a Decentralized MDP (DecMDP) [15].

For a system with N agents, its DecMDP can be described with the tuple:

$$(\mathcal{S}, (\mathcal{A}^{(n)})_{n=1}^N, (\mathcal{Z}^{(n)})_{n=1}^N), \mathcal{P}, (\mathcal{O}^{(n)})_{n=1}^N, \mathcal{R}, \gamma),$$

where:

- $\mathcal{S}$ is the state space for the entire system.
- $\mathcal{A}^{(n)}$ is the action space for agent n. Where $\mathcal{A}$ denotes the action space for the whole system:

  $$\mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^n \cdots \times \mathcal{A}^N.$$

  An element of $a \in \mathcal{A}$ is called a *joint action* and corresponds to the tuple $a = (a^1, \ldots, a^N), a^n \in \mathcal{A}^n$.
- $\mathcal{Z}^{(n)}$ is the observation space for agent n.
- $\mathcal{P}$ are the transition probabilities that describe the environment dynamics:

  $$\mathcal{P}(s'|s, a) = \mathbb{P}\left[S_{t+1} = s' | S_t = s, a_t = a\right]. \tag{2.8}$$

- $\mathcal{O}^{(n)}$ are the observation probabilities for agent n, describing the sensing process for that agent:

  $$\mathcal{O}(z^{(n)}|s) = \mathbb{P}\left[Z_t^{(n)} = z^{(n)} | S_t = s\right]. \tag{2.9}$$

- $\mathcal{R}$ is the expected reward function:

$$\mathcal{R}(s,a) = \mathbb{E}\left[r_{t+1}|s_t = s, a_t = a\right]. \tag{2.10}$$

- γ is the discount factor.

A policy for an agent n is a mapping $\pi^{(n)} : z^{(n)} \rightarrow \Delta(\mathcal{A}^{(n)})$ of the observation space of agent n to the distribution over the actions of that agent. A *joint policy* is a mapping $\pi : \mathcal{Z} \rightarrow \Delta(\mathcal{A})$ that can be seen as a tuple $\pi = (\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(N)})$. This policy maps the agents' observations $z \in \mathcal{Z}$ to a distribution over the joint actions $\mathcal{A}$. The goal of the agents is to compute a policy π that maximizes:

$$J = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | a_t \sim \pi(z_t)\right]. \tag{2.11}$$

2.3.1 Coordination mechanisms

Since the actions of one agent can affect other agents in nearby intersections, having isolated self-interested agents that only try to maximize the gain in their own intersection may lead to better local performance for some agents but worse global performance when dealing with large networks. Thus, some form of collaboration or information sharing between agents is necessary.

Coordination can be achieved by simply adding information about other intersections to the state space. However, this can lead to an exponential growth that increases with the number of intersections and is unfeasible in larger networks. Coordination can also be achieved with Coordination graphs [16], that focus on the network topology and limit the agents' interaction with its neighbors. Other coordination methods where agents reason and represent about the other agents' beliefs exist, mostly extending the field of Game Theory. The most relevant coordination method for this work is Fictitious Play.

Fictitious Play

In the field of Game Theory, Fictitious Play is a learning rule where each player presumes that the opponents are playing stationary strategies and, every round, all players best respond to the empirical frequency of play of their opponents.

This can be applied in a collaborative RL-based setting, where each agent plays a game with all its adjacent agents. During learning, the states and actions of each agent are shared with its neighbours so they can build an estimation of the other agent's policies. This estimation of policies is called a model.

Specifically, during training, each agent i builds a model that estimates the policy for each of its neighbours, $j \in NB_i$:

$$M_{i,j}([s_i, s_j], a_j) = \frac{V([s_i, s_j], a_j)}{\sum_{a_j \in A_j} V([s_i, s_j], a_j)}, \tag{2.12}$$

where Q-values are updated by the following rule:

$$br_i^k = max_{a_i \in A_i} \left[\sum_{a_j \in A_j} Q_{i,j}^k([s_i, s_j], [a_i, a_j]) \times M_{i,j}([s_i, s_j], a_j) \right]$$
$$Q_{i,j}^k([s_i, s_j], [a_i, a_j]) = (1 - \alpha) Q_{i,j}^{k-1}([s_i, s_j], [a_i, a_j]) + \alpha[r_i^k + \gamma br_i^k]. \tag{2.13}$$

During execution, each agent selects their actions using the estimated models of its neighbours:

$$a_i^{k+1} = \underset{a_i \in A_i}{\arg\max} \left[\sum_{j \in 1,2,\ldots,|NB_i|} \sum_{a_j \in A_j} Q_{i,j}^k([s_i, s_j], [a_i, a_j]) \times M_{i,j}([s_i, s_j], a_j) \right]. \tag{2.14}$$

This algorithm maintains an explicit coordination mechanism while addressing the curse of dimensionality for large-scale problems (such as large-scale traffic networks) by exploiting the principle of locality of interaction [17] since agents only build an estimate of the neighboring agents, and Modular Q-Learning [18] where the state space is partitioned into partial state spaces that consist of two agents. Hence, the size of this partial space is always $|S|^2$, independently of the number of agents that are learning, allowing for a scalabe coordinated MARL system.

2.3.2 Parameter sharing

Coordination and performance gains can also be achieved by sharing sensations, sharing experiences, sharing policies.

Formally, agents share sensations when they either have immediate access to its neighbors' observations or that a message being relayed by the neighbors is a representation of their current observation. Sharing experiences, extends the previous concept by sharing the tuple $(z_t^n, a_t^n, z_{t+1}^n)$, that now contains information about the action taken and how the environment reacted to that agent.

Sharing policies is the act of periodically having agents combining the policies, for example, in Deep Reinforcement Learning (DRL) this is achieved by using common neural network parameters for the policies of different agents or by performing model pooling, an averaging of the weights of many agents.

These methods form a technique called parameter sharing, that allows agents to learn at a faster rate with a fewer number of total experiences.

Chapter 3

Related Work

Traffic signal controllers have evolved over the years to better respond to the growing traffic flow by making use of the technological advancements in computation and the increasing data collected from sensors. This Section explores previous approaches that also tried to solve traffic control problems.

3.1 Fixed-timing methods

Traffic signal controllers with fixed timings normally define different cycle profiles and alternate between them depending on the time of day on an attempt to deal with the different traffic flows that are usually observed.

Some of these methods are defined by mathematical models that use derivative calculus, linear programming, and other optimization algorithms:

- Webster [2] is a closed form method that calculates the phase split ratio, using a fixed cycle length, that minimizes the travel time for a single intersection, assuming the traffic flow is uniform during a certain time period. This approach does not consider timing offsets for nearby traffic signals which can degrade the efficiency of transportation when considering a more complex environment.

- GreenWave [19] is a method that implements coordination by optimizing the timing offsets of nearby intersections to minimize the number of stops for vehicles moving in a certain direction.

- Maxband [20] also optimizes timing offsets by minimizing stops but now considers a two-way environment. It uses linear programming to achieve the maximal bandwidth in both ways for a set of arterials.

Other methods use traffic simulators to build the traffic model. For example, Rouphail, *et al.* [21] applied a genetic algorithm (GA) using the CORSIM simulator to minimize link delay and queue time in a 9 intersection network but the results were limited by the slow convergence of the GA algorithm.

3.2 Adaptive systems

Later traffic controllers started using models that used sensor data to optimize different traffic metrics to better adapt to the changes in traffic flow:

- Max-pressure [5] aims to balance the queue length of neighboring intersections. The pressure of a phase is defined as the difference between the queue length of incoming and outgoing lanes. It is shown that Max-pressure maximizes the throughput of the system if the pressure of the phases is minimized.

- SCATS (Sydney Co-Ordinated Adaptive Traffic System) [22] iteratively selects the next signal plan, from a set of pre-defined plans, depending on the current traffic conditions and a pre-defined performance measure. The model infers the performance of all plans before each cycle and then selects the plan that has better performance.

- Tubaishat, *et al.* [23] use a wireless sensor network to create a decentralized system that improves the localized flow and coordination between neighbor traffic lights.

- RHODES [24] is a hierarchical system that predicts the traffic load on each link and allocates phase time according to the predictions.

- Liu, *et al.* [25] developed a controller that identifies upstream and downstream vehicles, in intervals of 15 minutes, to measure their delay and then choosing a signal timing plan that minimizes it.

- Tan, *et al.* [26] developed a traffic controller that senses the number of incoming vehicles and uses fuzzy logic to determine the green time of a single intersection.

- Lee, *et al.* [27] also used a fuzzy logic controller but in multiple intersections. The controller takes decisions based on vehicle data of adjacent junctions.

These systems generally outperform fixed-timing controllers but were tested in very simple scenarios with a single intersection or with very restricted traffic assumptions, like uniform traffic flow and cannot adapt well to real world city-level traffic.

3.3 Reinforcement learning

Earlier models were limited by simplistic simulations and lack of computation power, but with the latest technological growths, highly complex simulators emerged and a became popular tool to develop traffic control models.

Recently, reinforcement learning has become popular to build traffic signal controllers, since an agent can learn traffic control policies by interacting with the environment, without having a pre-defined model of the system.

The reinforcement learning framework fits naturally into the traffic light controller problem since we can define the traffic controller as the agent, the traffic data as the state representation and the phase controls as the agent's actions.

Different learning models have been explored to build traffic signal controllers. However, it is hard to compare the proposed solutions and results since the problem definition varies greatly between the literature.

This Section provides an overview of the literature that adopted a reinforcement learning approach to solve the traffic control problem. Section 3.3.1 explores the MDP definitions used in the literature, namely, the most used states, actions and rewards. Section 3.3.2 explores the classic, single-agent models used in the literature and Section 3.3.3 extends the previous model to allow for multi-agent settings.

3.3.1 Modeling

Based on the MDP framework defined in Section 2.2, the most used representations for the states $\mathcal{S}$, actions $\mathcal{A}$, and rewards $\mathcal{R}$ in the literature are described in this section.

In the context of the traffic control problem, the environment is the network of roadways that approach the intersections and the vehicles that traverse these roadways. Since the traffic problem has a complex and stochastic environment, most works rely on simulators that accommodate the RL framework. Even though in these simulators the environment is fully observable, using all of the data available might prove difficult when trying to apply the model to the real world.

State representation

Most works use state representations that are derived from data that real sensors would capture in a typical urban roadway. The definition of the environment state in the literature is a combination of one or more of the following parameters:

- Cumulative Delay: The sum of the times a vehicle as been stopped in any intersection.
- Phase: The information about the current phase. For example, the index of the current phase can be used in systems that have pre-defined phases or the time passed since the signal turned red to prevent a red signal from being active too long, promoting fairness between other lanes.
- Queue length: The total number of waiting vehicles in a certain lane or a set of lanes. In some works, the number of approaching vehicles is also considered for this metric.
- Volume: The total number of vehicles, moving or not, in a certain lane or a set of lanes.
- Vehicle position: The position of each vehicle in the network, usually implemented with a boolean matrix, where each cell represents a portion of of the network and a value of 1 represents that a vehicle is in that location at a certain time.

- Vehicle Speed: The speed of the vehicles on a given phase, usually normalized by the speed limit of the respective lane.
- Saturation: The ratio between the current traffic flow and the maximum traffic flow in an intersection.

These state definitions can be simpler or more complex depending on the number and complexity of the metrics used. A more complex state representation is built in hopes to better represent reality, but may lead to very big state spaces that can hinder the training performance, without actually boosting the effectiveness of the model. It should be noted that some of these values are harder to obtain than others. For example, vehicle speed and position might be difficult to accurately obtain from sensors positioned in the roadway.

Action representation

In the traffic control problem, the set of actions, $\mathcal{A}$, is the set of phase controls of the traffic controller. The phase control actions that are available to the controller depend on the assumptions of the model used. Specifically, if the controller needs to select phases from a pre-defined set of valid phases or can choose them freely; if it needs to follow the order of a pre-defined phase plan or it learns the best order; if the phase cycle time is fixed or variable.

Examples of actions used in the literature are:

- Phase duration: The agent chooses the current phase duration, following a pre-defined phase plan.
- Phase ratio: Given a fixed cycle time, the agent defines the phase split ratio, usually taken from a set of pre-defined split ratios.
- Keep or change phase: At every decision point, the agent decides to either extend the current phase or change to next phase, following a pre-defined phase plan.
- Choose next phase: The agent decides what is the next phase, from a set of pre-defined phases. This way the agent builds the phase plan, without explicitly following a cycle.

Reward representation

Regarding the reward function $\mathcal{R}$, in the traffic control problem the objective is, ultimately, to minimize the travel time of all vehicles in the network. However, this metric can be hard to obtain for every vehicle since most times the vehicle destination is unknown.

Other metrics that can be more easily measured after each action is taken, in the hope that minimizing them also minimizes the travel time of all vehicles.

Some of the metrics used in the literature are the following:

- Cumulative Delay: The cumulative delay (as defined in the state representation) of all vehicles in a set of lanes or phase.

- Queue length: The total number of waiting vehicles in a certain lane or a set of lanes.
- Volume: The total number of vehicles, moving or not, in a certain lane.
- Vehicle Speed: The speed of the vehicles on a given phase, usually normalized by the speed limit of the respective lane.
- Throughput: The number of vehicles that traverse one or more intersections in a period of time, for example, in the last phase.

The reward function can be defined as a weighted sum of one or more of the previous metrics, this approach can be tricky since small changes in each weight can greatly impact the results of the model and there is no straightforward way of tuning these weights. Thus, the weights are generally tuned empirically or a single metric is used.

3.3.2 Classic reinforcement learning

The first distinction in the different reinforcement learning methods is if the transition probability function, $\mathcal{P}$, needs to be learned or not.

In model-based methods the agent learns a transition model that estimates the probability of moving between states given the possible actions and then computes the rewards of each transition. Then, using methods based on dynamic programming, it estimates the value function and makes decisions based on this estimation.

While model-based methods need to learn $\mathcal{P}$ and $\mathcal{R}$, model-free methods bypass this step and learn the value function or policy by interacting with the environment and observing the rewards directly.

In the context of the traffic control problem, learning the transition probability function means modeling the environment, such that metrics like vehicle speed, position and acceleration can be predicted.

Wiering [28] used a model-based approach in a multi-agent model that operates in a 6 intersection network where each intersection is controlled by an agent. Each controller receives the discretized position and destination of each car in the approaching lanes, leading to 2^{78} possible traffic situations. Even though the defined RL-controllers outperform more simple controllers, like fixed time and Longest Queue First (LQF)[1], this model assumes that each car can communicate with each traffic light controller, which is currently unfeasible. The network is also simplified since every street has the same number of lanes, resulting in an unrealistic homogeneous traffic pattern. The author also mentions the possibility of having smarter driving policies that avoid congested intersections, assuming the previous communication is made possible.

The previous work was extended in [29] by adding the direction of each car to the state and tested on a bigger network with 36 roads and 15 intersections. The RL controllers outperformed fixed controllers and co-learning between cars and traffic lights minimized waiting time by 50%.

[1] A policy that sets green phases to approaches with the longest queue first.

Some works [28, 30, 31, 32] have tried model-based approaches but most of the research community adopts model-free methods since it is difficult to fully model $\mathcal{P}$, given the natural unpredictable behavior of human drivers.

Most works using model-free methods rely on algorithms like Q-learning and SARSA to learn a optimal traffic control policy. Thorpe at al. [33] created a model-free system using SARSA and compared the performance of 3 different state representations, namely, the volume and the presence of absence of vehicles in each section of the network, by dividing each lane in equal and unequal distance sections. The RL model outperformed fixed time and greatest volume controllers, independently of the state representation used, and the state with unequal distance sections outperformed the other 2 state representations.

Zhou, *et al.* [34] compared a SARSA model with a fixed time approach. The authors used the saturation ratio between current and maximum flow and vehicle speed, both normalized in 7 discrete values for a total of 49 states. The SARSA model had almost half travel time when compared to an offline model when the traffic flow was low but had negligible difference in high traffic flow.

El-tantawy, *et al.* [35] compared different learning methods, state, action and reward representations and action selection methods. For a single intersection, an approach based on TD(λ) with eligibility traces outperformed Q-learning and SARSA. RL-controllers generally outperformed pre-timed controllers regardless of the state representation. When the arrival rate is uniform, the best approach was to follow a pre-defined phase plan but when the arrival rate varies, letting the controller choose the order and duration of each phase showed better results. For action selection mechanisms during learning, a mix of ε-greedy and softmax had better online performance and faster convergence when compared to both ε-greedy and softmax alone.

Touhbi, *et al.* [36] extended the network used by El-tantawy. *et al.* [35] with different traffic volumes and a modified state definition, and compared different reward functions. For low traffic flow, the reward definition is irrelevant for the controller performance. For high traffic flow a reward based on the cumulative delay outperformed rewards based on queue length and throughput.

Earlier reinforcement learning based controllers are applied to a single intersection since the state space grows exponentially with the number of intersections that are controlled. Given that single intersection models are oversimplified and cannot extrapolate to city level traffic, other works tried to apply reinforcement learning to multiple traffic junctions by building multi-agent models.

3.3.3 Multi-agent reinforcement learning

In a multi-agent setting, each agent controls one intersection in a traffic network with several intersections. This way, the explosion of the state space is minimized by making each agent operate on a small partition of the environment. In a non-collaborative approach, each agent tries to maximize a certain reward, such as queue length or cumulative delay, of its own intersection by using the state that represents that intersection. These are usually called Independent learners (ILs).

Independent Learners

Early systems comprised of independent learners (ILs) and few intersections. These perform better in smaller intersections, but with time, researchers were able to adapt ILs to larger road networks.

Camponogara, *et al.* [37] created a multi-agent system based on Q-learning and modeled it as a distributed stochastic game. The authors applied the system in a simple network with 2 connected intersections and compared it with a random and Longest Queue First policy. The proposed multi-agent model had significant performance gains over the other two policies. However, the agents did not collaborate and the proposed scenario was very simplistic.

Aslani, *et al.* [9] uses a well-known reinforcement learning model, actor-critic, and a classical discrete function approximation method, *tile coding* [7] to control 50 junctions in downtown Tehran.

Mnih, et al. [12] introduce the *deep Q-network* (DQN) in the domain of Atari learning environment (ALE). The approach is characterized by the use of deep artificial neural networks [14], to estimate the Q-function, and the use of a running replay buffer to store experiences, defined by tuples (s, a, r, s'), that serve as input to the neural network. DQN is soon adapted for the task of ATSC controlling one junction [38], outperforming common baselines.

Chu, *et al.* [39] verify that ILs using DQN under perform a greedy algorithm that selects the phase with the largest number of cars. The DQN-ILs also under perform the much simpler Q-learning counterparts for a 25 junctions road network. The results seem to suggest that there is a trade off between size and performance.

Collaborative Learners

Since the actions of one agent can affect other agents in nearby intersections, having isolated self-interested agents that only try to maximize the gain in their own intersection may lead to better local performance for some agents but worse global performance, specially when dealing with large networks. Thus, some form of collaboration or information sharing between agents is used to attempt to maximize this global performance.

The naive approach would be to simply add information about all other intersections to the state space. This leads to an exponential growth that increases with the number of intersections and is unfeasible in larger networks. Thus, the main challenge in the multi-agent setting is to implement coordination and information sharing between the agents while maintaining a state-space with manageable size.

There are two types of collaborative MARL systems that are relevant for this work: Joint action learners (JALs), that explicitly build a model for coordinating agents' actions, and agents that communicate.

Coordination graphs are a form of JAL and have been applied on reinforcement learning-based adaptive traffic signal control.

Kuyer, *et al.* [40] use the Green Light District (GLD) simulator to design a vehicle-based model like the model-based approach created by Wiering [28]. The system achieved coordination by using Max-Plus, a coordination graph algorithm. The model was compared to the original Wiering model and to the extension made by Steingröver [41] that adds a congestion bit to the state space. The devised model

outperformed the others in both small (3 to 4 intersections) and big (15 intersections) networks.

Van der Pol [42] applied a deep learning approach to single and multi-agent settings. The learning agent uses a Deep Q-Network (DQN) algorithm with a binary matrix as input that represents if a vehicle is present in a certain location or not. For a single intersection network, the DQN agent had better stability and performance when compared to a baseline agent using linear approximation. The author then applied the architecture made to play Atari games [12] in the traffic control problem, namely, batch normalization, prioritized experience replay and double Q-learning. The performance and stability of each method was tested. Apart from batch normalization, all methods had an improvement in stability and performance. A fine-tuned DQN agent was then created from the previous experiments and used in a multi-agent scenario in increasingly complex networks, applying the Max-Plus algorithm with transfer learning, a method where the local joint function between agents is learned in a smaller problem, a 2x1 intersection network, and is then used as initialization for the local joint function for harder problems (a 3x1 and 2x2 network grids). The DQN agent with transfer learning had better performance than the vehicle-based approach introduced by Wiering [28] and the Kuyer's extension [40] that used the base Max-Plus algorithm in Wiering's model.

Another joint action learning approach is using a variant of *fictitious play* where an agent selects its action according to the estimates their opponents' reaction to its state. Aiming to reach a steady-state condition, or an equilibrium where no agent has an incentive to deviate.

El-tantawy *et al.* [8] introduces a variant of fictitious play where agents observe the state of the environment, each picks an action, and receives a local reward. The reward is based on reducing the delay on the intersection and is known only to the intersection. Instead of communicating the following action, each agent stores a model from their neighboring agents' action. By counting the relative frequency that a particular neighbour has picked an action, the agents can update their action model. The method uses tabular Q-learning and was shown to control traffic in downtown Toronto, composed of 59 intersections.

Finally, some agents train communication messages end-to-end, follow a learning to communicate paradigm [43].

In Wei [44], the agents communicate representations of the neighbors' traffic state. Those representations are estimated using a special DQN to learn data flows in graphs. Agents exchange representations and ultimately share the parameters learned that compose the policy via parameter sharing (weight averaging from multiple learned policies).

Zheng [45] proposes FRAP, a DQN that can learn phase representations that accommodate many types of intersections by using parameter sharing.

Steingröver, *et al.* [41] extended the model-based system proposed by Wiering [28] by introducing communication between agents by adding a bit to the state representation that represents the traffic congestion in nearby intersections and by using this congestion information when estimating the optimal action.

This model was also extended by Iša, *et al.* [46] by adding another bit to the state space that represents if a following lane is obstructed or not. The main problem in the previous approaches is the exponential increase in state-space. For example, the space of the model of Iša, *et al.* [46], that adds

bits for congestion and obstruction is four times larger than the original model proposed Wiering [28].

Collaborative multi-agent systems are a way to handle the curse of dimensionality when considering complex traffic networks and are shown to outperform fixed-timing, single RL agents and non-collaborative multi-agent RL models. However, most works rely on either directly adding information about other agents to the state representation, which usually leads to a state-space explosion, or take advantage of coordination graph approaches, like the Max-plus algorithm, that exploits the space locality of the agents. The actual effectiveness of these coordination methods is hard to grasp since each work defines a different set of state-action representations and testing scenarios.

This work is inspired on [47, 48] where comparisons are made between traffic controllers with a fixed MDP formulation. To ensure fairness, a methodology inspired by Varela [49] is used for building experiments and making evaluations.

Chapter 4

Methodology

Following a rigorous methodology is essential in making experiments reproducible and the results comparable. The methodology used is slightly adapted from Varela [49], a methodology for Reinforcement Learning based Adaptive Traffic Signal Control for multiple coordinating agents. Whereas the original methodology for independent learners consisted of four steps, in this work, step number two is expanded into two distinct parts: MDP formulation and RL method. The five steps are as follows: the simulation setup, the MDP formulation, the RL method, training and evaluation (Fig. 4.1).

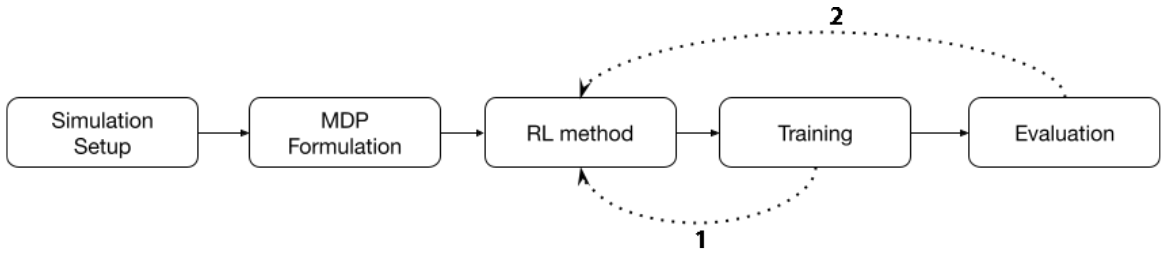


Figure 4.1: Diagram illustrating the proposed methodology, composed of five stages. Solid arrows denote the main development flow, whereas dashed arrows denote the iterative process of model tuning.

Since the Markov decision process (MDP) defines the optimization problem, any meaningful comparison between Reinforcement Learning methods, must be have the same base MDP. Moreover, the MDP formulation has a decisive impact on the performance of the reinforcement learning models, as it can be demonstrated by holding the learning algorithms fixed and changing the base MDP formulation [50, 51]. In this work, a base MDP is fixed and several baselines and RL-based methods are compared, to test distinct function approximators, coordination methods and observation scopes.

4.1 Simulation setup

The first step of the proposed methodology is the simulation setup. This consists on the choice of traffic simulator used, where the agents will train in a realistic simulation and attempt to learn effective traffic control policies, the topology of the networks used in said simulator, taken from the city of Lisbon and finally, defining traffic demands.

4.1.1 Traffic simulator

In contrast to traffic macro-simulators, that simulate traffic flow as a whole, traffic micro-simulators like Paramics [52], SUMO [53], AIMSUN[1] and CityFlow [2][54] generate vehicle dynamics, *i.e.*, each vehicle property, such as position, speed, acceleration, route and emission rates are simulated for each vehicle. These have been used to evaluate current traffic controllers and prototype new ones. In the reinforcement learning context, they can be used to model the environment where the agent learns the traffic policy.

This work uses the CityFlow[2] micro-simulator [54] since it is open-source, simple to use and more efficient than other widely used simulators like SUMO [53].

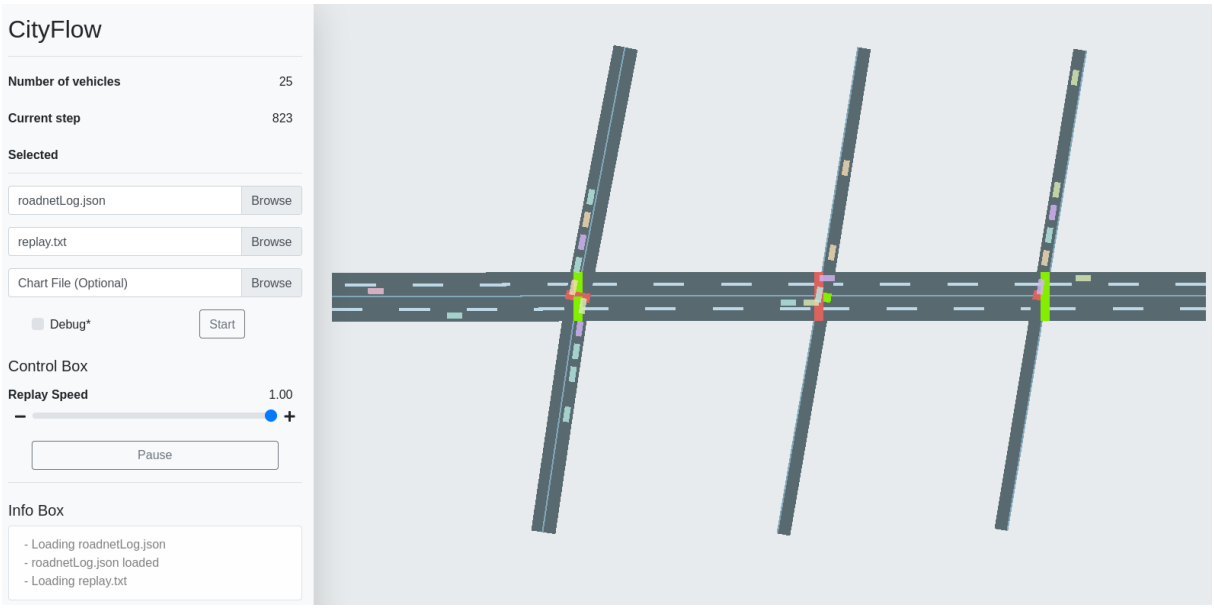


Figure 4.2: CityFlow simulator screenshot.

4.1.2 Road networks topology

Networks can be either synthetic or extracted from real world locations. Available open source services, such as OpenStreetMap [55], allow segments of cities' districts to be exported and, during the simulation setup step, 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.

In this work, all networks use geospatial data from OpenStreeMaps to build the configuration files to be used by the simulator. The following steps are required: **(i)** Extract the region of interest from OpenStreetMap and open the resulting file with the JOSM[3] editor, an extensible editor for OpenStreetMap files, in order to fine-tune the network; **(ii)** Convert the edited OpenStreetMap file into a SUMO network format using the `netconvert`[4] tool; **(iii)** Open the resulting SUMO network file with the

[1] `http://www.AIMSUN.com`, Transporting Simulation Systems (TSS).
[2] `https://cityflow-project.github.io`
[3] `https://josm.openstreetmap.de/`
[4] `https://sumo.dlr.de/docs/netconvert.html`

`netedit`[5] tool, a graphical network editor for SUMO, in order to ensure that all intersections are properly setup; **(iv)** Convert the sumo network file to a CityFlow compatible network file using the converter tool provided by CityFlow.

4.1.3 Traffic demands

In the domain of ATSC, the traffic demands that are simulated can be either derived from real-world data or synthetically made.

Real-world data allows for the creation of realistic traffic demands that match real observations, shortening the gap between the simulator and the real-world, where the implemented traffic controllers are to be deployed at some point. However, this complicates the setup process due to the fact that this data needs to be validated before being used and is usually network specific. This data can also be hard to obtain, noisy or even unavailable. As such, data-driven traffic demands are outside of the scope of this work.

For all experiments in this work, a constant synthetic traffic demand is used.

4.2 MDP formulation

An MDP consists of a state feature, a reward signal, an action schema and an observation scope. Like defined in Section 2.3, in MARL, a group of collaborating agents is defined by a DecMDP that accounts for the lack of observability and interactions between agents.

The DecMPD is defined by the tuple:

$$(\mathcal{S}, (\mathcal{A}^{(n)})_{n=1}^{N}, (\mathcal{Z}^{(n)})_{n=1}^{N}), \mathcal{P}, (\mathcal{O}^{(n)})_{n=1}^{N}, \mathcal{R}, \gamma). \tag{4.1}$$

- **State space** $\mathcal{S}$: $s \in \mathcal{S}$ is the state at time t, consisting of features of the incoming approaches of an intersection. In this work, it is described by a feature map $\phi(s)$ composed of an internal state and the data on its incoming approaches:

  $$\phi(s) = (x_g, x_t, x_0, \ldots, x_p, \ldots x_{P-1}).$$

  The internal state is defined by the index of the current green phase, $x_g \in \{0, 1, \ldots, P-1\}$, where P is the number of phases, and the time since this phase has been active, $x_t \in \{10, 20, \ldots, 90\}$. The feature x_p on the incoming approaches of any given agent n at phase p is defined by the cumulative delay:

  $$x_p = \sum_{v \in \mathcal{V}_p} e^{-5(v/v_p^*)}, \tag{4.2}$$

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

where v_p are the velocities of the vehicles in the incoming approaches of phase p for the agent and v_p^* is the speed limit for phase p. If every vehicle travels at the speed limit for the phase, or there aren't any cars in the phase then there is no delay. As a vehicle travels slower than the speed limit than delay becomes positive until it hits full stop ($v = 0$) and its delay becomes its maximum, 1. This choice was heavily influenced by the research done by Pedro [48], where several state features are compared. This definition of cumulative delay has three main benefits: **(i)** It packs the information of the number of cars and their velocities in one single number, having a reduced state space when compared to other features; **(ii)** It doesn't involve any assumptions w.r.t how slow does a vehicle need to be to be considered stopped; **(iii)** Finally, the negative exponential helps to balance situations where there are dominating influx in one phase.

- **Action space** $\mathcal{A}$: At time t, each agent n must take an action $a_t^n \in \mathcal{A}^n = \{0, 1\}$. The action space is homogeneous and binary where 0 means to keep the current phase active for the next decision step and 1 means to switch to the next phase. The action schema has the following constraints:

  – 5 seconds of yellow time,

  – 5 seconds of minimum green time,

  – 85 seconds of maximum green time,

  – Agents decide every 10 seconds.

  Hence, the signal plans are cyclical with variable cycle length from a minimum of 20 seconds and a maximum of 180 seconds for the two existing phases.

- **Observation space** $\mathcal{Z}$: Each agent n can observe only a partition of the state $z_t^{(n)} - \bigcup_n^N z_t^{(n)} = s_t$. $z_t^{(n)}$ is totally defined by the agent's internal state and the data on its incoming approaches. The feature space for agent n becomes $\phi((z^{(n)})_{n=1}^N)$.

- **Reward** $\mathcal{R}$: The reward collected by an agent for selecting action a at state s is given by:

$$\mathcal{R}(a, s) = -\sum_{p=0}^{P} x_p, \tag{4.3}$$

  where p is the phase indicator, x_p is the cumulative delay feature at phase p of agent n. The reward is actually a penalty where each agent seeks to minimize the delay on the incoming approaches.

- **Transition probability** $\mathcal{P}$: In RLATSC its usual to set $\mathcal{P} = \emptyset$, meaning that the transition probabilities are not explicitly estimated by the reinforcement learning algorithm. We must use *model-free* [7] reinforcement learning methods, in which the agents learn only by interacting with the environment.

- **Observation probability** $\mathcal{O}$: Following the same logic, $\mathcal{O} = \emptyset$.

- **Discount factor** γ: The discount factor $0 < \gamma \leq 1$ helps to shape the optimization problem, by leaning the agents' actions towards either minimizing the penalty on the short or long term. When

$\gamma \to 0$ the agent acts myopically, conversely if $\gamma = 1$ then the agent is indifferent in relation to instantaneous rewards and future rewards. In this work, $\gamma = 0.98$ is used.

The choices made in this MDP formulation were driven by previous research by P. Santos, *et al.* [51]. For a more detailed survey on the different MDP formulations used in the literature, refer to Wei [10].

4.3 Reinforcement learning methods

The Reinforcement learning methods consist in learning algorithms with different function approximation methods, coordination methods and observability scopes.

In this work, the coordination between agents is accomplished by using the original MARLIN algorithm, defined by El-tantawy *et al.* [8] for the ATSC domain and several variations of it.

It is worth noting that: **(i)** The original MARLIN algorithm receives a discrete state space, thus, the state defined in the previous MDP formulation needs to be discretized. **(ii)** In this algorithm, each intersection needs to share their states and actions during training and their states during the execution.

The concrete discretization methods used along with other algorithm specific implementations are further explained in Chapter 5.

Specifically, five learning algorithms were trained and tested:

- A group of independent learners using an Actor-Critic model.
- A group of independent learners using a DQN model.
- A group of collaborative agents using MARLIN without function approximation.
- A group of collaborative agents using MARLIN with function approximation.
- A group of collaborative agents using a continuous MARLIN model.

A few baseline models will also be implemented to compare against the RL-based methods. Specifically, a random controller, a fixed time controller, a Webster controller and a Max-pressure controller will be implemented. The concrete implementation of these baseline models is also defined in Chapter 5.

4.4 Training

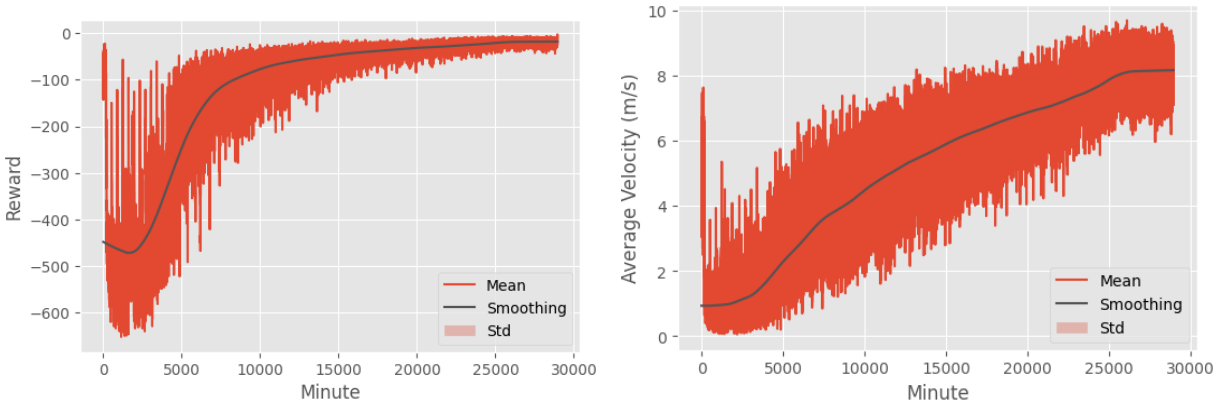
During training, several simulation and algorithm specific procedures depend on random number generators. Simply changing the random seed of said generators can cause statistically significant differences in the performance of the implemented traffic controllers. Because of this variance, to obtain performance results that reflect how the traffic controller truly performs, multiple independent training runs are seeded for every experiment and the results are then averaged for each controller. This random seeding also makes every experiment fully replicable.

Since the learning methods have exploration and exploitation phases, during the simulation, a gridlock can happen in the network, preventing the vehicles to move through the road network. This can

happen more often during the agent's exploration phase, where actions are selected randomly. When a gridlock happens, the agents stop learning and the simulation essentially halts. To avoid gridlocks, the reinforcement learning task is made episodic. The simulator is reset after a set time to ensure that unfavorable outcomes do not carry on indefinitely.

There are two key performance parameters and two auxiliar performance parameters: The rewards must increase during training, because the agents are able to make better decisions and the loss, for deep reinforcement learning models, such as DQN (Eqn. 2.7), that indicates that the policies generated are stable, is getting close to a steady state regimen. The other two auxiliar metrics are the average number of vehicles in the road network and the average speed. As training progresses agents should be able to make better decisions, reflecting on a decrease on the average number of vehicles in the network, as they disperse faster and their average speed increases.

If rewards fail to increase or loss does not drop significantly then it's not worth to proceed to the evaluation phase and the RL method must be re-configured (dashed arrow 1 in Figure 4.1).



(a) Mean Rewards for the continuous MARLIN algorithm during training.

(b) Mean Speed for the continuous MARLIN algorithm during training.

Figure 4.3: Training metrics of the continuous MARLIN algorithm during training.

4.5 Evaluation

Similar to the training phase, every evaluation needs to have multiple independent runs to allow for significant results and reproducibility.

As discussed in Section 2.1.2, the key metric used during evaluation is the average travel time. Agents usually cannot observe the travel time instantaneously – as it depends on the route the vehicles are traveling, on complex inter-vehicle dynamics and on future decisions made by other agents along their route, thus, additional metrics, such as, vehicle speed and number of stops are also evaluated for each model.

4.5.1 Hyper-parameter tuning

While the training evaluations determine if the agents are learning, the objective of the evaluation run is to confirm that the average behavior of the policies of a given model is satisfactory. It's often the case that, during this evaluation, many rounds of fine tuning must ensue to arrive at a satisfactory parametrization.

Similar to the training run, if the travel time observed during the evaluation run of each method is unreasonable, the RL method must be re-configured again (dashed arrow 2 in Figure 4.1). After the RL methods are stabilized, the results of the best models can now compared and analysed.

Refer to Appendix A for a complete list of the fine-tuned hyper-parameters that were found during this intermediate evaluation runs.

4.5.2 Performance analysis and comparison

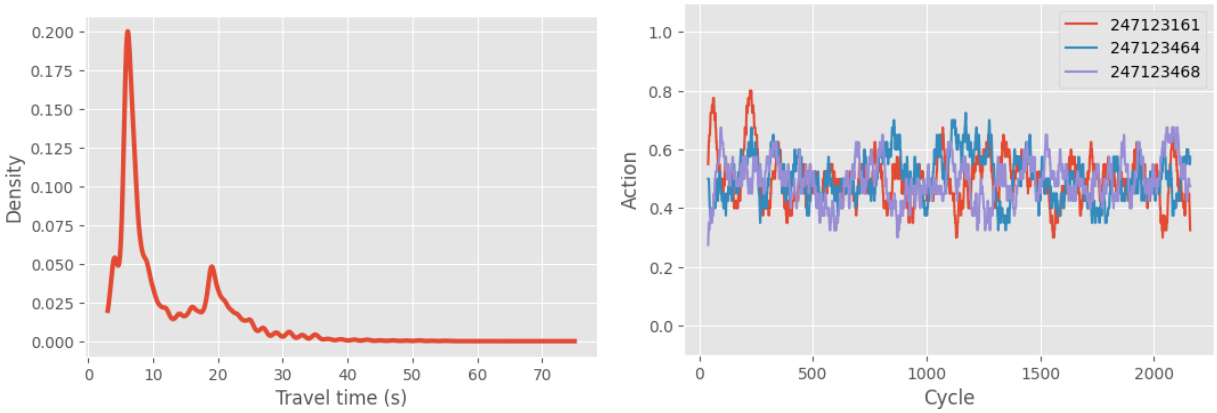
To compare the different baseline and RL-based controllers, metrics such as, rewards, speeds, number of stops, travel time, number of vehicles and waiting time are measured.

First, these metrics are compared by analysing their mean and standard deviation values. This analysis provides an overview of the performance of each traffic controller.

Secondly, the distribution of these metrics can also be explored to determine the significance of the previous analysis and to determine some controller properties, such as, fairness. For example, one traffic controller might learn a policy where the road with most lanes gets the maximum green time and conversely, the road with least lanes gets the minimum green time. This policy might obtain a reasonable average travel time, but it is very unfair.

The policies can also be analysed by plotting the average actions of each agent over a series of time steps. This describes the frequency that each intersection changes phases.

A concrete analysis of the results of the implemented controllers using the above methods is done in Chapter 6.



(a) Travel time histogram for the DQN model in the arterial network.

(b) Action plot for the random model in the arterial network.

Figure 4.4: Example of travel time distribution and policy analysis.

Chapter 5

Experimental Setup

For this work, 5 RL-based algorithms and 4 baseline algorithms were evaluated and compared, as defined in Section 4.3. The following sections define the specific simulation setup and implementation details for each RL-based and baseline algorithms. For the complete list of parameters used in training and evaluation or other algorithm specific parameters, refer to Appendix A.

5.1 Simulation Setup

First, the specifics regarding the traffic simulation, network topology and traffic demands are defined.

5.1.1 Traffic Simulator

Regarding the simulation, for the aperiodic controllers (all except static and Webster), there is a minimum of 5 seconds and a maximum of 85 seconds of green time and a fixed yellow time of 5 seconds. This means that regardless of the agent's action, the phase will always remain the same until it reaches a duration of 10 seconds (5 yellow + 5 green) and it will always switch after it reaches a duration of 90 seconds (5 yellow + 85 green).

Every simulated time step corresponds to one second. However, the learning agents only observe and take decisions every 10 seconds (or time steps). This is due to the fact that agents can take an optimal action at a certain state but still receive a bad reward since the vehicles haven't yet moved out of the intersection, due to the observation rate being too frequent or the yellow light being in effect. It is also worth noting that, while the agents decide every 10 seconds, the control actions of every traffic light is enforced every 5 seconds, to allow traffic lights to switch from and to yellow, without having any learning agent involvement.

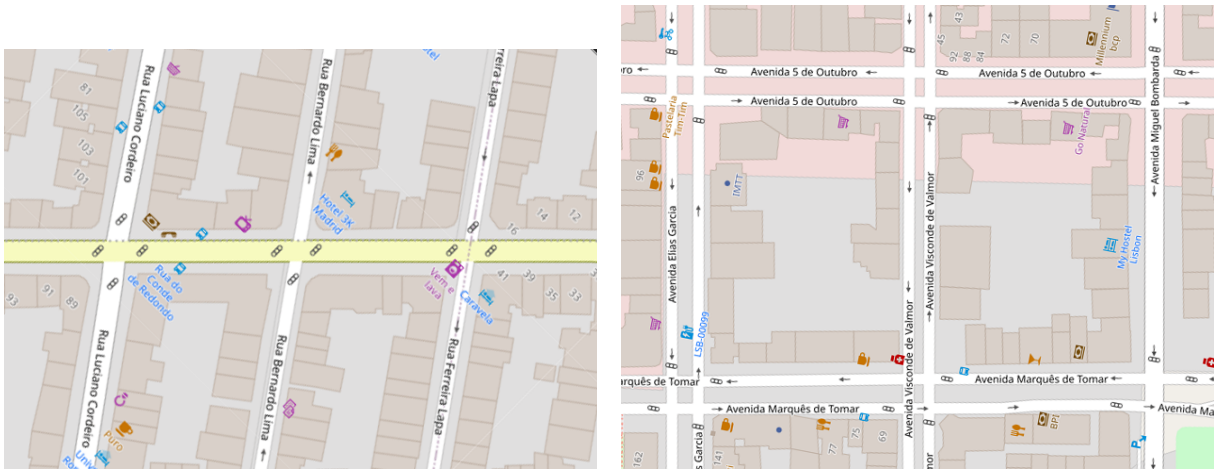
Using this implementation, the agents are able to perform actions and observe the direct outcome of their actions in the next observed state. In the particular case of switching phases, it allows an agent to observe, send the switch signal and then have 5 seconds of yellow light and 5 seconds of green light, that allows vehicles to progress through the intersection.

5.1.2 Road networks topology

For this works' experiments, two scenarios depicting different types of road networks will be considered. However, using any other arbitrary network is easily done by following the steps mentioned in Section 4.1.2 for that particular network. Both networks used in this work were extracted from real-world locations, more specifically, from the city of Lisbon.

The first network is an arterial road, 1x3 road mesh, depicted on Fig. 5.1, characterized by having the most traffic in the horizontal direction. This is located near the area of Saldanha and Marquês de Pombal.

The second is a 3x2 network grid, also depicted on Fig. 5.1, located slightly north from the first network, next to Campo Pequeno and Instituto Superior Técnico.



(a) The arterial road network.

(b) Grid network with 6 traffic controllers.

Figure 5.1: Networks used in experiments.

All traffic lights in both networks are modeled with two phases: One allows right-turn and through for vertical movement and the other allows right-turn and through for horizontal movement.

5.1.3 Traffic demands and routes

The traffic demands fed to the traffic simulator are synthetic, with constant weighted route emissions probabilities, where routes with large avenues receive greater weights.

The routing is done according to the following guidelines:

- They are the shortest paths connecting incoming and outgoing edges from the road network.
- If a generated path results in at least a loop, where the vehicle needs to change cardinal directions at least three times, the path is excluded from the final selection.

This filtering is necessary since those paths are extremely unlikely to happen in real life and may cause simulation gridlocks.

5.2 Training and Evaluation

As referred in Section 4.4, the training and evaluation phases depend on specific random number generators. Thus, to obtain statistically significant results, every algorithm is trained using 30 independent runs with set, unique random seeds. During training, every run is made episodic to avoid gridlocks and other performance issues. Each episode runs for a total of 6 hours (21,600 seconds) and every individual run trains for 80 episodes, totaling a simulated training time of 600 days.

To evaluate the algorithms, every training run performed is tested during one episode's length (6 hours) and also with set, unique random seeds, totaling a simulated evaluation time of 30 episodes (7.5 days).

5.3 Reinforcement learning methods

The five RL methods considered are: an Actor-Critic based model, a DQN based model and three variations of a fictitious play based model.

The exploration-exploitation phase of all RL-based algorithms is balanced using an ϵ-greedy policy, where ϵ, the variable responsible for performing random action selection, decreases from 0.9 to 0.01 linearly during training, achieving the lowest value at the second to last episode.

All learning algorithms use the MDP formulation defined in Section 4.2. However, some specific algorithms require slight adjustments, such as, state discretization for tabular algorithms. These adjustments are also specified in the following sections.

5.3.1 Actor-critic

An Actor-critic algorithm, following the implementation of Aslani, *et al.* [9], with a 0.9 critic learning rate (α), a 0.3 actor learning rate (β) and a trace decay (λ) of 0.55.

Since part of the state defined in the original MDP in Section 4.2 is continuous and this is a tabular method, some discretization needs to be done. This is achieved by using tile coding, with one tilling having 5 tiles, where each state variable is partitioned into a set of tiles and then the tiling is created by combining the tiles in each state variable in a vector. For example, an observation of (1, 40, 5.0, 20.0) would become (1,2,1,4) after discretized.

5.3.2 DQN

The DQN method trains a group of independent learners, with one agent per intersection, without any method of coordination.

Every agent has a fully connected neural network with 3 layers that receives the agent's state as input and outputs the Q-values for every action for that same agent. The hidden layer has a ReLU activation function and contains 32 units. The loss of every backwards-pass is calculated using the mean squared

error (squared L2 norm), the reward used is the delay for that intersection, as defined in Section 4.2, and the optimization is done by the Adam algorithm.

The networks also use some modifications, such as:

- A warm-up period of 10000 steps where random actions are taken to initialize the weights.
- Double Q-learning, that counteracts overestimation problems with the traditional Q-learning, using a synchronization rate of 500 time steps.
- Experience replay, making previous experiences more efficient and better model convergence since the data is more i.i.d. (independent and identically distributed), which is assumed in most learning convergence proofs. The replay buffer used holds a maximum of 50000 experiences.
- Batched learning, where every learning step is done in batches of 1000 experiences, taken from the replay buffer.

5.3.3 MARLIN

The MARLIN algorithm, implemented by El-tantawy, *et al.* [8], is based on the fictitious play model, where each agent plays a game with all its adjacent agents. In MARLIN, every agent saves a Q-table and a policy estimation for every neighbour. The Q-table receives the concatenated state of both the agent and the neighbour as input and outputs the Q-values for every combination of joint-actions between the agent and the neighbour. In this specific case, it receives 8 features, 4 from each agent and outputs 4 Q-values, one for each possible binary action of both agents. The policy estimation is a simple frequency count of the neighbours actions for every pair of the agent's and neighbours concatenated states. The specifics of this model are detailed in Section 2.3.1.

Since the original algorithm uses the number of waiting and approaching vehicles (WAVE), a discrete state space, some modifications need to be made to apply MARLIN in the MDP formulation defined in this work.

Three different variations were trained and tested: A rounded delay implementation, a tile coding implementation and a continuous implementation.

Rounded Delay

The first method simply transforms every phase delay into the nearest integer, the algorithm then runs just like the original, as defined before.

Tile Coding

The second method discretizes the state space by using the tile coding method applied in the Actor-critic algorithm defined above, also using 5 tiles.

Continuous

The third method uses the continuous state space directly, replacing the tabular Q-learning and policy estimator with deep Q-networks. Each agent now has two neural networks for each of its neighbours. For the Q-values, similar to the tabular version, the first neural network receives the concatenated state as input and outputs the Q-values for every possible joint action of the two agents. The reward given to the network is the mean reward of both agents. This network is similar to the one described in the DQN algorithm. It uses the same modifications, hidden layers and loss function.

The policy estimation is accomplished using a binary classifier with a network similar to the previous. This network also receives the concatenated state as input and outputs the expected probability of every action for the neighbour, using the Binary Cross Entropy as loss.

All of the networks, including the policy estimation networks, have parameter sharing, meaning they share the same neural network parameters. This technique allows the agents to learn at a faster rate with a fewer number of total experiences.

The action selection that is done after training remains unchanged, as defined in Section 2.3.1.

5.4 Baseline methods

A few baseline models will also be implemented to compare against the RL-based methods. Specifically, a random controller, a fixed time controller, a Webster controller and a Max-pressure controller will be implemented.

5.4.1 Random

The random controller simply chooses a random action at every decision point. It follows the same action restrictions defined in Section 5.1.1 that are applied to the learning methods.

5.4.2 Static

The static controller follows a pre-defined phase plan, alternating green and red lights in fixed intervals. This controller is periodic, requiring a fixed cycle length, a fixed yellow time and the decisions are made every time step, instead of every 10 time steps.

For this experiment, the cycle time was set to 60 seconds, yellow was set to 6 seconds and the specific phase plans executed were calculated using the global timings of the Webster method defined in the next Section.

5.4.3 Webster

Similar to the static controller, the Webster method is also aperiodic, requiring a fixed cycle and yellow time.

The Webster method observes the environment for a period of time, calculates the durations for every phase given the fixed phase time using the number of vehicles in each lane and executes that phase plan during the next observation period, where new data is received to calculate the next phase durations. This algorithm assumes that, while the data is being collected, the traffic flow is uniform.

The cycle time was set to 60 seconds, the yellow time to 6 seconds and the data aggregation period to 600 time steps (10 minutes).

After the algorithm finishes, the phase duration calculation can be performed on all the data collected previously. These durations are called *global timings* and they are the phase plan used in the Static controller defined in the previous Section.

5.4.4 Max-Pressure

The Max-Pressure algorithm defines a new feature labeled ”pressure”. The pressure of a phase is defined as the difference between the queue length of incoming and outgoing lanes. This algorithm maximizes the throughput of the system if the pressure of the phases is minimized. Similar to the learning methods, this algorithm also follows the action restrictions defined in Section 5.1.1.

5.5 Software

As mentioned before, the traffic simulation is done by the CityFlow micro-simulator[1] [54]. The tensor computation used in the deep RL methods is done by the PyTorch library [56] and the extra data management that is not handled by PyTorch is done using the numpy [57] and pandas [58] libraries.

The Tensorboard [59] toolkit was also used to provide measurements and visualizations of the learning algorithm’s during the training phase.

[1] `https://cityflow-project.github.io`

Chapter 6

Results

This chapter will present and discuss some of the results obtained from the experiments of the methods defined in the previous Section in both the arterial and grid networks. For a complete list of results refer to Appendix B.

6.1 Arterial network

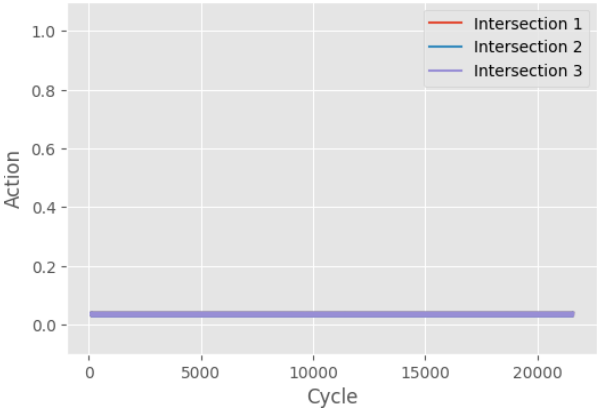
The first network is an arterial network with 3 intersections, represented in Figure 5.1. The network topology causes an imbalance of traffic flow since the horizontal road has more lanes than the other three and the demands are created depending on the number of lanes of each road.

Method	Travel Time	Speed
Random	17.693±18.127	7.485±3.858
Static	13.882±12.585	8.644±3.660
Webster	13.665±12.453	8.670±3.647
Max-Pressure	**10.872±8.917**	**8.867±3.004**
ACAT	13.976±15.210	8.277±3.441
DQN	11.716±7.861	8.417±3.100
MARLIN-Rounded	12.622±12.230	8.358±3.277
MARLIN-TileCoding	**10.788±7.290**	8.505±2.924
MARLIN-Continuous	10.921±6.856	**8.631±2.879**

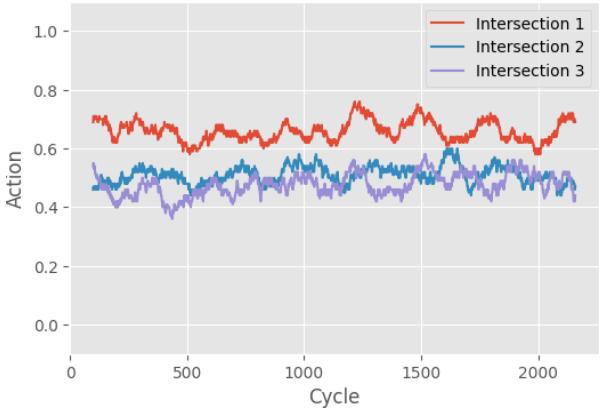
Table 6.1: Arterial network results.

Regarding the baselines, every traffic controller outperforms the random controller, as it is expected. The Max-Pressure controller greatly outperforms the static and Webster controllers in every metric. Since both the static and Webster controllers are periodic, they require a fixed cycle length (in this work, 60 seconds) and are thus less reactive when compared to the Max-Pressure or RL-based controllers. Since these aperiodic controllers follow the keep/change action schema, they can cycle through the two phases more frequently and vehicles are able to cross the intersections faster, being able to reach cycle times of 20 seconds. These can be easily visualized in Figure 6.1 that depicts the average action taken

by each agent in the Webster and Max-pressure controllers during evaluation.



(a) Average action taken by each agent for the Webster controller.



(b) [Average action taken by each agent for the Max-pressure controller.

Figure 6.1: Average action taken for the Webster and Max-pressure controllers in the arterial network, where action 1 represents changing to next phase and action 2 represents keeping the current phase active.

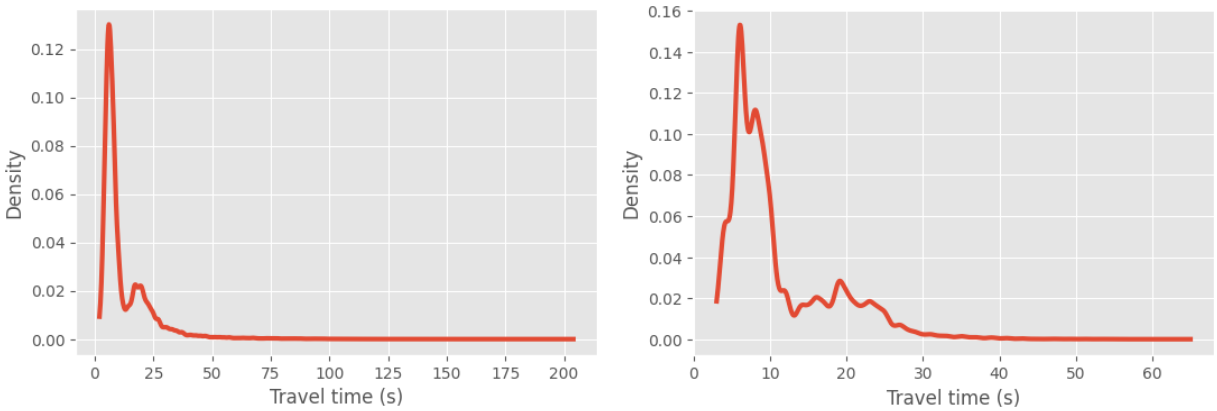
First, a comparison between the performance of the models that have different discretization methods can be done both in the independent and coordinated learners.

When comparing the independent learners, the DQN model achieves an average travel time $\approx$ 2.3 seconds lower than the Actor-critic model. One factor that can influence this difference is the fact that the Actor-critic controller uses a discrete discretization method (tile coding) to the state space, while the DQN controller uses the continuous state directly, applying a non-linear approximator (neural network) to estimate the Q-values.

When comparing the different coordinated learners, the MARLIN implementation that uses rounded delay performs much worse than the other two, while the tile coding and continuous implementations have similar performance. Since the rounded delay implementation simply transforms the continuous state space by rounding the delay to the nearest integer, the resulting state space is too big for the agents to learn effectively. This is due to the coordination mechanism applied that requires one Q-table for every unique link, each indexed by the concatenated state of two agents, where each table cell contains the Q-values for the joint action of both agents. For example, in the arterial network, where there are 4 unique links, 10 possible phase durations, the delay ranges between 0 and 21 in the first phase and between 0 and 42 in the second phase and there are 4 possible joint actions between two agents, the algorithm needs to learn 4,978,713,600 different Q-values, a much higher value when compared to the tile-coding implementation with 5 tiles that only needs to learn 1,000,000 different Q-values. Figure 6.2 shows the density of the travel time obtained in the MARLIN-Rounded and the MARLIN-Continuous implementations.

Thus, we can conclude that, regardless of the method of coordination used, the tabular methods have higher average travel time and lower average vehicle speed when compared to methods that use non-linear approximators, such as deep neural networks.

Secondly, a comparison between the performance of the models that have different coordination mechanisms (or lack of) can be done in the RL-based controllers.



(a) Travel time histogram for the MARLIN-Rounded implementation.

(b) Travel time histogram for the MARLIN-Continuous implementation.

Figure 6.2: Travel time histogram for the MARLIN-Rounded and MARLIN-Continuous implementation.

Even with the size of the state space employed by the MARLIN-Rounded controller, the model still outperformed its non-coordinated counter-part, the Actor-critic controller, by ≈ 1.36 seconds lower average travel time. The tile coding and continuous MARLIN implementations both outperformed the DQN controller, where the tile coding MARLIN implementation achieves a lower average travel time of ≈ 0.93 seconds when compared to the DQN controller.

This shows that, in this network, the use of explicit coordination mechanisms allow controllers to have a lower average travel time and higher average speed when compared to methods that use no coordination, such as the independent learners and baselines.

It is worth noting that, even though the MARLIN implementation using tile coding outperforms every other controller, the Max-Pressure baseline has a very similar average travel time, only being ≈ 0.09 seconds higher when compared to the MARLIN controller. It also has the highest average speed when compared with any other method. This can be explained by the observability that this controller has when compared to the other controllers.

In all the other baselines controllers and independent learner controllers, the agents in each intersection act by only using information that comes from the incoming lanes of that specific intersection. For every individual intersection, the Webster controller uses the number of vehicles and the independent learners use the delay in each incoming lane. The observability of these controllers is smaller when compared to the Max-Pressure and MARLIN controllers.

The Max-Pressure controller minimizes the phase pressure, that is defined as the difference between the queue length of incoming and outgoing lanes, thus, using information not only from the incoming but also the outgoing approaches. It has also been shown in other works [48] that this baseline can outperform other baselines and even RL-based methods.

The MARLIN algorithm only uses information about the incoming approaches. However, each agent selects actions using the delay of its own intersection and the neighbouring intersections, thus, requiring agents to share information during executing but having more observability than the independent learner controllers.

We can conclude that for this network, when using function approximation methods, non-linear ap-

proximators outperform discrete approximators, applying coordination methods to the multi-agent system allows controllers to obtain better performance when compared to non-coordinated ones and finally, controllers with higher observability obtain better performance when compared to controllers that have lower observability.

6.2 Grid network

The second network is a grid-like, 2x3 mesh, that also includes roads with a different number of lanes. This network is represented in Figure 5.1.

Method	Travel Time	Speed
Random	26.388±25.875	7.103±3.798
Static	20.372±15.501	7.962±3.725
Webster	20.496±15.742	7.953±3.719
Max-Pressure	**16.043±11.212**	**8.452±3.109**
ACAT	21.093±22.830	7.783±3.451
DQN	**16.772±10.026**	**8.112±3.003**
MARLIN-Rounded*	———±———	———±———
MARLIN-TileCoding	18.191±13.946	7.915±3.241
MARLIN-Continuous	17.184±9.623	7.858±2.955

Table 6.2: Grid network results.
*Too computationally extensive to finish.

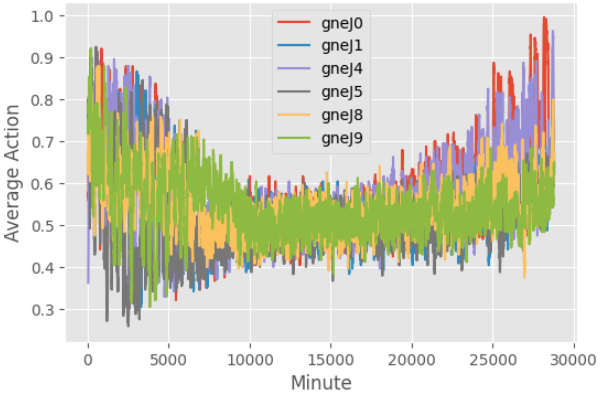
Similar to the previous network, every traffic controller outperforms the random controller and the Max-Pressure controller outperforms the static and Webster controller. Further reinforcing that controllers that are more reactive, obtain better performance when compared to less reactive controllers, such as aperiodic controllers.

Regarding the different discretization methods, the DQN model also outperforms the Actor-critic model in both average travel time and average speed, showing that a non-linear approximator is more efficient than a discrete discretization method when tested in a larger network.

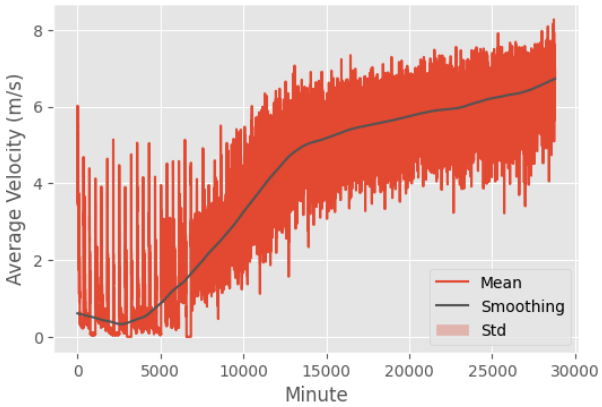
For the coordinated learners, the MARLIN implementation that used rounded delay proved to be too computationally extensive to finish the training phase. Following the same calculation done for the previous network, in the grid network, the MARLIN-Rounded controller needed to learn 59,660,697,600 different Q-values. Even though the MARLIN-Continuous implementation has a lower performance than the MARLIN-TileCoding in the simpler arterial network, when provided with a bigger and more complex environment, such as the grid network, the MARLIN-Continuous achieves an average travel time ≈ 1 second lower than the MARLIN-TileCoding implementation, showing that the use of a non-linear approximator, such as a deep neural network, can outperform a discrete discretization method, such as tile coding, when the complexity of the environment scales.

As in the arterial network, regardless of the method of coordination used, non-linear approximators have higher performance when compared to tabular methods that use discrete function approximators.

Regarding the different coordination mechanisms in the RL-based controllers, both MARLIN implementations outperform the Actor-critic controller. However, the DQN controller outperforms both MARLIN implementations, achieving an average travel time ≈ 0.41 seconds lower than the MARLIN-Continuous implementation. This discrepancy might be explained due to the lack of training steps on a more complex network. For example, by analysing Figure 6.3, displaying the average actions taken and average vehicle speeds on the MARLIN-TileCoding controller during training, we can observe that the policy is still very unstable and the average speeds are steadily increasing when the model finishes the training phase.



(a) Average actions taken by the MARLIN-TileCoding model during training in the grid network.



(b) Average vehicle speed in the MARLIN-TileCoding model during training in the grid network.

Figure 6.3: Average Action and vehicle speeds in the MARLIN-TileCoding implementation during training in the grid network.

In this grid network, the Max-Pressure baseline manages to outperform every other baseline and RL-based controller in every metric except throughput. Achieving an average travel time of ≈ 0.73 seconds lower than the best independent learner and an average travel time of ≈ 1.14 seconds lower than the best coordinated learner. Showing that higher observability, achieved in this particular controller by using environment information extracted from the outgoing lanes, leads to better performance.

Concluding, in this network, non-linear approximators outperform discrete approximators, regardless of the coordination method used. The relevance of the coordination mechanisms was inconclusive due to the fact that the learning models did not train enough. And controllers with higher observability obtain better performance when compared to controllers with lower observability.

Chapter 7

Conclusion

This thesis provides a comparative study of varied traffic controllers with different discretization methods, coordination mechanisms and observability scopes, under a homogeneous environment and a fixed MDP formulation. Every controller follows a multi-agent approach, where each agent controls one intersection by observing a partial environment state and choosing to maintain the active green phase or to switch to the next phase. This work compares a group of baselines and RL-based controllers, specifically, two independent learners using an Actor-critic model and a DQN model, three different implementations of the MARLIN algorithm and five baselines to compare to, including, a random controller, a static controller, a Webster controller and a Max-pressure controller.

This comparative study follows a rigorous methodology, where first, the simulator is set up and the networks and traffic demands are defined, then, a fixed MDP formulation is created. Afterwards, the hyper-parameters of the RL-based methods are tuned using training and evaluation phases. Finally, after being tuned, they are compared between each other and between the defined baseline controllers.

The results obtained by the current literature in the domain of RL in TLC use different methodologies, including different traffic micro-simulators, MDP formulations and result analysis. Thus, they are very hard to compare to newly created controllers or even between the existing literature. It is also hard or even impossible to fully reproduce the traffic controllers and results in some works given the vague methodology that is presented in them. Hence, following a rigorous methodology, such as the one in this thesis, contributes to the application of RL in TLC by allowing different works to be compared, fully reproduced at a later date and even possibly extended.

The simulator was set up and the process of obtaining the real-world networks used in this work is defined, step by step, so it can be fully replicated and potentially extended to any other arbitrary real-world network. Then, a fixed MDP formulation is defined following the results found by Pedro [48], where it is shown that the cumulative delay representing the state and reward is found to have good performance, when compared to other definitions, while packing the lane information into a single number, minimizing the state space. It is also shown that aperiodic controllers usually outperform periodic controllers since the former can react at a faster rate to traffic flow changes, thus, in this work a keep/change action schema is used in all traffic controllers that allow it.

After the simulation is setup, the MDP is defined and the hyper-parameters of the algorithms are tuned, the results of the evaluations were compiled and compared in two distinct networks, a 1x3 arterial network and a 2x3 grid-like network, focusing on three different axes: The function approximation method used. The coordination mechanism used. And finally, the observability that each controller has.

Results show that, when comparing the function approximation method used, in both networks, controllers that use non-linear function approximators, such as deep neural networks, achieve higher performance than controllers that use discrete discretization methods, such as tile coding, regardless of the coordination method used. These controllers achieve up to $\approx 25\%$ lower average travel time, when compared to its discrete counterparts. It is also shown that for the smaller network, the tile coding and continuous implementations of the MARLIN algorithm have similar performance, but for the bigger, more complex network, the continuous implementation outperforms the discrete tile coding implementation.

When comparing uncoordinated and coordinated controllers, for the first network, the coordinated MARLIN controller that uses tile coding is the best performing controller, followed by the Max-Pressure baseline and the continuous MARLIN controller, showing that a controller that uses an explicit coordination mechanism can outperform every baseline and every independent learner that was implemented, being able to achieve an average travel time of $\approx 8\%$ lower when compared to the best independent learner. For the second network, the coordinated methods only outperformed one of the two independent learners due to lack of training steps.

Finally, the observability of the methods tested in this work can be separated in four classes: Methods that use no information. Methods that use, for each intersection, information about the incoming approaches of that intersection. Methods that use, for each intersection, information about the incoming and outgoing approaches of that intersection. And finally, methods that use, for each intersection, information about the incoming approaches of that intersection and neighbouring intersections.

It is shown that methods that use no environment information, such as the Random controller, perform the worst, as is expected, followed by the aperiodic controllers and independent learners, that use the incoming approaches of a single intersection. The best performing controllers are the Max-Pressure controller and the MARLIN controller, that use, information about the outgoing approaches and neighboring intersections, respectively. Thus, results show that, controllers with higher observability can obtain better performance when compared to controllers with lower observability.

It is also worth noting that, while micro-simulators are used to develop RL-based traffic controllers and provide a start to improve urban mobility, there are a multitude of factors, that mostly regard safety and explainability, that need to be taken into account when moving from a simulator to the real-world. For example, simulating vehicle collisions, different weather types, pedestrian crossing, sensor noise and parking lanes are factors that could heavily influence the traffic environment that are not being currently simulated in this work.

7.1 Future Work

For future work, a more extensive analysis of the policies learned by the agents could be done, including Q-value and maximizing action graphs over the state space. For the non-coordinated algorithms, by locking the active phase and its current duration, the q-values or actions can be plotted in two dimensions (the phase delays). For the MARLIN algorithm where each agent selects an action based on its own state and the state of all its neighbours, the multi-dimensional input could be analysed using PCA, a predictive decision tree based on Q-values or an Andrews curve plot.

Different learning approaches regarding the coordinated algorithms could also be done, such as, more training time or a different exploration-exploitation approach for the continuous MARLIN implementation, since it has a non-linear classifier that attempts to predict the actions taken and with the current ϵ-greedy approach, during most of the experiment, the classifier is attempting to learn from a random action selector.

There is also a multitude of factors that could be varied to assess the performance of the presented controllers in different environments, such as: A variable traffic flow, emulating the real-world spectrum between free-flow and rush hour, congested traffic. All the safety factors mentioned previously (vehicle collisions, weather types, pedestrian crossing, sensor noise and parking).

Bibliography

[1] J. M. Sussman. *Perspectives on Intelligent Transportation Systems (ITS)*. Springer, 2005. ISBN 978-0-387-23257-7.

[2] F. Webster. Traffic signal settings. Technical Report 39, British road res. Lab., 1958.

[3] 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*, 99:144 – 163, 2019.

[4] I. Ivanova, M. Stefanov, J. Verity, N.-E. Brokopp, E. Grassi, P. M. Mula, P. Pesce, A. Zych, and M. Byalkova. Sustainable Urban Mobility in the EU: No substantial improvement is possible without Member States' commitment. Technical report, European Court Of Auditors, 2020.

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

[6] 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*, 2018:1–12, 2018.

[7] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[8] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad. Multiagent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers: Methodology and Large-Scale Application on Downtown Toronto. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1140–1150, 2013. ISSN 1524-9050, 1558-0016. doi: 10.1109/TITS.2013.2255286.

[9] 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*, 85:732 – 752, 2017.

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

[11] L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Trans. Syst., Man, Cybern. C*, 38(2):156–172, Mar. 2008. ISSN 1094-6977, 1558-2442. doi: 10.1109/TSMCC.2007.913919.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[13] C. Watkins. *Learning From Delayed Rewards*. PhD thesis, King's College, 1989.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

[15] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition independent decentralized markov decision processes. *J. Artif. Int. Res.*, 22(1):423–455, Dec. 2004. ISSN 1076-9757.

[16] C. Guestrin, D. Koller, and R. Parr. Multiagent Planning with Factored MDPs. 14:1523–1530, 2021. URL https://proceedings.neurips.cc/paper/2001/hash/7af6266cc52234b5aa339b16695f7fc4-Abstract.html.

[17] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. volume 1, pages 133–139, 01 2005.

[18] N. Ono and K. Fukumoto. A modular approach to multi-agent reinforcement learning. In G. Weiß, editor, *Distributed Artificial Intelligence Meets Machine Learning Learning in Multi-Agent Environments*, pages 25–39, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69050-4.

[19] R. P. Roess, E. S. Prassas, and W. R. McShane. *Traffic Engineering*. Pearson/Prentice Hall, 2004. ISBN 978-0-13-191877-1.

[20] J. Little, M. Kelson, and N. Gartner. MAXBAND: A Program for Setting Signals on Arteries and Triangular Networks. 795:40–46.

[21] N. M. Rouphail, B. B. Park, and J. Sacks. Direct Signal Timing Optimization: Strategy Development and Results.

[22] P. R. Lowrie, Roads and Traffic Authority of New South Wales, and Traffic Control Section. *SCATS, Sydney Co-Ordinated Adaptive Traffic System: A Traffic Responsive Method of Controlling Urban Traffic*. Roads and Traffic Authority NSW, Traffic Control Section.

[23] M. Tubaishat, Y. Shang, and H. Shi. Adaptive Traffic Light Control with Wireless Sensor Networks. doi: 10.1109/CCNC.2007.44.

[24] P. Mirchandani and L. Head. A real-time traffic signal control system: Architecture, algorithms, and analysis. 9(6):415–432. ISSN 0968-090X. doi: 10.1016/S0968-090X(00)00047-4. URL https://asu.pure.elsevier.com/en/publications/a-real-time-traffic-signal-control-system-architecture-algorithms.

[25] H. Liu, J.-S. Oh, and W. Recker. Adaptive Signal Control System with Online Performance Measure for a Single Intersection. 1811(1):131–138. ISSN 0361-1981. doi: 10.3141/1811-16. URL https://doi.org/10.3141/1811-16.

[26] K. Khiang, M. Khalid, and R. Yusof. Intelligent Traffic Lights Control By Fuzzy Logic. 9:29–35.

[27] J.-h. Lee, K.-m. Lee, K. Seong, C. Kim, and H. Lee-kwang. Traffic Control Of Intersection Group Based On Fuzzy Logic. In *In Proceedings of the 6th International Fuzzy Systems Association World Congress*, pages 465–468.

[28] M. Wiering. *Multi-Agent Reinforcement Learning for Traffic Light Control.*

[29] M. Wiering, J. Veenen, J. Vreeken, and A. Koopman. Intelligent Traffic Light Control. .

[30] M. Wiering, J. Vreeken, J. Veenen, and A. Koopman. *Simulation and Optimization of Traffic in a City*. . ISBN 978-0-7803-8310-4. doi: 10.1109/IVS.2004.1336426.

[31] A. Salkham and V. Cahill. Soilse: A decentralized approach to optimization of fluctuating urban traffic using Reinforcement Learning. pages 531–538. doi: 10.1109/ITSC.2010.5625145.

[32] M. Khamis and W. Gomaa. Enhanced Multiagent Multi-Objective Reinforcement Learning for Urban Traffic Light Control. 1:591. doi: 10.1109/ICMLA.2012.108.

[33] T. L. Thorpe and C. W. Anderson. Traffic Light Control Using SARSA with Three State Representations.

[34] X. Zhou, F. Zhu, Q. Liu, Y. Fu, and W. Huang. A Sarsa(λ)-Based Control Model for Real-Time Traffic Light Coordination. 2014:e759097. ISSN 2356-6140. doi: 10.1155/2014/759097. URL https://www.hindawi.com/journals/tswj/2014/759097/.

[35] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad. Design of Reinforcement Learning Parameters for Seamless Application of Adaptive Traffic Signal Control. 18. doi: 10.1080/15472450.2013.810991.

[36] S. Touhbi, M. A. Babram, T. Nguyen-Huu, N. Marilleau, M. L. Hbid, C. Cambier, and S. Stinckwich. Adaptive Traffic Signal Control : Exploring Reward Definition For Reinforcement Learning. 109: 513–520. ISSN 1877-0509. doi: 10.1016/j.procs.2017.05.327. URL http://www.sciencedirect.com/science/article/pii/S1877050917309912.

[37] E. Camponogara and W. Kraus. Distributed Learning Agents in Urban Traffic Control. 2902:335. doi: 10.1007/978-3-540-24580-3_38.

[38] W. Genders and S. Razavi. Using a Deep Reinforcement Learning Agent for Traffic Signal Control.

[39] T. Chu, J. Wang, L. Codecà, and Z. Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1086–1095, 2020. doi: 10.1109/TITS.2019.2901791.

[40] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis. *Multiagent Reinforcement Learning for Urban Traffic Control Using Coordination Graphs*. doi: 10.1007/978-3-540-87479-9_61.

[41] M. Steingrover, R. Schouten, S. Peelen, E. Nijhuis, and B. Bakker. *Reinforcement Learning of Traffic Light Controllers Adapting to Traffic Congestion.*

[42] E. van der Pol. Deep Reinforcement Learning for Coordination in Traffic Light Control (MSc thesis).

[43] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *NIPS*, 2016.

[44] H. Wei, N. Xu, H. Zhang, G. Zheng, X. Zang, C. Chen, W. Zhang, Y. Zhu, K. Xu, and Z. Li. CoLight: Learning Network-level Cooperation for Traffic Signal Control. In *Proc. of the 28th ACM International Conference on Information and Knowledge Management*, pages 1913–1922, 2019. doi: 10.1145/3357384.3357902.

[45] G. Zheng, Y. Xiong, X. Zang, J. Feng, H. Wei, H. Zhang, Y. Li, K. Xu, and Z. J. Li. Learning phase competition for traffic signal control. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[46] J. Isa, J. Kooij, R. Koppejan, and L. Kuijer. Reinforcement Learning of Traffic Light Controllers Adapting to Accidents. page 14, Februrary 2, 2006. doi: 10.1.1.386.9994.

[47] M. Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *ICML*, 1993.

[48] P. P. Santos. Trafic light control using deep reinforcement learning. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, Jan. 2021.

[49] G. S. Varela, P. P. Santos, A. Sardinha, and F. S. Melo. A methodology for the development of rl-based adaptive traffic signal controllers. *arXiv:2101.09614*, 2021.

[50] 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*, 18(3):227–245, 2014. doi: 10.1080/15472450.2013.810991. URL `https://doi.org/10.1080/15472450.2013.810991`.

[51] P. P. Santos, G. S. Varela, A. Sardinha, and F. S. Melo. Reinforcement learning-based adaptive traffic signal control: A critical survey. *IEEE*, 2021.

[52] G. D. B. Cameron and G. I. D. Duncan. PARAMICS—Parallel microscopic simulation of road traffic. 10(1):25–53. ISSN 1573-0484. doi: 10.1007/BF00128098. URL `https://doi.org/10.1007/BF00128098`.

[53] M. Behrisch, L. Bieker-Walz, J. Erdmann, and D. Krajzewicz. *SUMO – Simulation of Urban MObility: An Overview*, volume 2011. ISBN 978-1-61208-169-4.

[54] H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, and Z. J. Li. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. *The World Wide Web Conference*, 2019.

[55] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[56] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[57] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[58] T. pandas development team. pandas-dev/pandas: Pandas, Feb. 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

[59] 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. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Appendix A

Experiment Hyper-parameters

The tables bellow show the hyper-parameters used in the training and evaluation phases and for the baseline and RL algorithm that were optimized using the Evaluation phase described in Section 4.5.

A.1 Training

All algorithms were trained using the following parameters:

Parameter	Value
Num. runs	30
Experiment time	1,728,000 Secs. (20 days)
Episode Length	21,600 Secs. (6 hours)
Num. of episodes	80
Initial ϵ (epsilon)	0.9
Final ϵ (epsilon)	0.01
Epsilon scheduled time	1,706,400 Secs. (19,75 days)
Discount factor (γ)	0.98
Random Seeds	0, 10, 20, . . . , 290

Table A.1: Complete training Parameters.

A.2 Evaluation

All RL algorithms were evaluated using the following parameters:

Parameter	Value
Num. runs	30
Experiment time	21,600 Secs. (6 hours)
Num. of episodes	1 per run
Random Seeds	1, 11, 21, . . . , 291

Table A.2: Complete evaluation Parameters.

A.3 Baseline specific

Parameter	Value
Fixed cycle time	60
Yellow duration	6
Agregation period	600

Table A.3: Hyper-parameters used in periodic Webster controller.

Parameter	Value
Fixed cycle time	60
Yellow duration	6

Table A.4: Hyper-parameters used in periodic static controller, using the global phase timings calculated by the Webster controller.

Parameter	Value
Decision step	10
Yellow duration	5
Min green	5
Max green	85

Table A.5: Hyper-parameters used in aperiodic Max-pressure controller, following the action restrictions applied in the RL-based controllers.

A.4 RL specific

Parameter		Value
Optimizer		Adam
Learning rate (α)		0.005
Warm up steps		10000
Batch size		1000
Replay size		50000
Target sync rate		500
	Input layer	4
Multi-layer	Hidden layer	32
Perceptron	Activation	ReLU
	Output layer	2
Loss Function		Mean Squared Error

Table A.6: Hyper-parameters used in the DQN controller, using double Q-learning, experience replay, and an ϵ-greedy policy to balance exploration-exploitation. Following the MDP formulation defined in Section 4.2. The input of the neural network is the state of an agent and the output are the Q-values for every action for that agent.

Parameter		Value
Optimizer		Adam
Learning rate (α)		0.005
Warm up steps		10000
Batch size		1000
Replay size		50000
Target network update period		500
	Input layer	8
Edge Multi-layer	Hidden layer	32
Perceptron	Activation	ReLU
	Output layer	4
Loss Function		Mean Squared Error
	Input layer	8
Policy Estimation	Hidden layer	32
Multi-layer Perceptron	Activation	ReLU
	Output layer	1
	Activation	Sigmoid
Loss Function		Binary Cross Entropy

Table A.7: Hyper-parameters used in the continuous MARLIN controller that uses DQN with double Q-learning, experience replay, and an ϵ-greedy policy to balance exploration-exploitation for both the Q-value estimation and neighbour policy estimation. Following the MDP formulation defined in Section 4.2. The input of both neural networks is the concatenated state of two agents. The output of the edge multi-layer perceptron are the Q-values for the joint action of both agents and the output of the policy estimation multi-layer perceptron is the probability of the neighbour agent selecting each of its actions.

Parameter		Value
Critic learning rate (α)		0.9
Actor learning rate (β)		0.3
Trace Decay (λ)		0.55
Tile Coding	Num. Tilings	1
	Tiles	5
	Max. Size	lanes capacity

Table A.8: Hyper-parameters used in the Actor-critic controller, using eligibility traces and tile coding.

Parameter	Value
Learning rate (α)	0.9

Table A.9: Hyper-parameters used in both discrete MARLIN controllers.

Appendix B

Complete experimental results

Method	Travel Time	Speed	Num. Stops	Throughput
Random	17.693±18.127	7.485±3.858	0.464±0.650	26690.467±10.932
Static	13.882±12.585	8.644±3.660	0.266±0.445	26508.933±581.196
Webster	13.665±12.453	8.670±3.647	0.262±0.443	26696.500±12.851
Max-Pressure	**10.872±8.917**	**8.867±3.004**	**0.219±0.432**	**26703.400±44.716**
ACAT	13.976±15.210	8.277±3.441	0.362±0.619	26697.700±0.702
DQN	11.716±7.861	8.417±3.100	0.296±0.470	26712.833±2.914
MARLIN-Rounded	12.622±12.230	8.358±3.277	0.294±0.503	26704.100±12.307
MARLIN-TileCoding	**10.788±7.290**	8.505±2.924	0.232±0.459	26710.167±2.866
MARLIN-Continuous	10.921±6.856	**8.631±2.879**	**0.227±0.444**	**26715.600±3.098**

Table B.1: Complete arterial network results.

Method	Travel Time	Speed	Num. Stops	Throughput
Random	26.388±25.875	7.103±3.798	0.690±0.897	62999.733±34.261
Static	20.372±15.501	7.962±3.725	0.396±0.505	63047.967±2.025
Webster	20.496±15.742	7.953±3.719	0.398±0.503	62964.067±104.999
Max-Pressure	**16.043±11.212**	**8.452±3.109**	**0.349±0.545**	**63078.200±41.153**
ACAT	21.093±22.830	7.783±3.451	0.574±0.876	63031.700±36.157
DQN	**16.772±10.026**	**8.112±3.003**	**0.417±0.530**	63060.167±68.848
MARLIN-Rounded*	———±———	———±———	———±———	———±———
MARLIN-TileCoding	18.191±13.946	7.915±3.241	0.463±0.641	63073.533±12.159
MARLIN-Continuous	17.184±9.623	7.858±2.955	0.473±0.563	**63082.300±14.466**

Table B.2: Grid network results.
*Too computationally extensive to finish.