Designing traffic signal control systems using reinforcement learning

Maxime Tréca

université
PARIS-SACLAY

Designing traffic signal control systems using reinforcement learning

Création de systèmes de contrôle de feux à l'aide de méthodes d'apprentissage par renforcement

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580, sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et sciences du numérique
Référent : Université de Versailles Saint-Quentin-en-Yvelines

Thèse préparée dans l'unité de recherche DAVID (Université Paris-Saclay, UVSQ), sous la direction de Dominique BARTH, professeur, le co-encadrement de Julian GARBISO, docteur et le co-encadrement de Mahdi ZARGAYOUNA, maître de conférence.

Thèse soutenue à Versailles, le 6 juillet 2022, par

Maxime TRÉCA

Composition du jury

Amal El Fallah-Seghrouchni Professeure, LIP6, Sorbonne Université	Présidente
Alain Dutech Professeur, LORIA, INRIA Nancy	Rapporteur et Examinateur
René Mandiau Professeur, LAMIH, Université Polytechnique Hauts-de-France	Rapporteur et Examinateur
Lila Boukhatem Maîtresse de Conférences, LISN, Université Paris Saclay	Examinatrice
Dominique Barth Professeur, DAVID, UVSQ	Directeur de thèse

THESE DE DOCTORAT

NNT : 2022UPASG043

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Création de systèmes de contrôle de feux à l'aide de méthodes d'apprentissage par renforcement
Mots clés : Apprentissage par renforcement, simulation de trafic, contrôle de feux, apprentissage profond.

Résumé : Cette thèse étudie le fonctionnement des systèmes de contrôle de feux de signalisation dans l'optique d'optimiser le trafic routier. Cette optimisation s'appuie sur des techniques d'apprentissage par renforcement qui modélisent un ou plusieurs agent maximisant une tâche dans un environement. Pou run état donné du système (i.e. le réseau routier), l'agent choisit une action (i.e. une configuration de feux) qui est appliquée à l'environnement dans le but de maximizer un objectif (i.e. minimiser le temps d'attente des véhicules du réseau).

Ces travaux se divisent en trois parties. Tout d'abord une partie introductive dresse l'état de l'art des différentes disciplines abordées durant le thèse, soit une présentation du contrôle de feux tel qu'il est utilisé dans le monde des transports, une présentation de l'apprentissage par renforcement en tant que branche de l'intelligence artificielle puis une présentation du domaine RL-TSC (reinforcement learning for traffic signal control), soit comment les méthodes d'apprentissage par renforcement sont appliquées au contrôle de feux.

Dans un second temps, la thèse présente les outils utilisés afin de développer des méthodes de contrôle de feux à l'aide de l'apprentissage par renforcement. Plus particulièrement, nous définissons un modèle mathématique du contrôle de feux, ainsi qu'un modèle d'apprentissage utilisé par les différents algorithmes pour apprendre dans ce contexte (i.e. définition des états de l'environnement, des actions prises par les agents et les récompenses associées). Finalement, nous présentons l'environnement d'experimentation utilisé pour l'optimisation de trafic (simulateur `SUMO`), et les ptocoles utilisés pour mesurer la performance des algorithmes d'apprentissage par renforcement utilisés dans ce contexte.

Finalement, la troisième partie de ces travaux vise à analyser et comparer diverses méthodes de contrôle de feux utilisant de l'apprentissage par renforcement. Dans un premier temps, plusieurs approches classiques de la littérature (e.g. Q-learning, LRP et méthode acteur-critique) sont appliquées sur une intersection isolée. Après avoir identifié et expliqué pourquoi les approches par valeur, comme le Q-learning, sont le plus adaptées au contrôle de feux, nous étendons cette approche à l'utilisation de réseaux de neurones profonds comme moyen d'approximation par fonction. Cette approche plus moderne est associée à une meilleure adaptabilité de l'agent et donc de meilleures performances.

Dans un second temps, nous étendons notre analyse à un cadre multi-agent. Cette approche introduit un certain nombre de contraintes supplémentaires sur l'apprentissage, comme la non-stationnarité, mais permet également aux agents de communiquer et de même se coordonner afin de mieux optimiser le traffic en l'anaysant à plus grande échelle. Plusieurs approches innovantes sont proposées dans ce cadre, comme notamment une coordination par vague verte permettant à plusieurs agents indépendants de se coordonner automatiquement le long d'une artère. Cette approche donne de meilleurs résultats que les méthodes non-coordonnées de la littérature, mais uniquement lors que le réseau n'est pas congestionné. Une seconde approche que nous avons développé, DEC-DQN, permet aux intersections d'un réseau routier d'apprendre à adapter leurs feux de signalisation en fonction du trafic en temps réel. Le point central de cette méthode est que les intersections du réseau peuvent communiquer entre elles selon un protocole qu'elles ont elle-mêmes appris, ce qui leur permet de s'adapter à plusieurs scenarios routiers sans instructions explicites. Une expérimentation sur simulateur de trafic dans une ville synthétique à grande échelle valide la performance de DEC-DQN par rapport à d'autres méthodes phrase de la littérature RL-TSC comme MARLIN. Les résultats de ces simulations indiquent la supériorité de notre approche dans plusieurs scénarios de trafic différents : trafic faible, congestionné et variable.

Title : Designing Traffic Signal Control Systems Using Reinforcement Learning
Keywords : Reinforcement learning, traffic simulation, traffic signal control, deep learning.

Abstract : This thesis studies traffic light control systems in order to optimize traffic flows. This optimization is based on reinforcement learning techniques, which model one or multiple agents maximizing a task in a given environment. This thesis defines a novel deep reinforcement learning method that allows intersections of a road network to learn to adapt their traffic light signals depending on the current state of the road network. The main contribution of this method is the ability for intersections to communicate according to a communication protocol that they learn themselves, allowing them to adapt to multiple traffic scenarios. This novel method, DEC-DQN, outperforms various coordinated deep reinforcmeent learning methods in use in the traffic signal control literature.

Designing Traffic Signal Control Systems Using Reinforcement Learning

by

Maxime Tréca

Supervisors:	Prof. Dr.	Dominique Barth	Université de Versailles Saint-Quentin-en-Yvelines
	Dr.	Julian Garbiso	Institut Védécom
	Prof. Dr.	Mahdi Zargayouna	Université Gustave Eiffel

Acknowledgements

I would like to express my appreciation to my supervisor, Dominique Barth, for his guidance during my years of research. When Dominique informed me of a research topic on traffic optimization, I had no idea that it would result in studying a subject I knew very little about–reinforcement learning–for the next three years. I thank him for this fruitful discovery. Dominique was always available for guidance and scientific discussions, no matter how busy his schedule was. However, he also let me explore my own ideas. I owe him the ability to be autonomous and curious in research, which is precious. I am proud to have him as an advisor. I would like to extend this appreciation to my two co-supervisors, Julian Garbiso and Mahdi Zargayouna. Julian has always been very generous with his time and energy, from proofreading papers before deadlines to conceptualizing novel ways of optimizing traffic. His moral support has also been vital during difficult periods of my thesis. I am happy to call him a friend. I would like to thank Mahdi for his enthusiasm, kindness, and keen eye regarding scientific publications. He was always available to give me excellent advice regarding manuscripts or model choices. I would like to thank Amal El Fallah Seghrouchni, Lila Boukhatem, Alain Dutech, and René Mandiau for participating in my thesis committee.

I met amazing people during my thesis, whether at Institut Védécom or the DAVID and GRETTIA lab. Although the simple mention of their names does not do justice to their talent or kindness, I would like to thank Alexis, Aziz, Bertrand, Bintou, Catherine, Coline, David, Fabienne, Fares, Frank, Jean-Michel, Joseph, Kartik, Leila, Loric, Mael, Mehdi, Nassim, Perla, Pierre, Safa, Sandrine, Shabbir, Tatiana, Thierry, Toussaint, Tristan, Xavier, Yacine, Yann, Ylène, Youssef and others for these three years together. I would also like to extend these thanks to my students, who kept my passion for teaching and computer science alive. I am also grateful to Université de Versailles Saint-Quentin-en-Yvelines, Institut Védécom, and Université Gustave Eiffel for financing and accompanying my research and allowing me to publish and participate in conferences. Finally, I would like to thank the thousands of people around the world who develop and maintain open-source software. More specifically, this thesis research would not have been possible without the thousands of hours of work behind the GNU/Linux, NixOS, Emacs, and SUMO projects.

My wife Marci has been an anchor during this time. I can never repay her enough for her patience, love, joy, and for the sacrifices she made for me to complete this thesis. Last but certainly not least, I am grateful for my friends and family for their continuous support.

Contents

I Introduction

The XXth century has indubitably been the century of the personal motor vehicle. In the United States, the rates of vehicle ownership per 100.000 inhabitants have soared from 0.1 in 1900 to 323.7 in 1950 and 800.3 in 2000 (Davis and Boundy, 2021). As this increasing number of vehicles started circulating on urban transportation networks, the need for traffic signal control (TSC) became apparent for two reasons. Its primary and essential goal was to guarantee the safety of road users. Its second and corollary goal was to reduce traffic congestion caused by the introduction of a large number of vehicles in urban areas. This thesis aims to leverage recent advances in machine learning to fulfill these two missions: optimizing traffic flows on a traffic network while ensuring the safety of its users.

1.1 Motivations

Until the 2000s, traffic signal control was mainly seen through the prism of traffic engineering and operational research. Early TSC solutions inspired by Webster's work (1958) provided simple fixed traffic signal settings based on historical traffic data that performed relatively well. More advanced traffic solutions such as adaptive traffic signal control (ATSC) which relies on routing traffic using real-time instead of real-time traffic data, soon followed. The recent rise of reinforcement learning (RL) and the development of novel sensor technology have provided a theoretical and practical basis for the use of machine learning for traffic signal control. The study of ATSC methods has become exceedingly popular in the literature in recent years due to these new perspectives. The reinforcement learning-based traffic signal control (RL-TSC) literature has since showcased impressive achievements in simulated traffic settings during its relatively short history. In recent years, this research has culminated with the use of TSC methods coupled with deep reinforcement learning (DRL) techniques and agent coordination. Such methods allow multiple intersections to optimize traffic flows on large simulated networks using real-world traffic data, outperforming several well-established TSC methods from the traffic engineering literature. The numerous manners in which the RL-TSC literature applies reinforcement learning on traffic optimization problems is a testament to the fact that RL is a highly well-suited tool for the control of traffic lights. This last remark should, however, not divert us from the fact that traffic signal control is much more than a simple application domain of machine learning methods, but a complex and fascinating research topic that predates the field of RL itself. It would hence be a mistake to study RL-TSC without understanding traffic engineering first.

This thesis's primary motivation is to provide a complete (but not exhaustive) study of how one can apply modern reinforcement learning techniques to the problem of traffic signal control. By first defining what traffic *is*, we were able to incrementally build a model and framework for RL-TSC while analyzing, discussing, and explaining each hypothesis, model choice, or design decision along the way. This modeling work has not only allowed us to build a state-of-the-art coordinated

traffic signal control method but can also serve as a basis for any future work on RL-TSC research in the future.

1.2 Traffic Signal Control

The task of traffic signal control, which consists in assigning a right-of-way on conflicting traffic flows over an intersection through the use of light signals, can be seen as a simple optimization problem. Each intersection of the road network aims to achieve maximum vehicle throughput on its lanes while maintaining safety constraints for road users. This optimization problem has various answers in the TSC literature depending on the characteristics of the intersection at hand. Indeed, the optimal traffic light assignment over an intersection depends on geographic constraints and traffic data accessibility. Intersections for which only historical traffic data is available will not route traffic the same way as intersections using real-time traffic data through sensors or loop detectors. Similarly, traffic routing differs depending on its scale of operation: the optimization problem is easy to solve over isolated intersections but becomes increasingly complex when spread on a larger scale, such as arterials.

Given that most state-of-the-art traffic signal control solutions, presented in chapter 2, are both *adaptive* (i.e., they can access traffic data in real-time) and *coordinated* (i.e., they optimize traffic flows on multiple intersections), we aim at developing a traffic signal control leveraging both of these features, while being able to automatically learn to route traffic using a branch of machine learning known as reinforcement learning.

1.3 Reinforcement Learning

Reinforcement learning is a class of machine learning algorithms aiming at solving tasks through reward maximization. Most RL models feature an agent interacting with an environment to solve a task. The environment goes through successive *states* (e.g., traffic congestion around an intersection) answered by the agent with an action (e.g., a traffic light setting) applied to the environment. Once the action is applied, the environment transitions to a new state, and the agent receives a *reward* value that quantifies the quality of the previous action given the task it is trying to solve. By efficiently testing state-action combinations to maximize the agent's cumulated reward signals, RL algorithms can learn an optimal *policy*, which maps optimal actions to different environment states. We present the general RL framework in chapter 3.

The use of reinforcement learning for traffic signal control tasks, which we cover extensively in chapter 4, has been increasingly popular for multiple reasons. First, the theoretical framework of reinforcement learning is a good fit for traffic signal control problems. Since learning models usually place agents at the intersection level, they can both learn in a single-agent (SARL) or multi-agent (MARL) setting since the RL framework covers both single and multi-agent learning. Second, developing RL-TSC methods is relatively easier than developing classical traffic signal control methods. Indeed, *model-free* RL methods learn from reward signals from the environment without explicitly modeling how their actions affect state transitions of the environment. In the case of traffic signal control, this implies that a model-free method can learn to route traffic without prior knowledge regarding traffic dynamics. Furthermore, the availability of open-source traffic

simulators and scientific computing programming libraries has made it easy to prototype novel RL-TSC methods and test their performance on traffic simulations.

1.4 Contributions

We have made several scientific contributions during our research work on coordinated traffic signal control methods:

- In chapter 6, we analyze the effect of different action space definitions on an intersection's overall traffic routing abilities using the Q-learning algorithm. Our experiments showed that a step-based action space definition was superior to a phase-based one. We published these results in the form of guidelines for action space definition for RL-TSC applications at the AAAI ICAPS 2020 conference (Tréca et al., 2020a).
- In chapter 7, we present a simulation library written in Python, carmulator, which has been created during our research work to quickly prototype novel RL-TSC methods and compare them to existing methods using the `SUMO` simulator. All the results obtained during this thesis work are reproducible using this simulation library released under an open-source license.
- In chapter 8, we define a novel bootstrapping method used to accelerate the convergence process of a Q-learning-based traffic signal control method on an isolated intersection. By pre-estimating the value function of each possible traffic state around an intersection using approximation results from queuing theory, we were able to drastically improve the convergence speed of a RL-TSC agent. We have presented this novel method at the IEEE VTC 2020 conference (Tréca et al., 2020b).
- In chapter 8, we develop a novel near-optimal traffic signal control method. This method features a backtracking algorithm that enumerates all the possible strategies of a traffic light controller over a given horizon by repeatedly saving and loading `SUMO` simulation states and returns the optimal one. This near-optimal method allows to set an upper performance bound on traffic simulation scenarios, which proves extremely useful in evaluating the performance of RL-TSC methods. This method is currently being submitted for publication.
- In chapter 9, we present a deep reinforcement learning algorithm for traffic signal control that relies on green wave coordination over arterial streets. This form of traffic light coordination, which, to our knowledge, has not been studied in the RL-TSC literature, outperforms standard deep Q-learning algorithms in normal traffic conditions. This method has been accepted for publication and will be presented at the TRISTAN 2022 conference.
- In chapter 9, we present a novel RL-TSC method featuring direct coordination between agents of the same road network. Agents using this direct coordination method learn a common communication protocol through a shared deep neural network and can hence coordinate automatically without human intervention. This coordination method provides excellent results, which we consider to be state-of-the-art. This method has been submitted to the NeurIPS 2022 conference.

1.5 Structure of this Thesis

This thesis aims to build a state-of-the-art coordinated traffic signal control method from the ground up. Hence, the structure of this thesis reflects the deeply iterative nature of this work, divided into three main parts.

The first part of this thesis contains a thorough review of classical traffic signal control (chapter 2) and the theoretical framework of reinforcement learning (chapter 3). These two reviews help establish the necessary terms and concepts which we use to present the field of RL-TSC (chapter 4). Following these definitions, the second part of this thesis looks at the model in use for traffic signal control. More specifically, we define a mathematical model of traffic signal control (chapter 5), which we can, in turn, use to formally determine how traffic is represented and optimized from a reinforcement learning standpoint (chapter 6). Finally, we describe the traffic simulation setup used to conduct traffic signal control experiments of this thesis (chapter 7). This thesis's third and final part defines efficient traffic signal control methods using the experimental framework described in part 2. More specifically, we first aim at comparing multiple traffic signal control on isolated intersections to establish the optimal RL-TSC method and associated parameters (chapter 8). On this basis, we extend our scope of analysis to multi-intersection networks and the study of coordination modes between these intersections (chapter 9). Finally, we summarize our main findings and present future areas of research in the last chapter of this thesis (chapter 10).

Part I

Theory

The first part of this thesis presents the context in which reinforcement learning-based traffic signal control methods were developed. Thus, this part successively introduces the field of traffic signal control (chapter 2), including its origins, main concepts and methods, and the theory of machine learning (chapter 3) by presenting its general framework and the main categories of RL algorithms. Finally, we describe how these disciplines merged to give birth to RL-TSC (chapter 4) by doing a thorough literature review of this research topic and by discussing its main challenges.

2 An Introduction to Traffic Signal Control

Traffic signal control, which is the study of the use of traffic lights to ensure the safety and efficiency of a road network, has been central in the proper management of urban mobility for more than a century. The role of this chapter is twofold. It first aims to establish a short history of the field of TSC by presenting the motivations for its inception and its pivotal role in the modernization of urban mobility while showcasing how TSC methods have considerably evolved over a century. The second objective of this chapter is to present a certain number of crucial traffic signal control concepts and notions which will be essential to the understanding of *how* these TSC methods optimize traffic before formally introducing a mathematical traffic model in chapter 5 of this thesis.

2.1 The Science of Traffic Signal Control

The birth and adoption of traffic lights is a direct consequence of the mass production of automobiles in the early XXth century. The widespread availability of the Ford Model T, which started production in 1913, caused an exponential surge in traffic congestion in most major cities in North America and Western Europe in the late 1910s and 1920s, respectively. Congestion became so problematic in some large cities that walking or taking the subway was commonly thought to be faster than using a car (Wells, 2013). These issues regarding urban mobility caused the birth of traffic signal control in the 1910s, which then developed to become an entire field of study throughout the century. This section quickly reviews the origins, terminology, and main types of methods used in TSC.

2.1.1 Origins of Traffic Signal Control

The first modern traffic light was installed in Cleveland, Ohio, in 1914 (see Figure 2.1) to modernize the existing traffic routing solutions. Traffic lights soon expanded to several major American cities to streamline heavily congested intersections.

While they reduced congestion and improved traffic safety, the first traffic lights were frowned upon by inner-city inhabitants. Since they caused an increase in traffic and average vehicular speeds in these areas (which was proof of their effectiveness), pedestrians felt safer with the use of stop signs, even though they caused more traffic accidents at the time (McShane, 1999). At first, the control of traffic at intersections was under the responsibility of the city police, either through modern traffic control systems, such as traffic lights or semaphores[1] or using direct gestures. Since they were directly operating traffic lights, police officers drove most early innovations regarding traffic signal control systems. The addition of an orange light for safety reasons in 1917 and the octagonal shape of stop signs in 1914 is a testament to this involvement. The extensive use of human intervention in routing traffic was a driving factor in the traffic light automation that soon followed. The first automated traffic light system (TLS) appeared in Houston, Texas, in 1922, and most major cities in North America and Western Europe adopted these automated systems as early as 1925. As well as cutting down operating costs of traffic light systems significantly, this innovation effectively transferred the task of using and developing TSC systems from police officers to electricians and soon-to-be traffic engineers.

Figure 2.1: The First Modern Traffic Light Control System, Installed on Euclid and 105th Avenue. Cleveland, Ohio, August 1914.

[1] Semaphores were mechanical devices with rotating `Stop` and `Go` signs giving a right of way to vehicles around an intersection.

Interestingly enough, one should note that most TLS systems developed during the 1930s and 1940s were all remarkably similar, even though they arose in different locations and no one had explicitly set standards at the time. This natural gravitation towards the same set of traffic rules partly explains why TLS systems were similar in most parts of the world by the 1960s (McShane, 1999).

2.1.2 Traffic Signal Control Terminology

Over a century, traffic signal control has evolved from an experimental technique to reduce congestion on a few intersections to an entire field of research with specific concepts and terminology. This section introduces a certain number of key concepts and terms used in the field of traffic engineering and which are crucial to understanding the challenges posed by traffic signal control. Note that this section does not extensively cover traffic engineering concepts, which one can find in multiple works in the transportation literature (Koonce and Rodegerdts, 2008; Sullivan et al., 2015; Urbanik et al., 2015), but to provide a general introduction to TSC to the reader to underline its mechanisms and challenges.

Traffic signal control is commonly applied on intersections composed of multiple entry points, also known as *approaches* (e.g. arrows on Figure 2.2). These approaches meet on the *crossing area* of the intersection, on which multiple traffic streams can cross (e.g., gray zone on Figure 2.2). A traffic stream can engage on the crossing area when it has a *right of way* over the intersection, usually given by a traffic light controller.

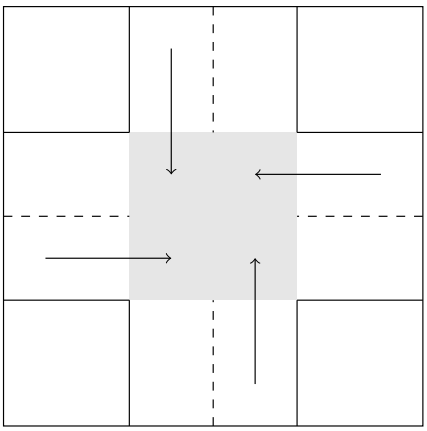

Figure 2.2: An example 4-way intersection.

Non-conflicting traffic streams that can safely and simultaneously cross an intersection can form a traffic *phase*. A *signal cycle* is a repetitive pattern of phases implemented by a traffic light controller, ensuring that all intersection traffic streams can eventually cross it. Adding constraints on the organization and compatibility of traffic streams in a signal cycle still allows for many different valid signal cycles on the same intersections. A common signal cycle pattern for 4-way intersections, known as the NEMA[2] signal cycle, is represented on Figure 2.3.

[2] *National Electrical Manufacturers Association*.

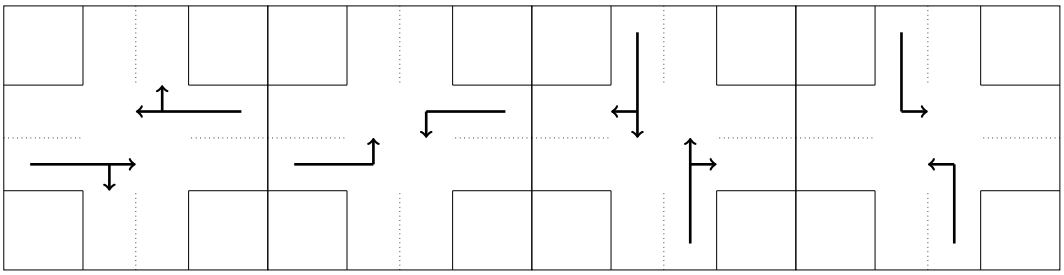

Figure 2.3: An example NEMA signal cycle on a 4-way intersection.

One should note that if the signal cycle of an intersection is the only way this intersection influences traffic, there are four main levers of action within signal cycle design that influences traffic on that intersection (Papageorgiou et al., 2004):

1. Its phase specification, or how it organizes phases within its signal cycle.
2. Its split time, or the relative duration of each phase within the signal cycle.
3. Its cycle time, which is the total duration of the signal cycle.
4. Its offsets with neighboring intersections, which can create *green waves* along the intersections of an arterial street.

2.1.3 A Typology of Traffic Signal Control Methods

All existing traffic signal control methods belong to two major categories, given their mode of operation. Depending on whether it routes traffic on a single intersection or multiple ones, a TSC method will either be *isolated* or *coordinated*. Additionally, if the traffic signal control method adapts to the current traffic state, it is defined as *adaptive*, otherwise as *fixed*. This section quickly presents each class of TSC methods according to these classes and underlines some of their advantages and limitations.

2.1.3.1 Fixed Methods

The earliest and simplest traffic light systems implement *fixed-time* signal cycles. These timing strategies assign fixed durations to each phase of the signal cycle, usually using historical traffic data, giving insight into the distribution of traffic flows over intersections. While a good first approach regarding green light time assignment is to increase the green phase length of high-demand lanes, attribution for fixed signal timing can quickly become complex (Urbanik et al., 2015). More advanced fixed-time traffic lights can also switch traffic light configurations on demand since traffic demand usually changes during the day. Fixed methods cannot adapt to traffic conditions in real-time, given their nature. These methods are easy and cheap to deploy in real-life applications but are also likely to perform relatively poorly in areas subject to high variations of traffic demand. While our primary goal is to study RL-TSC which is by nature adaptive, we present a few fixed methods such as Webster's or a near-optimal method in detail in section 8.1 of this thesis.

2.1.3.2 Adaptive Methods

Adaptive traffic signal control methods implement variable-length signal cycles that can adapt to traffic conditions in real-time by querying traffic state through the use of sensors or cameras. Adaptive methods generally provide better results than their fixed-time counterparts, as they can react to traffic demand changes in real-time. They are still rarely seen in real-world situations due to their high deployment and maintenance costs. According to the United States department of transportation, less than 1 percent of US intersections used an adaptive traffic signal control method in 2011 (USDT, 2011). While it is true that ATSC methods are more costly to deploy and are marginal in most countries, they provide several benefits compared to fixed-timing plans that would make their adoption worthwhile. Aside from the obvious fact that fixed-timing plans cannot adapt to traffic conditions in real-time, they also require regular maintenance and updates to keep track of traffic demand changes. Outdated signal timing plans are estimated to be responsible for 10 percent of total traffic delays in the US, which translates to an $8.7 billion yearly cost in fuel and productivity loss (USDT, 2011). According to A. Robertson, creator of the TRANSYT and SCOOT traffic signal control methods, the switch from fixed to adaptive methods becomes more urgent as TSC technology progresses (Robertson, 1986):

> I find it difficult to believe that, as we approach the end of this century, traffic engineers and drivers will continue to tolerate signals with green and red times that were decided by the flows and queues that happened to be observed on one day many years earlier, rather than in the last five minutes.

2.1.3.3 Isolated Methods

Isolated TSC methods, as indicated by their name, take a single intersection into account when routing traffic. They represent the majority of traffic controllers in use. On the one hand, isolated TSC methods present several advantages. They are easy to implement, offer a relatively low complexity, and are highly scalable since removing or adding an isolated traffic light on a road network has little to no incidence on the other intersections of the network. On the other hand, these methods are by nature limited since they can only act on traffic on a per-intersection basis, which limits their usefulness in highly used road networks or when traffic light coordination is desirable (Mannion et al., 2016). We analyze these methods in detail in chapter 8 of this thesis.

2.1.3.4 Coordinated Methods

Conversely, coordinated TSC methods aim to optimize traffic around a given intersection and make each intersection interact with its neighbors to some degree to optimize traffic further. The mechanisms relating to inter-intersection coordination differ depending on the TSC method. Coordinated methods allow for more complex traffic management features since they can access traffic data over larger network portions and potentially coordinate their signal cycle implementations to optimize traffic. For instance, green wave or bandwidth-based methods presented in the next section make extensive use of coordination to function correctly. Note that if these methods usually perform better than their isolated counterparts, they incur a high equipment cost since all intersections must communicate in real-time, which increases their overall complexity, limiting

their applicability. Since developing an intelligent and adaptive coordinated traffic signal control method is one of the objectives of this thesis work, these methods will be extensively discussed, in their adaptive form, in chapter 9 of this thesis.

2.2 Operation of Traffic Signal Control Methods

The previous section has shown the main motivations behind traffic signal control and how these methods can be categorized depending on their mode and scale of operation. This section broadly presents *how* traffic signal control methods optimize traffic through several historically significant TSC methods, some of which are still in use today. We present these methods according to two common ways to classify TSC methods. On the one hand, these methods differ according to their responsiveness, dividing them between fixed, actuated, and adaptive methods (Gartner et al., 1995). On the other hand, these methods can also differ according to the traffic-related metric they aim at optimizing. Some, known as *bandwidth-based*, aim at optimizing traffic flows along an arterial and are hence necessarily coordinated. Others, known as *delay-based*, aim at minimizing the average time it takes for a vehicle to exit the network. Delay-based methods have a multitude of different application settings (e.g., isolated, coordinated, fixed, actuated, adaptive) and are known to perform better than bandwidth-based ones under variable traffic flows and complex signal settings (Robertson, 1986).

2.2.1 Fixed Methods

While somewhat simple in design at first, fixed traffic signal control methods can regroup a significant number of distinct techniques and modes of operation, including isolated and coordinated methods, both using delay-based and bandwidth-based optimization objectives.

2.2.1.1 Fixed Delay-Based Methods

Regarding delay-based fixed methods, the first and significant traffic signal control method to be developed is due to Webster (1958), which studied the optimal settings of an isolated intersection depending on the traffic demand around it. Using one of the first computer traffic simulations, Webster defined a *total delay function* expressing the mean delay per vehicle as a function of the intersection's cycle time, phases, and flow values. Using Webster's formula (see section 8.1.1 for a full definition), a traffic engineer can minimize vehicular delay around an intersection (under normal traffic conditions) by setting green splits proportionally to the traffic flow within the intersection. Even though posterior works have refined it, Webster's formula is an essential foundation of the traffic signal control literature (Rouphail et al., 1998). Fixed delay-based traffic signal control methods featuring multiple agents have also appeared relatively early in the history of traffic engineering. `TRANSYT` (Robertson, 1969) is a fixed and coordinated traffic signal control method whose objective is to minimize the sum of average vehicle queues by computing optimal per-intersection splits and offsets on a given road network. `TRANSYT` estimates the average vehicle flow value on each link of the network, also called cyclic flow profiles, by using historical data. Based on cyclic flow profile data, pre-specified staging, minimum green times, and cycle time value, `TRANSYT` then simulates traffic flows using different signal timing parameters, each associated with a performance

index. The settings with the best performance index are selected and applied for each intersection. TRANSYT has since evolved into a commercial modelization tool containing a traffic simulator and a signal cycle optimizer.

2.2.1.2 Fixed Bandwidth-Based Methods

Perhaps more surprisingly, multiple bandwidth-based (and hence coordinated) fixed methods also appeared relatively early in the history of traffic signal control, due to the works of Little[3] (1966), which transcribed the bandwidth problem as a mixed-integer linear program. This program is computed using bounds on its cycle time and red phases for a given signal cycle and information about speed along the arterial. Solving this program allows finding cycle times, speed limits, and phase organizations, maximizing the bandwidth along an arterial. The MAXBAND algorithm (Little et al., 1981) uses Little's bandwidth problem formulation to compute the optimal signal parameters to maximize the bandwidth along an arterial. MAXBAND also can generate splits using traffic volume and capacity data. A later extension, MAXBAND–86, also considers left-turn phase sequences in the linear program (Chang et al., 1988). One of MAXBAND's limitations is that its model supposes that traffic flows are uniform along an arterial, meaning that platoons of vehicles are supposed to travel at the same speed and spread on the arterial. MULTIBAND (Gartner et al., 1991) alleviates this weakness by allowing different bandwidth values for each link of the arterial. This modification yields better performance at the cost of a larger solution space. An extension of this method, MULTIBAND–96 (Stamatiadis and Gartner, 1996), adds the possibility to optimize bandwidth along multiple arterials simultaneously.

2.2.2 Actuated Methods

Vehicle actuation methods use vehicle detection systems such as pressure plates or sensors to change traffic signals in real-time. Since these methods rely on vehicle detection, they do not belong in the fixed method category; however, the RL-TSC literature usually considers them distinct to adaptive methods since they have two different modes of operation. On the one hand, actuated methods allocate a minimal green time for each phase of the signal cycle and increase them if vehicles using these phases are detected. On the other hand, adaptive methods estimate in advance the arrivals of vehicles on all phases of the signal cycle and pre-computes its signal cycle accordingly (Shenoda, Machemehl, et al., 2006). Actuated methods are hence somewhat less advanced than adaptive ones.

2.2.2.1 Classical Actuated Methods

Miller first defined a vehicle actuation TSC method on an isolated intersection (Miller, 1963). This intersection is given minimum green time duration and means for vehicle detection. The intersection then scans for vehicles through all of its approaches. When a vehicle is detected, the corresponding approach benefits from a green time extension, as long as it complies with minimum green times defined for other approaches. If no vehicle is present, the method proceeds to the next approach. This method has been refined multiple times by improving its decision process

[3] Little is also known for his work on queuing theory, and especially for Little's law (Little, 1961).

by, for instance, computing the relative gains and losses caused by switching the signal at each period. Out of the extensions of Miller's method, MOVA (Vincent and Peirce, 1988) is probably the most popular. Once a phase reaches minimum green time, MOVA checks whether the links of the active phase are still saturated by computing their output flow rates. If at least one link is still saturated, the current phase is extended until it either becomes under-saturated or reaches the maximum green time. If more than one approach is saturated, MOVA switches to a saturated mode where it estimates queue emptying rates for all approaches at the end of minimum green time and tries to maximize queue capacity along its lanes. The method was tested by its authors and boosted average performance by 13% compared to other vehicle-actuated methods in use at the time (Vincent and Peirce, 1988).

2.2.3 Adaptive Methods

Finally, adaptive methods regroup traffic signal control methods which can adapt their signal timing plans in real-time through the use of sensing technologies allowing the method to monitor the state of traffic in real-time. While these methods are among the most advanced TSC methods presented so far, they are also the least implemented in real traffic scenarios because of their increased cost and complexity.

2.2.3.1 Classical Adaptive Methods

Among the numerous adaptive traffic signal control methods present in the literature, some are of particular interest. First, the SCOOT (Hunt et al., 1981) method is the traffic-responsive version of TRANSYT. Instead of relying on historical traffic data, SCOOT continuously updates its cyclic flow profile estimations using sensors deployed on multiple links of the network. This adaptive capability ensures improved performance: testing in the city of Glasgow showed that SCOOT outperformed TRANSYT by an average of 12% (Robertson, 1986). The SCOOT algorithm has since evolved to become a commercial TSC solution quite widespread in Great Britain and Australia. A second adaptive method, OPAC (Gartner, 1983), leverages dynamic programming methods instead of standard parametric models to leverage real-time arrival data around an isolated intersection. OPAC has since then been extended several times to include more functionality, such as arterial traffic optimization (Gartner et al., 2001).

2.2.3.2 Hierarchical Methods

A more advanced sub-class of adaptive traffic signal control methods regroups *hierarchical methods*. These methods aim to formulate an exhaustive traffic model on a road network and split it into smaller problems distributed across multiple *layers* communicating with each other. Some of these methods have been routing traffic for decades in major urban areas, such as CLAIRE–SITI (Scemama and Carles, 2004) in Brussels, Toulouse, and New Dehli or GERTRUDE in Bordeaux, Lisbon or Beijing (Lambert, 2017).

Since they are both adaptive and coordinated, hierarchical methods rank among the most ambitious ones and often use dynamic programming to solve sub-problems on different scales. A first example of hierarchical TSC systems is PRODYN (Henry et al., 1984). The central controller of

PRODYN first defines a complete traffic optimization program using multiple state equations. Real-time traffic data gathered through sensors is then fed to the program, which splits it using *decomposition coordination*, each sub-problem only depending on local intersection variables. These sub-problems are then solved using intersection-based data and recursive programming techniques and sent back to the central controller, who deduces signal settings for each intersection of the network. In comparison testing, PRODYN has unsurprisingly been found to perform better than TRANSYT (Henry et al., 1984). A second hierarchical TSC method of interest is RHODES (Mirchandani and Head, 2001), which uniquely features three distinct levels of operation. The *dynamic network loading* module in PRODYN captures the slow-changing variables of the network, such as its geometry or the preferred routes of vehicles. Using this data, this module estimates the load in vehicles per hour for each link of the network (as well as other variables such as queue discharge rates and destination probabilities) and sends these estimations to a *Network Flow Control* layer. This second layer allocates a per-intersection green time for each of these estimated traffic streams on a per-platoon basis and passes them to the *intersection control* layer, which explicitly computes the best possible phase and splits settings using forward recursion and dynamic programming.

This chapter gave a general presentation of the field of traffic signal control through two axes.

First, we presented the overall characteristics of traffic signal control. We explained how the need for TSC emerged in the early XXth century and how it focused on two key missions: ensuring the safety of drivers and optimizing traffic flows. We then defined key concepts in traffic engineering, such as the organization of an intersection and the role of signal cycles. We finally established a typology of traffic signal control methods depending on their mode and scale of operation. We explained how fixed, adaptive, isolated, and coordinated methods all had advantages and drawbacks and could be used in distinct traffic situations. Given its potential efficiency and flexibility, we established that developing an adaptive and coordinated TSC methods was the main aim of our work.

Second, we presented multiple classes of real-world traffic signal control methods and briefly explained how they operated. This presentation covered various types of TSC methods, ranging from simple fixed methods to actuated and adaptive methods, and finished with complex hierarchical systems.

3 Reinforcement Learning Theory

Reinforcement Learning describes a class of task-solving machine learning algorithms. At their core, RL algorithms are "a way of programming agents by reward and punishment without needing to specify how the task is to be achieved." (Kaelbling et al., 1996). Reinforcement learning methods are particularly suited for tasks in which an agent must accomplish a task in an environment (e.g., autonomous driving, playing video games) without prior information about this environment. This chapter first gives a general overview of how reinforcement learning methods model the interactions between an agent and its environment and how reinforcement learning algorithms can learn to solve a task from these interactions. Then, it presents how RL models can be extended to allow for multiple agents learning within the same environment and how RL algorithm can use function approximation techniques to increase their learning efficiency.

3.1 Fundamental Reinforcement Learning Concepts

Reinforcement learning models the interaction of an *agent* and an *environment*. The agent aims to maximize its objective by acting on its environment but is not told how different actions will affect its goal depending on the current environment state. Hence, the agent must test multiple actions in a trial-and-error fashion to learn which ones are best suited to maximize its objective. Each reinforcement learning model is divided into three parts. The interactions between the agent and its environment are modeled by a *decision process*; the agent learns from repeated interactions with the environment using a *learning algorithm/and chooses the actions to apply to the environment using a /policy*. We present all three components of reinforcement learning models and their associated challenges in this section.

3.1.1 Markov Decision Process

A Markov decision process (MDP) is a stochastic control process that can model the decisions of an agent aiming at maximizing a global objective function $\mathcal{F}$ in a given environment (Sutton and Barto, 2018). A MDP is defined as a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ and $\mathcal{A}$ are respectively

the *state* and *action* spaces of the MDP, while $\mathcal{R}$ and $\mathcal{T}$ are its *reward* and *transition* functions respectively (Bellman, 1957).

3.1.1.1 Markov Decision Process Loop

The interactions between the agent and the environment in a MDP are modeled as follows. At each discrete time step, the agent observes the state of the environment $s \in \mathcal{S}$ and chooses an action $a \in \mathcal{A}$ to solve its task. Once action a is applied, the environment transitions to a new state s' according to the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ which maps the probability of the system transitioning from state s to state s' when the agent selects action a. Finally, the agent receives a *reward* r computed according to the reward function $\mathcal{R}$, which evaluates the quality of the agent's action according to the task it is trying to solve. In order to choose actions maximizing its successive rewards, the agent uses a *policy* π which a mapping from each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ to the probability $\pi(s, a)$ of taking action a when in state s. As we will see section 3.1.2, various RL algorithms iteratively refine the agent's policy to approximate the optimal policy π^* which yields maximal rewards, perfectly dictating its action choices depending on the current environment state. Figure 3.1 summarizes the interactions between the agent and its environment.

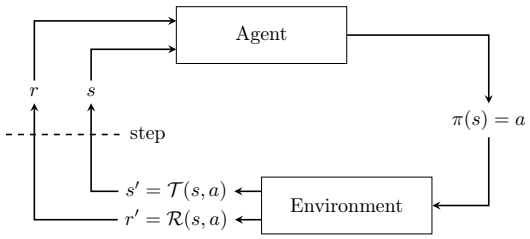

Figure 3.1: Schematic view of agent-environment interactions in a MDP.

3.1.1.2 Reward Signals and Objective Function

The fundamental driving force behind reinforcement learning is the maximization of the agent's expected cumulative rewards. Indeed, the reward scalar r the agent receives at each time step is the only signal indicating the quality of the current agent's actions. MDPs hence imply a strong *reward hypothesis*, stating that any task can be expressed as a reward maximization goal (Sutton and Barto, 2018). This property is one of the most distinctive features of RL models since a reward signal is in theory sufficient for learning to occur.

Since the reward function, $\mathcal{R}$, is paramount in driving learning in RL models, it needs to be strongly correlated with the global objective function of the agent, $\mathcal{F}$. In other words, an agent cannot hope to maximize a metric given by its objective function $\mathcal{F}$ if the signal reward it receives, dictated by $\mathcal{R}$, is not correlated to this objective.

3.1.1.3 State Representation

While they are usually not explicitly defined, multiple state definitions coexist in MDPs. On the one hand, the environment has a *true* state which entirely characterizes it. On the other hand, the agent uses a representation of this true environment state, denoted $s \in \mathcal{S}$.

These two environment states differ since the true state of an environment might contain information that is irrelevant to the maximization objective of the agent. Indeed, the state space of a MDP must contain a sufficient amount of features from the true environment state so that the agent can clearly differentiate environment states. However, including too many features from the true environment state increases the *dimensionality* of the state space and distinguishes environment states that are similar in the context of the task at hand, which is likely to cause a slower learning process (Abdulhai et al., 2003). For this reason, RL models aim at developing an action space containing enough data for the agent to reach an acceptable policy while keeping dimensionality under control. Since we never directly refer to the true environment state in RL models, the environment state s used throughout this thesis refers to the state representation of the environment by the agent.

Markov decision processes assume *full observability* of the environment by the agent. This property ensures that the agent can observe the true state of the environment in order to form its own state representation. Alternative formulations of MDPs, such as partially observable Markov decision processes do not allow for full observability of the true environment state, which forces the agent to estimate this state indirectly. Partial observability models are briefly presented in section 3.2.1.1. Furthermore, some RL algorithms such as linear automata (see section 8.2.2) do not use state representation at all to learn and only rely on the reward signal from the environment.

3.1.2 Learning Algorithm

As stated in section 3.1.1.2, the role of any RL algorithms is to maximize the cumulated reward signals received by the agent during its interactions with the environment. Their primary strategy is to successively try all available actions $a \in \mathcal{A}$ on the environment to identify high-payoff ones. This learning process is, however, not straightforward for two reasons. First, the same action can yield vastly different rewards given the current state of the environment (e.g., steering left in an autonomous vehicle may result in taking a highway exit or crashing depending on its position), which forces the agent to estimate the quality of an action *relatively to the state the environment is in*. Second, the agent cannot measure the quality of some actions immediately after applying them (e.g., investing in stocks may cause a short-term loss but a long-term profit), forcing it to take *delayed rewards* into account. Reinforcement learning methods leverage two distinct components working hand-in-hand to learn how to maximize the agent's utility under these constraints. The learning algorithm estimates the quality of each state of the environment while the agent policy decides which action the agent should take next based on these quality estimates. This section presents the former.

3.1.2.1 Dynamic Programming Methods

Dynamic programming (DP) methods are the only class of learning algorithms providing exact solutions for solving MDPs, short of exhaustively searching the policy space. This optimality is achieved by supposing a complete knowledge of the underlying MDP, and in particular of the reward and transition functions $\mathcal{R}$ and $\mathcal{P}$, which is an assumption that rarely holds in practice (Barto, 1995). The key idea of DP methods is to compute a *value function* V, which estimates the expected value (in terms of expected reward) of each encountered state of the state space $\mathcal{S}$ (Bellman, 1957). For a given state s, the associated value estimate $V(s)$ is computed by estimating,

for each next possible state s', the quality of going into such a new state according to the current policy π. The quality of a state is itself measured as the reward associated with this transition and the value estimate of the new state discounted by a factor γ, weighted by the probability of such a transition occurring according to the current agent policy π (Sutton and Barto, 2018).

$$V_\pi(s) = \sum_{s' \in \mathcal{S}} \mathcal{T}_\pi(s, s')(\mathcal{R}_\pi(s, s') + \gamma V_\pi(s')) \tag{3.1}$$

DP algorithms iteratively refine their policy by using this recursive formulation of the value function. For a policy π, the value function V_π is computed in a process called *policy evaluation*. Once the value function V_π is computed, it can, in turn, be used to improve the existing policy to a superior one, π', in a process called *policy improvement*. By alternating policy evaluation and policy improvement steps in a process known as *general policy iteration*, DP methods converge to an optimal policy π^*, which yields a maximum utility over an infinite horizon (Sutton and Barto, 2018).

3.1.2.2 Monte-Carlo Methods

As opposed to dynamic programming, Monte-Carlo (MC) methods do not assume perfect knowledge of the environment. Instead, MC methods aim at approximating value estimate V by averaging each of its observed return values at the end of a learning episode. Theoretically, the value functions estimated with MC methods converge to the exact value function when the number of visits to each state of $\mathcal{S}$ goes to infinity (Sutton and Barto, 2018).

Since an agent using MC methods has no information regarding the environment, it cannot directly use policy improvement as in the DP case since it requires computing rewards using the reward function $\mathcal{R}$. MC methods can however estimate the value of actions relative to states by using a *quality function* Q. Similarly to the value function $V(s)$, the quality function $Q(s, a)$ computes the expected returns associated with a state s *when choosing* action a. MC methods keep track of each state-action couple (s, a) encounter, $N(s, a)$ within an episode, as well as the associated total cumulated rewards. Using these values, it can approximate the quality of each state-action pair by averaging the total episode gains:

$$Q(s, a) \leftarrow Q(s, a) + \frac{1}{N(s, a)}(\sum r - Q(s, a)) \tag{3.2}$$

MC algorithms are approximation methods, which means that its quality function estimates $Q(s, a)$ improve the more the state-action couple (s, a) is visited. Hence, the longer an agent explores a given MDP and the more distinct state-action couple it encounters, the better the resulting quality estimates will be. Furthermore, since its quality function estimates are computed using the cumulated gains over the entire episode, MC methods can only be applied in environments having a terminal state.

3.1.2.3 Temporal-Difference Learning

Temporal-difference (TD) learning is inspired by both DP and MC methods. Similarly to MC methods, TD methods approximate value or quality estimates since they have no prior knowledge

of the environment. Similarly to DP methods, these value estimates include the estimated value of the next system state (see Equation 3.1.2.1), meaning that they can bootstrap. A simple illustration of TD learning is the TD(0) formula, which iteratively updates the value estimate $V(s)$ of a state s by using the reward value r obtained when transitioning to state s, as well as the value estimate $V(s')$ of the successor state s', weighted by a parameter α known as the *learning rate*.

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{3.3}$$

TD methods present two significant advantages compared to DP and MC methods. First, they can operate incrementally by estimating value functions from other value estimates in an *online* manner (i.e., while being in an episode), as opposed to MC methods. Second, TD methods do not require any model of the environment since the transition and reward functions are not needed for the computation of value estimates, which makes them much more flexible than DP methods.

3.1.2.4 Policy-based Methods

The three types of RL algorithms we have presented so far are *value-based* since they all aim at estimating value or quality functions to approximate an optimal agent policy. Instead of computing value estimates to deduce an optimal policy, *policy iteration* (Howard, 1960) methods aim at directly searching for the agent's optimal policy π^* without relying on value estimates (Arulkumaran et al., 2017). To this end, a parameterized policy is updated to maximize the agent's utility, usually through gradient-based optimization. The REINFORCE algorithm (Williams, 1992) or learning automata (Kaelbling et al., 1996) are instances of such RL policy search methods. Finally, *actor-critic* methods aim to balance value-based and policy iteration methods by using both mechanisms: the critic (value function estimator) gives feedback to the actor (the policy) after each interaction with the environment, both influencing each other in the process. These methods are, in a way, a special case of policy gradient methods (Arulkumaran et al., 2017) and are described in more detail in section 8.2.2.

3.1.2.5 Model-Free and Model-Based Methods

As stated earlier in this section, RL methods such as MC and TD do not need to know the transition function $\mathcal{T}$ in order to properly function. More generally, a method is known as *model-free* when it does not model the transition function $\mathcal{T}$ of the environment. In other words, model-free methods observe successive states of the environment and do not aim to estimate how a chosen action a might influence the transition of the environment to the next state s'. The main advantage of model-free methods is their relative simplicity since no mechanisms exist to estimate environment state transitions and their broad applicability to a large number of RL problems.

Conversely, methods that take into account the transition function are known as *model-based*. It is important to note that while DP methods are necessarily model-based since they cannot function without knowing the transition function $\mathcal{T}$, TD and MC can also be model-based. Indeed, these methods can approximate the transition function $\mathcal{T}$ through successive observations of the environment states. In practice, model-based methods can compute transition estimates through the use of state counters (Wiering, 2000), sometimes coupled with dynamic programming (Bakker et al., 2010; Kuyer et al., 2008) or Bayesian methods (Khamis et al., 2012a,b). Model-based methods

allow for richer models of the environment, which makes them both faster and more sample efficient, ensuring good policy performance in a relatively short amount of time (Yau et al., 2017; Ye et al., 2019). However, this performance usually comes at the cost of model complexity.

3.1.3 Agent Policy

The previous section describes how the learning algorithm aims to estimate the intrinsic value of states and actions of the MDP, which is a *prediction problem*. The last and crucial component of reinforcement learning models is the agent's policy scheme which leverages these value estimates to establish an optimal policy in order to select actions maximizing its rewards. This second mechanism is known as a *control problem*.

3.1.3.1 The Role of Policies

A policy has two often contradicting roles. On the one hand, RL methods using quality estimates have no guarantee to visit all state-action pairs in $(\mathcal{S}, \mathcal{A})$, which may cause the policy to get stuck on a local optimum. The role of policies is hence to promote *exploration* of the state and action space by visiting each pair of $(\mathcal{S}, \mathcal{A})$ infinitely often, which is usually a necessary condition for reaching an optimal policy. On the other hand, the agent uses the policy to maximize its utility, which is obtained by selecting actions with high-value estimates, a technique known as *exploitation*. The drive to explore the state-action space by selecting sub-optimal action for exploration contradicts the drive for maximizing the agent's utility, a phenomenon known as the *exploration-exploitation dilemma*.

3.1.3.2 Greedy and Softmax Policies

A basic approach to aiming at maximizing agent utility would be to pick, for each new system state s, the action $a*$ such that the associated value estimate $Q(s, a^*)$ is maximal across all actions of $\mathcal{A}$. This policy, known as *greedy*, does not guarantee sufficient exploration and is likely not to reach optimality. A commonly used policy alleviating this issue is the ε-greedy policy, which selects the action associated with the highest value estimate with probability $1 - \varepsilon$ or a random action otherwise, ensuring that the agent can visit all state-action pairs for $\varepsilon > 0$ (Sutton and Barto, 2018). A standard limitation of ε-greedy policies is that all actions are chosen with the same probability if a random action is to be selected, which may be undesirable if some actions are associated with low-value estimates. The softmax function policy counteracts subpar action selection by assigning a distinct probability weight to all actions of the action space based on their estimated values and on a temperature parameter that determines the randomness of action selection. Hence, the softmax policy favors high-payoff actions even when randomly selecting actions while maintaining sufficient exploration by assigning a non-nil probability weight to all actions (Sutton and Barto, 2018).

3.1.3.3 Illustration of Policy Impact

This section illustrates the influence of exploration and exploitation of the state and action space through agent policy through a simple learning problem. The Cart-Pole problem features an agent

whose objective is to balance a pole placed on a cart. The agent can move the cart left or right and receive a reward equal to the number of steps the stick stays on the cart without falling. We compare the performance of three learning agents on the Cart-Pole problem using the Q-learning algorithm (Watkins and Dayan, 1992), and an ε-greedy policy with different ε values: a constant value of 0.05, a constant value of 0.5, and a decreasing value of $\varepsilon = 1 - \log_{10}(n + 1/25)$, with n being the current learning episode (bounded between 0.1 and 1).

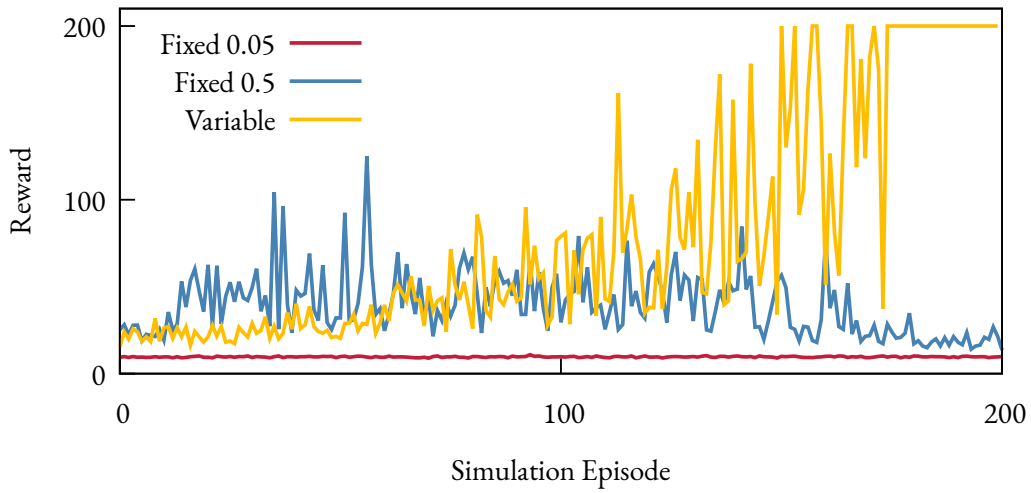

Figure 3.2: Learning process evolution of the Q-learning algorithm solving the CartPole problem using an ε-greedy policy with different ε rates. The maximum reward per episode is 200.

Results of these simulations, as shown on Figure 3.2 underline the importance of proper balance between exploration and exploitation. In the $\varepsilon = 0.05$ case, the exploration parameter is too low for sufficient exploration. The agent gets stuck in a local optimum by making the pole fall early, yielding a small but positive reward. Conversely, using a high exploration parameter $\varepsilon = 0.5$ causes quick exploration of the state space, which explains superior performance in the early episodes. However, this high random action selection rate proves unable to exploit high-payoff actions due to the high policy unpredictability. Hence, using a decaying exploration rate that favors exploration in early episodes and exploitation later allows the algorithm to converge to an optimal policy.

3.2 Reinforcement Learning Model Extensions

The concepts presented so far give us enough tools to build simple reinforcement learning methods, but such methods would suffer from substantial shortcomings. First, the presented RL model only features a single agent and would hence be unable to function with multiple agents learning in parallel over a road network as commonly seen in historical TSC methods (see section 2.1.3.4). A less obvious issue comes from the fact that all the reinforcement learning algorithms presented so far rely on an exhaustive exploration of the state and action spaces of the environment. This search can prove extremely inefficient when these spaces get sufficiently large and

pose acceptability problems when applied to traffic signal control tasks. The two reinforcement learning model extensions presented below deal with each of these issues to improve the overall capabilities of RL methods, which will be applied to traffic signal control later on.

3.2.1 Multi-agent Reinforcement Learning

Featuring multiple agents learning in parallel is likely to be desirable when modeling multi-agent systems–such as traffic signal control–which are commonly used in the field of reinforcement learning (Arulkumaran et al., 2017; Busoniu et al., 2008). Multi-agent reinforcement learning (MARL) models present clear advantages such as increased performance thanks to decentralized execution, improved robustness, or permitting experience sharing between agents (i.e., different learning agents exchanging value estimates they have learned separately) (Busoniu et al., 2008). While it would seem natural to use multiple learning agents without changing anything else, moving from a SARL to a MARL model modifies the theoretical framework in which these agents learn, which creates several new challenges which need to be addressed.

3.2.1.1 Partial Observability

The first effect caused by the introduction of multiple learning agents relates to choosing how much of the environment they can observe and act upon. Since it is common to feature MARL models in which each agent only acts locally, the decision process associated with such models usually changes to a partially observable Markov decision process (POMDP), which extend the model of MDPs by adding constraints on the ability of each agent of the system to observe the entire state of the environment (Panait and Luke, 2005). POMDPs are represented as a 6-uple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, O)$. Besides the usual MDP elements, POMDPs feature an observation space Ω, containing the set of states of the environment that are observable by each agent and an observation function, O, containing the probabilities of encountering a given observation from the observation space O depending the previous agent action a and the new true environment state s' (Oliehoek et al., 2008; Sutton and Barto, 2018). As opposed to traditional MDPs, an agent in a POMDP setting must maximize its utility under uncertainty as it can only receive partial observations from the observation space Ω instead of true system states from the state space $\mathcal{S}$. The agent hence learns to associate observations to system states by estimating the observation function O through the use of *belief states* which model observation probabilities through Bayesian estimations of the entire process' history (Bakker et al., 2010).

3.2.1.2 Agent Interactions

Another significant impact caused by the introduction of multiple learning agents in the same environment is that these agents can interact and influence each other. RL models can choose to explicitly model agent interactions through *coordination* mechanisms in which agents take each other into account or even communicate (see section 4.2.3 for an illustration of agent coordination applied to traffic signal control). Alternatively, MARL models can choose to ignore these interactions, hence implementing *independent* learning in which agents ignore each other and maximize their own local rewards. Regardless of modeling choices, the fact that agents influence each other in MARL models cannot be ignored. Since multiple agents act concurrently on the

environment, the actions of one agent can influence the environment state of another. This phenomenon, known as *non-stationarity*, can lead agents to believe that their actions caused changes in the environment that were in reality caused by others. The absence of a stationary environment can potentially cause RL algorithms to never converge to an optimal policy due to a moving-target issue (El-Tantawy and Abdulhai, 2010).

Furthermore, the correlation requirement between reward and objective function must be even more carefully designed in MARL systems. Indeed, since each agent greedily aims to maximize its locally observed reward function, one must ensure that these local optimization goals are not clashing with each other and are properly correlated with the global objective function (Busoniu et al., 2008).

3.2.2 Function Approximation

A widespread issue associated with RL methods is caused by the size of their environment's state and action spaces, which is also known as their *dimensionality*. Since RL algorithms have to perform an exhaustive search of these spaces to establish value estimates, the computation and memory storage costs associated with this search grow exponentially as they increase. A second, and perhaps worse, issue related to large state spaces is well summarized by Sutton and Barto (2018):

> The problem with large state spaces is not just the memory needed for large tables, but the time and data needed to fill them accurately. In many of our target tasks, almost every state encountered will never have been seen before. To make sensible decisions in such states it is necessary to generalize from previous encounters with different states that are in some sense similar to the current one. In other words, the key issue is that of generalization. How can experience with a limited subset of the state space be usefully generalized to produce a good approximation over a much larger subset?

Techniques of *function approximation* provide an elegant answer to both of the issues mentioned above. First, by not storing value estimates in a tabular fashion (i.e., each state is associated with its own value), these techniques can deal with much larger state spaces without dimensionality issues. Second, function approximation allows RL algorithms not only to learn the value estimates of states they visit but *generalize* these results to predict the value of states they have not yet encountered.

3.2.2.1 Function Approximation

Function approximation aims to extract information from state features and their associated values to approximate the entire value function of the problem. In other words, function approximation does not associate a value estimate $V(s)$ to each encountered state s separately but uses state and reward values to directly estimate how each feature of the state space impacts its associated value estimate. This task is achieved by approximating the value function using a parameterized function, which is by nature a supervised learning task (Sutton and Barto, 2018). Approximation function can take simple forms such as a linear function of features of the observed state or more complex structures such as multi-layered neural networks.

3.2.2.2 Deep Reinforcement Learning

An increasingly popular way of using function approximation in RL is to use neural networks as function approximators (Arulkumaran et al., 2017), leading to a specific branch of RL known as deep reinforcement learning (DRL) (Gregurić et al., 2020). While presenting how neural networks operate in detail is outside of the scope of this thesis[1], neural networks are machine learning models featuring multiple layers of neurons and activations weighted by parameters θ. A neural network maps a multi-dimensional input vector to a mono-dimensional one. Training a neural network involves computing a *loss function* $\mathcal{L}$ measuring the difference between the neural network's output and the observed value. Using gradient descent methods, the neural network is trained to properly estimate the correct output vector for a given input one. In the case of DRL, the output of such a neural network is the value estimate $V(s, \theta)$ of a state s given as input. The output value is compared to the true reward value obtained from the environment, and the weight parameters θ are then corrected accordingly using gradient descent.

3.2.2.3 Convergence Issues

DRL models provide a number of significant advantages, such as state generalization under much larger dimensionality than classical RL methods. However, DRL methods also break the convergence guarantee of classical RL algorithms by moving away from tabular representations (Van der Pol, 2016). The first reason for these convergence issues is that the observation data in RL models is assumed to be independently and identically distributed (i.i.d.). This is, however, not the case for DRL models since evolving policy and function approximation make these observations both correlated and unevenly distributed (McCloskey and Cohen, 1989). A second convergence issue of DRL methods is due to a *moving target* phenomenon. Since each observation updates the entire weights θ of the function used to approximate value estimates, these updates may affect earlier estimations and cause the learning target to oscillate constantly. We present common solutions to these convergence issues in section 8.3.2.

This chapter introduced reinforcement learning algorithms, which allow an agent to maximize a task in an environment by maximizing its cumulated expected rewards. We notably described how a MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ could model the interactions between the agent and the environment. The agent observes the current environment state s, chooses an action a, and receives an associated reward r while the system transitions to a new state s'.

Agents learn to maximize their cumulated expected rewards through the combined use of a learning algorithm and of a policy. The learning algorithm estimates the relative value of state and of state-action couples through the use of a value V and quality Q function, respectively. On the basis of these value and quality estimates, the policy dictates which action the agent should select based on the current system state in order to maximize its rewards.

Since classical single-agent RL models are rather limited, we finally introduced two RL extensions. First, we briefly described how multiple agents could solve tasks concurrently in the same

[1] Anthony and Bartlett (2009) provide a general introduction to neural networks, and Van der Pol (2016) a more succinct presentation of neural networks aimed at TSC applications

environment, even enabling agent coordination. We then finally introduced the concept of function approximation and deep reinforcement learning, which allows the agent to deal with much larger state spaces and to generalize past state encounters through a parameterized function.

4 Reinforcement Learning Applied to Traffic Signal Control

The typology of historical traffic signal control methods established in chapter 2 allows to distinguish TSC methods according to their features and complexity. It is common to sort these methods according to distinct generations: actuated methods form the first generation of TSC methods; centralized adaptive methods such as SCOOT or OPAC the second generation; and advanced hierarchical methods such as PRODYN or RHODES constitute the third generation (Gartner et al., 1995). The application of RL to traffic signal control gave birth to a fourth AI-based generation of methods (El-Tantawy and Abdulhai, 2012). This section gives an overall tour of the state of reinforcement learning applied to traffic signal control research since its inception in 1994 (Mikami and Kakazu). Notably, we present how TSC problems can be adapted to a RL-based framework and showcase the main contributions and advances in the field of RL-TSC for both isolated and coordinated methods.

4.1 Modeling Traffic for Reinforcement Learning

If multiple RL-TSC methods can tackle the traffic optimization problem quite differently, they still all have to define this problem within a RL-centric framework. First, some general model choices have to be made regarding traffic, such as defining what it means to optimize traffic, which components of the traffic network are considered agents, and how well the agent can observe its environment. Then, one must choose the specific elements constituting the MDP, such as which type of information about the road network the agent observes, how each agent can act upon the road network, and how its rewards are measured.

4.1.1 General Model Choices

This section refers to the broad model characteristics of a RL-TSC method. These choices are a crucial first building block of any RL-TSC method since they define which kind of agents are

learning to optimize traffic, how much information they gather from their environment, and how much they interact with each other.

4.1.1.1 Types of Learning Agents

An overwhelmingly common approach in the RL-TSC literature is to consider that each intersection is a learning agent. While this choice is evident for isolated TSC models (Abdulhai et al., 2003), it is also the preferred choice for MARL models featuring multiple intersections (Mannion et al., 2016). If multi-intersection road network optimization could theoretically be tackled by a single central planner controlling multiple traffic lights simultaneously, such a model would have important limitations. Indeed, such a model would suffer from poor scalability (since adding or removing an intersection to the network completely changes the learning model) and an exponential increase in dimensionality, explaining why such an approach is common among third-generation TSC methods but absent from RL-based ones (Yau et al., 2017). In the MARL case, each intersection has a local view of its environment (usually neighboring lanes) according to the POMDP model (see section 3.2.1). Note that other approaches are nevertheless possible on multi-intersection networks, such as considering vehicles as additional agents that can collaborate with intersections (Bakker et al., 2010; Kuyer et al., 2008; Steingrover et al., 2005; Wiering, 2000) or using intersection clusters as agent (Bazzan et al., 2010).

4.1.1.2 Model-Free and Model-Based Methods

A second major design decision regarding RL-TSC methods relates to their estimation of the transition function $\mathcal{T}$. Indeed, both model-based and model-free methods, described in section 3.1.2.5, have been applied in the RL-TSC literature (Mannion et al., 2016). As stated in this section, model-free methods are usually simpler and slower since they do not estimate state transitions, while model-based methods are more complex and efficient.

In the realm of RL-TSC, model-based approaches are believed by some to introduce unnecessary complexity (El-Tantawy et al., 2013; Mannion et al., 2016) as they can prove more problematic for learning problems with large state sets (Crites and Barto, 1995). The most telling example of this preference is shown in the literature review analysis of Noaeen et al. (2021) which states that only 8 out of the 160 surveyed RL-TSC papers used model-based RL methods, the last of which was published in 2014 (Khamis and Gomaa). Regarding multi-agent modeling choices, the use of POMDPs is prevalent in the literature since model-based (Bakker et al., 2010) and some actor-critic methods employ it (Richter et al., 2007), even though a few methods model interactions between intersections as stochastic games (Bazzan, 2009; Bazzan et al., 2010) or Markov games (Aragon-Gómez and Clempner, 2020).

4.1.1.3 Independent Learning and Coordination

A final major design decision regarding multi-intersection TSC methods is the modeling of interactions occurring between each learning agent. The simplest way to model these interactions is to ignore them and consider independent MARL models in which no agent-to-agent interactions exist (Tan, 1993). Even though we have shown in section 3.2.1 that agents necessarily influence each other in MARL models, some papers of the RL-TSC literature do not consider it to be a limiting

factor of agent performance in the case of TSC (Pham et al., 2013). While this statement might not be accurate, a significant number of independent MARL methods show excellent performance without ever addressing this issue (Noaeen et al., 2021). Explicit coordination mechanisms can, however, limit non-stationarity and increase overall traffic routing performances in RL-TSC contexts (Mannion et al., 2016). Such coordinated MARL models are applicable in both model-based and model-free situations and present numerous advantages besides alleviating issues relating to non-stationarity. Indeed, junction-to-junction coordination can be used to emulate green wave coordination or complex third-generation TSC methods as presented in section 2.2.3. Various modes of agent coordination are successfully used in the RL-TSC literature, ranging from simply observing a neighboring intersection's state to directly computing optimal joint actions (Yau et al., 2017).

4.1.2 Model Parameter Design

After having decided which type of agent is going to be learning how to optimize traffic and how these agents are going to interact with each other, any RL-TSC model still has several design decisions to take regarding the modeling of the environment. The proper definition of the objective function, state space, action space, and reward function is crucial for the learning process of any reinforcement learning method.

4.1.2.1 Objective Function $\mathcal{F}$

The general objective of optimizing traffic is rather vague when it needs to be explicitly translated into an objective function for the agent to maximize. Existing surveys of the RL-TSC literature give us valuable insight as to which objective functions are commonly used across various traffic optimization methods using reinforcement learning. Note that when constructing a RL-TSC method, one does not need to limit the objective function to a single metric. Indeed, some papers of the literature aim at maximizing multiple traffic-related metrics at once by using compound reward definitions or by using multiple objective functions (Brys et al., 2014; Houli et al., 2010; Khamis and Gomaa, 2014; Khamis et al., 2012a), which is a learning technique known as *multi-objectivity*. The surveys of Wei et al. (2019) and Noaeen et al. (2021) list the following classes of objective functions.

Temporal Metrics Time-based metrics are by far the most common form of objective functions in RL-TSC applications. Vehicular *travel time* is a metric measuring the total travel time of a vehicle across the network. Travel time is easily obtainable but does not differentiate between trip-related and congestion-related time spent in the network. A more accurate temporal metric is *delay*, which is defined as the difference between the observed and expected travel time of a vehicle. Delay measurements allow account for the time loss due to congestion or routing inefficiencies but necessitate an estimation model of the expected travel time of vehicles. Finally, the *waiting time* is defined as the amount of time a vehicle has waited at a red light or due to congestion. This metric has the advantage of only measuring waiting time due to network inefficiencies, making it a suitable objective function for traffic optimization. Furthermore, this metric has the advantage of being measurable before vehicles reach their destination. However, waiting time is readily available in traffic simulations but hard to obtain in real-life scenarios.

Congestion Metrics A second class of traffic metrics suitable for RL-TSC objective functions are congestion-related. Absolute congestion values such as *queue size*, have been used as an objective function in RL-TSC applications. Queue size metrics are well-suited to road networks with low vehicular capacity or known bottleneck areas. Alternatively, congestion metrics over time, such as *intersection throughput*, have been used to measure the efficiency and evenness of TSC methods.

Speed Metrics Finally, multiple forms of *vehicular speed* such as absolute speed, acceleration, or harmonic speed are also used as objective functions. These objective functions prioritize vehicle movement smoothness and uniformity and are sometimes associated with related variables such as the number of stops per trip.

4.1.2.2 Reward Function $\mathcal{R}$

As stated in section 3.1.1.2, the objective and reward functions of any RL model have to be tightly correlated since the agent will use signals from the latter to maximize the former. It is hence logical that the traffic metrics used to determine the objective function in the previous section are present in the reward function definition. Hence, temporal, congestion, and speed metrics are once again the most common components of reward functions in the RL-TSC literature (Yau et al., 2017).

Temporal Metrics In practice, delay measurements are commonly used as a reward function since they estimate the vehicular time loss due to traffic routing inefficiencies (Mannion et al., 2016). These measurements include vehicular delay or cumulated waiting time on a lane, either in absolute, difference, or average form (Arel et al., 2010; El-Tantawy and Abdulhai, 2010). Delay can also be squared to penalize further large delay values (Abdulhai et al., 2003; Brys et al., 2014).

Congestion Metrics Congestion metrics used for reward function definition can include queue size or variation of queue size (Araghi et al., 2013; Mikami and Kakazu, 1994), as well as intersection throughput metrics (Brys et al., 2014; Touhbi et al., 2017). Furthermore, some papers developed a metric known as *green time appropriateness*, for which the agent is penalized when unused green time is observed while vehicles are idle at red lights (Cahill et al., 2010).

Speed and Mixed Metrics A few papers use vehicular speed metrics as part of their reward functions, either as an absolute value or as a ratio between observed speed and maximum allowed speed (Van der Pol and Oliehoek, 2016). Usually, speed metrics are used as *multi-objectivity* metrics, as mentioned in the previous section. For instance, the reward function featured in Van der Pol and Oliehoek's model (2016) is composed of delay and speed measurements, as well as emergency stops and accident indicators.

4.1.2.3 State Space $\mathcal{S}$

Traffic signal control problems are a perfect illustration of the trade-off between complex and simple state representation mentioned in section 3.1.1.3: the state space of a MDP only contains a subset of features from the true environment state that is relevant to the agent. RL-TSC models

hence have a large amount of traffic-related features of the environment at their disposal in order to define the state space $\mathcal{S}$ of the MDP. The most common features used for state definition identified by Yau et al. (2017) are listed below.

Congestion Metrics Congestion metrics, and queue size per lane in particular, are among the most commonly used for state space definition within the RL-TSC literature (Mannion et al., 2016). These metrics can take into account all vehicles of the lane or halted vehicles only. Furthermore, the absolute number of vehicles can be used, as well as queue categories (e.g., low, medium, or high congestion) (Cahill et al., 2010; Chin et al., 2011). A minority of articles use relative queue size (i.e., the ordering of queue sizes of lanes around an intersection) instead of absolute values (Abdoos et al., 2011). Finally, the maximum queue size across all lanes also has been used as a state variable (El-Tantawy and Abdulhai, 2010).

Traffic Signal Metrics Traffic signal metrics such as phase-related indicators are sometimes used in state space definition. Such variables include the current green phase index in the signal cycle and the duration for which it has been active (Arel et al., 2010), or the current red phase timing. This type of information is beneficial for the learner as it indicates the current state of the signal cycle at its intersection.

Speed and Positional Metrics Newer RL-TSC methods, usually using deep neural networks, often use detailed positional data as state representations. Hence, vehicular positions can be represented using cellular encoding, also known as discrete traffic state encoding (DTSE) (Genders and Razavi, 2018), and used as state inputs for neural networks in the form of binary matrices (Van der Pol and Oliehoek, 2016) or even as images fed to a convolutional neural network (Mousavi et al., 2017). Other vehicular data, such as speed, can be used in place of positional indices (Van der Pol and Oliehoek, 2016).

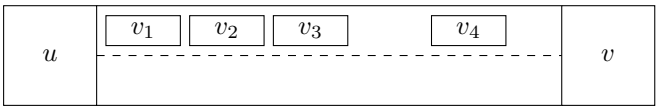

Figure 4.1: For a given lane, a MDP can use multiple environment features to represent the current traffic state. Number of vehicles in a queue (3) or on the lane (4) can be used for vehicle-related data. Current phase index and duration are phase-related indicators. Finally, DTSE representations can be used (`11101` in the current situation).

The RL-TSC literature is not unanimous in its choices regarding state space definition since it is often algorithm-dependent. For instance, the choice of simplifying, or discretizing, state representations to reduce dimensionality is taken in approximately 40% of papers surveyed by Noaeen et al. (2021), meaning that 60% of the surveyed papers opted for fuller state representations such as DTSE. The large amount of different features of the road network that are used in state representations in papers of the RL-TSC literature also underlines this lack of consensus.

4.1.2.4 Action Space $\mathcal{A}$

Since TSC consists in influencing traffic through the use of traffic signals, it is no surprise that action spaces of RL-TSC methods all revolve around traffic phase control. Yau et al. (2017) identify two main types of actions in their survey of the RL-TSC literature.

Traffic Phase Splits An agent using traffic phase splits chooses a time interval allocation for each phase of the signal cycle. This allocation can either be *phase-based*, meaning that the agent allocates the entire phase duration at once, or *step-based*, in which the agent evaluates whether to switch or extend the current phase at regular intervals. As we will see in section 6.4, the influence of using step or phase-based actions on traffic-routing performance has not been studied until recently. Results from a paper we published on the matter have shown that, in the case of traffic phase split actions, step-based actions were strictly superior to phase-based ones (Tréca et al., 2020a).

Traffic Phase Choice An agent using traffic phase choices directly decides which traffic phase is to be activated next. This action type offers more flexibility for the agent but comes at the cost of additional complexity due to necessary safety and compatibility checks on the generated signal cycles.

4.2 Traffic Signal Control Methods

The previous section described how papers of the RL-TSC literature model the RL framework applied for traffic optimization. These model choices covered agent representation, agent coordination, choosing between a model-free or model-based method, and defining each element of the underlying MDP. We now present the different RL-TSC methods themselves, in increasing order of complexity. Isolated TSC are presented first, followed by MARL methods, both in independent and coordinated cases, to finish with function approximation methods applied to traffic optimization.

4.2.1 Single Agent Reinforcement Learning Applied to TSC

Even though they are somewhat rare given the popularity of multi-agent systems for TSC, a few papers of the literature have studied the behavior of an isolated intersection using RL-TSC (Mannion et al., 2016). These papers can broadly be divided into two categories. Some feature a single intersection to study the performance of a specific learning algorithm–which happens to be Q-learning (Watkins and Dayan, 1992) in most cases (Abdulhai et al., 2003; Chin et al., 2011; El-Tantawy and Abdulhai, 2010; Wen et al., 2007)– while others voluntarily restrict their field of study to an isolated intersection to analyze specific learning-related phenomenon such as action space definition or function approximation techniques (Tréca et al., 2020a,b).

4.2.1.1 Classical SARL Methods

Papers from the first category often compare the performance of the Q-learning algorithm to classical TSC methods. For instance, El-Tantawy and Abdulhai (2010) found out that single-agent Q-learning outperformed fixed signal timing for multiple RL state definitions. Abdulhai et al.

(2003) used Q-Learning on an isolated intersection combined with the cerebellar model articulation controller (CMAC) function approximation technique (see section 4.2.4) and found that their method had results on par with the TRANSYT and SCOOT methods under constant vehicle flows but performed significantly better under variable flows. Similarly, Chin et al. (2011) studied the convergence of an isolated intersection using Q-Learning and an ε-greedy policy. The Q-Learning method could adapt to peak-hour traffic situations simulated using real-world data, even though the authors did not compare it to another TSC control plan.

4.2.1.2 Limitations of SARL Methods

If SARL models provide some benefits compared to fixed TSC methods on isolated intersections, they are still seldom used due to two significant shortcomings. First, they cannot be applied to urban areas with multiple intersections without a central controller, which would lead to unmanageable dimensionality as the number of intersections increases (as stated in section 4.1.1.1). Second, these methods do not feature desirable properties such as junction-to-junction communication or cooperative learning, which are essential in traffic management in urban areas. These limitations are also illustrated by the fact that most of the literature reviews regarding RL-TSC methods choose to exclude isolated TSC methods from their review (Noaeen et al., 2021; Yau et al., 2017).

4.2.2 Multi-Agent Reinforcement Learning Applied to TSC

Given the limitations of single-agent models, introducing multiple agents in traffic control systems is a logical and somewhat natural choice. Indeed, leveraging reinforcement learning over multi-intersection networks has been an objective of RL-TSC models since their inception. For instance, the three first papers coupling reinforcement learning and traffic signal control aimed to do so over multiple intersections (Cao et al., 1999; Mikami and Kakazu, 1994; Wiering, 2000).

4.2.2.1 Model-Free MARL Methods

A great number of learning techniques have existed early on for optimizing traffic, such as linear automata (Mikami and Kakazu, 1994), fuzzy logic and classifier systems (Cao et al., 1999) or model-based reinforcement learning (Wiering, 2000). However, Q-learning is the RL algorithm of choice for RL-TSC systems. Q-learning is present in 60% of the RL-TSC papers surveyed by Noaeen et al. between the years 1994 and 2020.

The Q-learning algorithm was first applied in a RL-TSC context by Abdulhai et al. (2003) and is popular for its relative simplicity (since it is model-free) and extensibility. An example of Q-learning applied in a multi-agent setting can be found in a paper by Abdoos et al. (2011) in which a set of 50 intersections implement a Q-learning algorithm in parallel, outperforming the tested fixed signal plans. An innovative extension of the traditional MARL Q-learning model for TSC can be found in `Soilse` (Cahill et al., 2010). On top of regular multi-agent Q-learning, `Soilse` features a pattern change detection (PCD) mechanism allowing the Q-learning algorithm to re-learn depending on the degree of traffic flow change. The more the nature of traffic demand changes, the more the learning rate α of the Q-learning algorithm increases, giving more weight to newer state observations. When traffic demand stabilizes, the learning rate starts decaying again (Cahill et al., 2010).

It should be noted that while such an approach might suffer from non-stationarity issues (see section 3.2.1.2), multiple agent learning concurrently using each a Q-learning algorithm, also defined a *independent Q-learning* (Tan, 1993) provides surprisingly strong performance benchmarks in a number of RL areas (Leibo et al., 2017; Tampuu et al., 2017) including TSC problems (Ye et al., 2019).

4.2.2.2 Model-Based MARL Methods

One of the earliest and most influential RL-TSC models featuring multiple agents, proposed by Wiering (2000), is model-based. Wiering's model features both intersections and vehicles as learning agents, aiming to minimize vehicular waiting time, optionally communicating destinations and waiting time (for vehicles), and congestion information (for intersections). A first extension of Wiering's model includes additional congestion data from neighboring intersections, increasing agent performance and dimensionality in doing so (Steingrover et al., 2005). A second extension refines the computation of estimates of the transition function $\mathcal{T}$ by leveraging maximum likelihood estimations and dynamic programming (Bakker et al., 2010). A final series of extensions by Khamis and Gomaa respectively added complex car acceleration models, Bayesian transition probability estimation, multi-objectivity and agent cooperation to Bakker et al.'s model (Khamis and Gomaa, 2014; Khamis et al., 2012a,b).

4.2.3 Agent Coordination Applied to TSC

Additionally to defining which learning algorithm should be used by intersections in order to optimize traffic, MARL models also have to decide on the interaction model of its agents. Indeed, as we have seen in section 3.2.1.2, MARL models can either choose to ignore agent-to-agent interactions, resulting in independent learning methods, or choose to model these interactions through coordination. These coordination mechanisms range from entirely independent learning to direct coordination and joint-action selection. This section reviews the most commonly used coordination modes of the RL-TSC literature.

4.2.3.1 Max-Plus Algorithms

Multiple Wiering-type models mentioned in the previous section have been extended to include direct coordination between junctions. Such extensions, due to Kuyer et al. (2008) and Bakker et al. (2010), leverage *coordination graphs* and the *max-plus* algorithm for agent coordination. Since it is impossible to coordinate all intersections simultaneously because the state space increases exponentially with the number of agents, coordination graphs decompose the global payoff function into a local function depending on a subset of agents. The global optimum can then be obtained by computing the local optimal joint actions of each sub-problem (Kok and Vlassis, 2005). In order to quickly compute the optimal joint action of each sub-problem, the max-plus algorithm organizes efficient message sharing between local agents for a fixed number of iterations to coordinate their action choice in a limited amount of time.

Even though the max-plus algorithm significantly speeds up the coordination process between agents, coordinated MARL methods remain highly computationally intensive, and their use is generally discouraged in time-critical applications (Bakker et al., 2010). In terms of performance, Wiering-type coordinated methods outperform all of the non-coordinated models of the same

type in highly saturated conditions. When traffic is not saturated, however, performance is on par with coordination-free model-based methods but at the cost of longer computation time (Bakker et al., 2010; Kuyer et al., 2008). Note that some model-free methods also feature coordination graphs coupled with the max-plus algorithm (Medina and Benekohal, 2012; Van der Pol, 2016).

4.2.3.2 MARLIN Algorithms

Another highly popular model-based approach used in RL-TSC coordination is MARLIN (El-Tantawy and Abdulhai, 2012; El-Tantawy et al., 2013). The MARLIN algorithms rely on two key concepts of the MARL literature: the *principle of the locality of interaction* and *modular Q-learning*. The principle of the locality of interaction states that for POMDPs in which agent interactions are limited to their neighborhood, optimizing the local joint utility of an agent and its immediate neighbors is sufficient to reach an optimal agent policy (Nair et al., 2005). Modular Q-learning can reduce the dimensionality of the state and action space of a problem by partitioning it between sets of two agents (Ono and Fukumoto, 1997). Like coordination graphs, modular Q-learning divides a joint problem between N agents of dimensionality $|s|^N$ into $N-1$ sub-problems between two agents, each of dimensionality $|s|^2$, hence keeping dimensionality in check. Once these sub-problems are solved, the agent chooses the action maximizing the sum of these sub-problems.

Two variants of the MARLIN algorithm exist. In the MARLIN-IC (for *indirect coordination*) version, each intersection models interactions with each of its neighbors in a joint Q-table and estimates the impact of its next action choices based on these joint Q-tables (see section 9.3.1 for a detailed explanation of the algorithm). As for MARLIN-DC (for *direct coordination*), agents directly exchange their current policies with their immediate neighbors and negotiate a joint set of actions maximizing their joint utility. Both of these methods have been tested on a simulated network of 59 intersections representing downtown Toronto using real traffic data. Both MARLIN variants outperformed the real-world method implemented on the same network (El-Tantawy et al., 2013). MARLIN has long been considered to be a state-of-the-art coordinated TSC method (Brys et al., 2014; Mannion et al., 2016; Yau et al., 2017). However, since its original publication in 2012, the field of RL-TSC has rapidly adopted more function approximation techniques which greatly improve the performance of MARL methods, coordinated or not (Noaeen et al., 2021).

4.2.3.3 Alternative Algorithms

If most RL-TSC coordination methods rely on well-known coordination mechanisms such as the max-plus or MARLIN algorithms, several original coordination techniques have appeared in recent years. One can find a novel approach to traffic light coordination in the γ-reward model of Liu et al. (2021), which considers spatial delayed reward as a vector for agent coordination. If a vehicle takes n steps to travel from intersection u to intersection v, the reward of agent u will not only take into account the local delay at step t but also part of the delay that it caused around intersection v at time $t+n$. This delayed reward forces agents to take the utility of other intersections into account when maximizing their own (Liu et al., 2021). Another coordination model developed by Chen et al. (2020) aims to optimize traffic on large-scale networks (around 2500 traffic lights) on a region-to-region basis. Traffic is optimized by region by uniforming traffic pressure (defined as the difference between upstream and downstream congestion around an intersection) using deep Q-learning. A final example of alternative TSC coordination methods can be found in the works

of Qi et al. (2020), which optimize traffic similarly to third-generation classical TSC methods by coordinating traffic lights through platooning estimations by supposing that some vehicles of the road network are autonomous and communicate with intersections.

4.2.3.4 Asserting the Usefulness of Coordination

All papers presented in this section have shown that RL-TSC methods featuring agent coordination provided superior performances to independent methods in a number of contexts, especially on large-scale networks. A paper by Wagner et al. (2019) has, however, claimed that traffic light coordination is difficult to achieve in real-world conditions and that few parts of a road network might benefit from it. Multiple simulated scenarios have backed these claims in which well-parameterized independent actuated methods have outperformed coordinated ones. However, the authors have claimed that these results are preliminary and require more investigation, which is the primary goal of chapter 9 of this thesis. This thesis tackles the complex issue of agent coordination and studies its potential benefits in chapter 9.

4.2.4 Function Approximation Techniques

If agent coordination has been largely studied in the field of RL-TSC since its inception, the application of function approximation techniques to enhance traffic routing optimization has been at the forefront of RL-TSC research in recent years. We distinguish two types of function approximation techniques applied to RL-TSC. The first category is formed by classical methods that are often based on simple or older neural networks models. The second category regroups recent methods that apply recent approximation methods from the deep reinforcement learning literature. These two categories of function approximation techniques are presented in this section.

4.2.4.1 Classical Function Approximation

To the best of our knowledge, Abdulhai et al. (Abdulhai et al.) were the first to apply a function approximation technique on a RL-TSC task. The technique they applied is the CMAC model (Albus, 1975), which can be seen as a hybrid data structure in-between an artificial neural network and a sophisticated lookup table (Brys et al., 2014). When a state-action pair is visited, CMAC propagates the Q-value estimates to other pairs based on their similarity, allowing for faster exploration of the state space, hence speeding up the convergence process. Pham et al. (2013) have applied a similar function approximation method known as *tile coding*. Tile coding partitions the state space according to different subsets (or tiles) and maps them to states by similarity.

The QTLC-FA function approximation method, used by Prashanth and Bhatnagar (2011), aims to approximate the Q-function with a matrix formed of multiple d-dimensional vectors (one per state-action pair), d being much lower than the overall dimensionality of the problem. These vectors are coupled with a tunable parameter matrix which is iteratively updated in a Q-learning-like fashion using gradient descent. For instance, in a three-by-three grid network used by the authors, the dimensionality of the problem is reduced from 10^{101} to $d \approx 200$ using QTLC-FA while retaining good performances compared to the non-approximated model.

4.2.4.2 Deep Reinforcement Learning

The use of deep learning has become prevalent in RL-TSC in the last few years, a rise that can be confirmed by many literature review papers specifically studying the use of deep reinforcement learning for RL-TSC (Gregurić et al., 2020; Haydari and Yilmaz, 2020). As newer and more efficient function approximation methods are discovered and showcased in the DRL literature, such methods gradually make their way into the field of RL-TSC. Two major types of DRL algorithms have proven the most efficient for a variety of learning tasks: actor-critic and deep Q-learning methods (Gregurić et al., 2020). In both cases, state-of-the-art versions of these algorithms are heavily modified to include several tricks and techniques such as dueling networks, prioritized experience replay, or multi-step learning to increase learning performance and alleviate learning issues (Gregurić et al., 2020). Given the number of additional techniques they employ, these methods are also colloquially known as *rainbow methods* (Hessel et al., 2018). Both versions have applied such rainbow methods to traffic routing tasks, each having its specificities.

In the case of actor-critic algorithms, these methods featured multiple techniques such as natural actor-critic (Richter et al., 2007), tile coding and radial basis function networks (Aslani et al., 2017), advantage actor-critic (A2C) (Chu et al., 2019; Xiong et al., 2019), asynchronous advantage actor-critic (A3C) (Genders and Razavi, 2018) or fuzzy radial basis function (Chun-Gui et al., 2009). As for deep Q-learning techniques, rainbow-type techniques using double Q-learning and coordination graphs (Van der Pol, 2016), image-type state representation, and convolutional neural networks (Shabestary and Abdulhai, 2018), recurrent neural networks (Shi and Chen, 2018) have also been applied. While the current DRL is not yet definitive about which algorithm structure provides the best results since they can widely differ depending on the learning task, duelling double deep Q-network (3DQN) algorithms seem to provide excellent learning capabilities in a wide array of learning tasks (Hessel et al., 2018).

This chapter has given a large overview of how reinforcement learning models described in chapter 3 can be applied to traffic signal control tasks. By analyzing a large array of papers from the RL-TSC literature, which includes a number of literature reviews and surveys, we were able to identify the most common modeling choices of RL-TSC models.

In the area of environment modeling, we have shown that intersections are almost always used as learning agents who often aim to minimize vehicular delay on the road network. We have also shown that a wide array of features of the environment could be used as components of the MDP. Similarly, MARL TSC models could both successfully feature independent learning and agent coordination. Regarding RL algorithms used by papers of the literature, we have identified that multi-agent and model-free methods were widely more popular than their single-agent and model-based counterparts. Among these methods, TD algorithms such as Q-learning are broadly used in the literature. Finally, we have shown that function approximation techniques have become a mandatory feature of any modern RL-TSC method given their efficiency and that actor-critic and deep reinforcement learning methods provided excellent results in recent works.

Part II

Model

After describing how reinforcement learning and traffic signal control are used in the literature to optimize traffic in various ways, we aim to replicate, explain and extend these traffic signal control methods. In order to undertake these tasks, the second part of this thesis focuses on formally defining the environment in which our learning problem occurs. This modelization process is done incrementally. Since we optimize traffic through traffic signal control, our first task is to properly define road networks and traffic in a simple mathematical model (chapter 5). Once this modelization task is completed, we then describe how the learning agent interacts with its newly defined environment by dealing with learning-related aspects of our model (chapter 6). Finally, we present the simulation framework in which they will be applied in practical terms, hence completing the description of our RL-TSC framework (chapter 7).

5 Traffic Model

The first–and often forgotten–necessary step to study traffic light control systems is to define what traffic *is*. Consequently, this chapter presents a simple discrete-time traffic model based on graph theory upon which we will be able to adequately describe the RL-TSC methods to be used in later parts of this thesis. The first section of this chapter describes road networks as graphs formed of vertices and arcs. The second section describes how we model vehicular movements and traffic signals on the vertices of a road network. The third and last section defines how we model traffic flows on this road network.

5.1 Road Network

Graph theory (Berge, 1973) provides a good set of tools to model a traffic network. The arcs of a graph are quite similar to streets or roads, and the same is true for vertices as intersections or junctions. Hence, we extend these concepts from graph theory to define the static part of our traffic model: its *network*.

5.1.1 Graph

Let $G = (V, A)$ be a directed multigraph (or *multidigraph*) representing a road network, where V is the set of vertices and A the set of arcs of G. An *arc* $(u, v) \in A$ (also denoted by uv) represents a connection from vertex u to vertex v on G. The road network G is modeled as a multigraph so that multiple arcs can link the same two vertices, similarly to lanes on streets of a road network. These connections are used by vehicles to move from vertex to vertex across the road network. Conversely, a *vertex* $v \in V$ is a point connecting multiple arcs of G. Figure 5.1 shows an example of how graphs can model road networks. A *path* over G is a sequence of arcs $a_1, a_2, \ldots a_n$ of A

of G indicating a valid route from a vertex u to a vertex v of V. We assume here that G is *strongly connected*, which means that for any pair of vertices $(u, v) \in V^2$, there is a path in G connecting u to v. The strong connectivity property makes it possible to reach any point of G from any starting point within the network, which is a fair and necessary assumption regarding road networks.

Figure 5.1: Example non-directed graph representing a road network.

5.1.2 Vertices

The *indegree* and *outdegree* of a vertex v respectively refers to the number of arcs going towards and out of v. Since G is strongly connected, the indegree and outdegree of any vertex of V is at least 1. The set of vertices connected to a vertex v of G is defined as its *neighborhood* and can split up between the incoming and outgoing neighboring vertices of v, respectively noted $\Gamma^-(v)$ and $\Gamma^+(v)$.

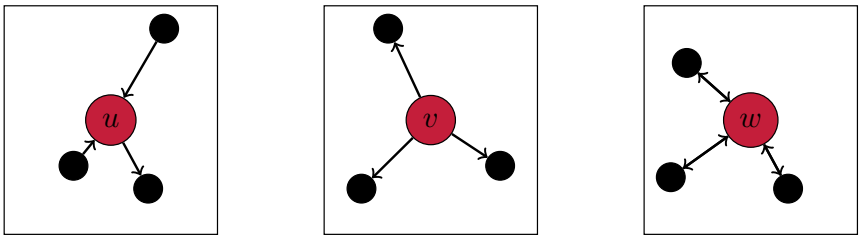

Figure 5.2: Example node degrees. Node u has an indegree of 2 and an outdegree of 1. Node v has an indegree of 0 and an outdegree of 3. Node w has an indegree and outdegree of 3.

5.2 Traffic Signals

Traffic signals are essential in modeling a road network. To our knowledge, no works of the RL-TSC literature formally define the way traffic light controls operate over an intersection. However, a tremendous amount of technical traffic literature exists regarding the design and operation of traffic light systems, either stemming from local traffic authorities (Koonce and Rodegerdts, 2008; Sullivan et al., 2015) or traffic simulator documentation (Erdmann and Krajzewicz, 2013). Based on this literature, this section introduces a simple model discrete-time of traffic flows over vertices of a road network.

5.2.1 Traffic Trajectories

A *traffic trajectory* $\overline{uw}$ over an intersection $v \in V$, which is composed of an incoming arc $uv \in A$ and an outgoing arc $vw \in A$, represents the trajectory of a vehicle going from arc uv to arc vw by crossing intersection v. Two traffic trajectories are said to be *compatible* if they do not overlap each other on the crossing area of the intersection since it could cause an accident. Traffic engineering aims to combine multiple traffic trajectories over an intersection while ensuring their compatibility. Note that the formal definition of trajectory compatibility is beyond the scope of our model and that compatible trajectories over an intersection are given as input through a compatibility table (see Table 5.1).

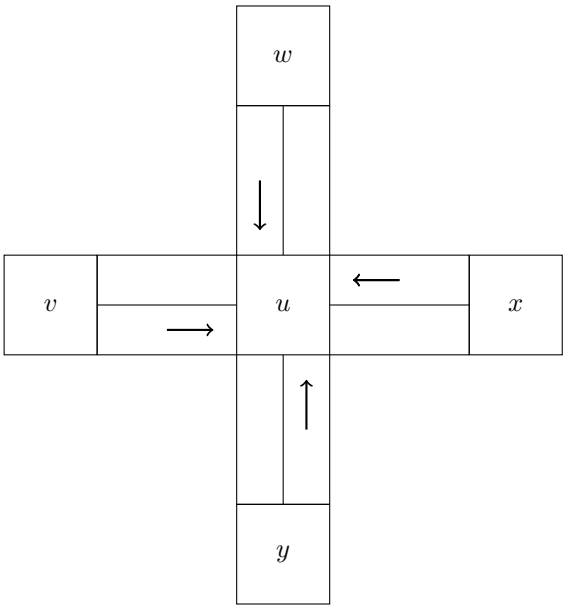

Figure 5.3: Illustration of a 4-way intersection.

Consider, for instance, an intersection of degree 4 displayed on Figure 5.3, and its associated traffic trajectory incompatibilities in Table 5.1. This table shows that right-turn traffic streams (e.g. $\overline{yx}$, $\overline{wv}$) are only incompatible with trajectories with the same outgoing arc. Conversely, left-turn lanes (e.g. $\overline{yv}$, $\overline{wx}$) are incompatible with all other traffic streams on the intersection,

excepted its symmetric trajectories (e.g. $\overline{vy}$ and $\overline{yv}$) and the opposite side right-turn (e.g. $\overline{yv}$ and $\overline{xw}$). This latter observation underlines the importance of *left-turn* trajectory over standard 4-way intersections.

	$\overline{vw}$	$\overline{vx}$	$\overline{vy}$	$\overline{wv}$	$\overline{wx}$	$\overline{wy}$	$\overline{xv}$	$\overline{xw}$	$\overline{xy}$	$\overline{yv}$	$\overline{yw}$	$\overline{yx}$
$\overline{vw}$					o	o	o	o		o	o	
$\overline{vx}$					o	o			o	o	o	o
$\overline{vy}$						o			o			
$\overline{wv}$							o			o		
$\overline{wx}$	o	o					o		o	o	o	o
$\overline{wy}$	o	o	o				o		o	o		
$\overline{xv}$	o			o	o	o				o	o	
$\overline{xw}$	o										o	
$\overline{xy}$		o	o		o	o				o	o	
$\overline{yv}$	o	o		o	o	o	o		o			
$\overline{yw}$	o	o			o		o	o	o			
$\overline{yx}$		o			o							

Table 5.1: Traffic trajectory incompatibilities on a 4-way intersection.

There are two common ways of addressing the specific and conflicting case of left-turns in traffic engineering (Koonce and Rodegerdts, 2008). The first kind of left-turns are *permissive* left-turns in which left-turning vehicles have a right of way on the crossing area. In the case of permissive left-turns, such vehicles station on the crossing area until they can safely cross in the absence of vehicles from these other streams. The second kind of left-turns are *protected* left-turns, in which left-turning vehicles are associated with a specific traffic signal and arc. The choice of implementing a permissive or protected left-turn is usually the result of a warrant analysis on a per-intersection basis (Sullivan et al., 2015), and is most of all the result of a trade-off between traffic safety and lower intersection capacity. Even though it is essential to mention the importance of left-turns in traffic signal control, our traffic model does not require to specify whether an intersection uses protected or permissive left-turns. In both cases, a left-turn (e.g., $\overline{yv}$ on Figure 5.3) is being represented in the same manner whether it is protected or permissive without impacting the rest of the traffic model. The actual left-turn type over an intersection, which depends on warrant analysis, number of lanes, and historical traffic flows, is left as an implementation detail discussed in chapter 7.

5.2.2 Traffic Phases

We designate by traffic *phase* over a vertex v a set of traffic trajectories on v that are all mutually compatible, meaning that vehicles following trajectories of these phases could safely do so simultaneously. Note that a phase can contain any number of compatible trajectories and that the same trajectories of an intersection can be grouped in different phases (see section 5.2.3.1). The principal type of traffic phase, also known as green phase, associates a green light signal with a set of trajectories $\phi_v = (\overline{xy}, ..., \overline{yz})$, giving them the right to cross intersection v. Each green phase ϕ_v is associated with a yellow (or amber) phase ϕ'_v. An amber phase associates each trajectory of

ϕ'_v with a yellow signal, which allows vehicles on these trajectories to cross the intersection while warning them that their right of way on the intersection is expiring and that they should decelerate accordingly. These amber phases are essential in avoiding collisions on the crossing area due to emergency braking by going directly from a green to a red signal. Finally, the red phase is a specific phase containing no trajectories at all: $\phi_v^0 = \emptyset$. Using a red phase is necessary for safety reasons by ensuring that vehicles crossing the intersection during an amber phase have time to go through before the next green phase becomes active.

5.2.3 Signal Cycles

A *signal cycle* on a vertex v is an periodic sequence $\Phi_v = (\phi_1, \dots, \phi_n)$ of traffic phases on the intersection. Signal cycles aim to efficiently organize the successive right of ways of multiple compatible traffic trajectories on an intersection over time. A signal cycle can be decomposed into a static structure (i.e., how phases are organized to form a signal cycle), which we present first. Once this structure is defined, a signal cycle can associate phases with phase durations, which dictates how a signal cycle changes over time.

5.2.3.1 Signal Cycle Structure

If a signal cycle Φ_v could potentially be a sequence of any phases over an intersection, it must satisfy two key constraints to be considered *valid*:

1. Each possible traffic trajectory over intersection v must appear at least once in the phases of the signal cycle Φ_v. This constraint is a necessary extension of the strong connectivity property on the graph G, since it ensures that for any two neighbors of v, $u \in \Gamma^-(v)$, $w \in \Gamma^+(v)$, w is reachable from u.

2. Each green phase ϕ_v of a valid signal cycle Φ_v must be directly followed by its yellow phase equivalent ϕ'_v, which must itself be directly followed by the red phase ϕ_v^0. This second constraint comes from the safety requirements stated above.

Intersections can broadly be categorized into two categories, depending on how they implement signal cycle rules. Intersections implementing a *fixed phasing scheme* maintain the same phase ordering within successive applications of their signal cycles. While the respective duration of each phase can vary between signal cycles (see the following subsection), the ordering of phases within the signal cycle cannot change. Conversely, intersections a *variable phasing scheme* can both change the duration and order of phases within their signal cycles, provided that the two constraints stated above are respected.

An important point to note is that an intersection can have multiple valid signal cycles. However, a signal cycle being valid does not necessarily induce that it is *adapted* for a given intersection. Consider, for instance, The 3-way intersection displayed on Figure 5.4. In the case of this intersection, multiple valid signal cycles can be defined. For instance, a signal cycle can use green phases successively granting a right of way to all incoming arcs of the intersection: $\phi_1 = (\overline{vw}, \overline{vx})$, $\phi_2 = (\overline{wv}, \overline{wx})$, $\phi_3 = (\overline{xv}, \overline{xw})$. Another valid signal cycle could consist in using green phases giving a right of way to successive pairs of arcs of the intersection $\phi_1 = (\overline{vw}, \overline{wv})$, $\phi_2 = (\overline{wx}, \overline{xw})$,

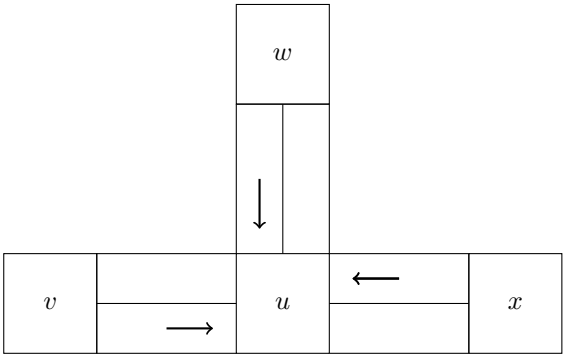

Figure 5.4: Illustration of a 3-way intersection.

$\phi_3 = (\overline{vx}, \overline{xv})$. A third signal cycle could also use green phases giving a right of way to all traffic trajectories one by one successively: $\phi_1 = (\overline{vw})$, $\phi_2 = (\overline{vx})$, $\phi_3 = (\overline{wv})$, $\phi_4 = (\overline{wx})$, $\phi_5 = (\overline{xv})$, $\phi_6 = (\overline{xw})$. It is, however, clear that this last signal cycle would be much less efficient at routing traffic than the two other proposed signal cycle since it only allows for a single traffic trajectory at a time while the others use two. This illustrates the fact that a valid signal cycle is not necessarily efficient.

5.2.3.2 Signal Cycle Evolution

Defining an ordering of phases within a signal cycle is insufficient to route traffic since traffic trajectories occur over *time*. Our traffic model defines the evolution of the traffic light signals over an intersection in discrete time, divided in time *steps* of equal length. Given a signal cycle $\Phi_v = (\phi_1, \ldots, \phi_n)$ on v, we designate by $\phi_t(v)$ the phase of Φ_v that is active at time step t, and by $d_t(v)$ the amount of steps for which the current phase on v has been active within the current signal cycle. The total duration of the signal cycle on intersection v is noted as C_v. At each time step, the traffic light controller on an intersection v can change the currently active traffic light $\phi_t(v)$ if the current phase active time, $d_t(v)$, is superior to a minimum phase duration $d_{\min}$. This minimum duration is usually implemented on intersections for safety and acceptability reasons. Conversely, if the currently active phase $\phi_t(v)$ has been active for $d_{\max}$ steps, it is forced to change at the next time step.

5.3 Traffic Flows

So far, our traffic model has used graph theory to define the structure of the road network and has used traffic engineering to describe discrete-time rules for vehicle crossing of intersections. The final section of this model deals with traffic flows themselves, defining what traffic *is*, and then modeling how traffic flows from vertex to vertex of the road network.

5.3.1 Modeling Traffic

Traffic is composed of vehicles that move over the road network's arcs. Since vehicle movement is continuous, properly defining it in a discrete-time model is highly difficult. While it is possible

to approximate vehicle movement using, for instance, cellular automata (Nagel and Schreckenberg, 1992), we do not opt for this option for two reasons. First, traffic experiments of this thesis are based on the SUMO traffic simulator (Lopez et al., 2018), which does not use a cellular automata model, which would mean that our theoretical traffic model would not match our experimental setup. Second, most traffic simulators, including SUMO, use advanced microscopic traffic simulation models (Chowdhury et al., 2000), which are much more precise and advanced than simple cellular automatons. These models include, among others, collision (Krauß, 1998) and lane-changing models (Erdmann, 2015). It hence appears much more logical to maintain a discrete-step model of the traffic environment and to delegate the continuous-time management of vehicular movement on lanes of the network to the SUMO traffic simulator, which we present in great detail in chapter 7. The use of a traffic simulator as a black box abstraction is represented by a *transition function* T, whose exact role is detailed later on in this section.

5.3.2 Vehicles and Lanes

A vehicle is formally defined as a tuple $c = (p, e) \in A^n \times \mathbb{N}$ where p is the path followed by the vehicle on graph G from its entry to its exit arc and e the time step of access of the vehicle on the network. The path p is computed on the road network graph using Dijkstra's shortest path algorithm and does not account for other vehicles present on the network. Each vehicle aims to follow its path p on the network graph G in order to exit the network through vertex v with a minimal waiting time. The waiting time of a vehicle is defined as the number of time steps the vehicle has been idle on the road network while following its route p, either due to a red light signal or due to another vehicle present on the network. The cumulated waiting time of a vehicle c at step t is given denoted by the value $\omega_t(c)$, which is computed by the transition function T.

Roads on traffic networks are usually divided into multiple lanes, each allowing for vehicle movement. Since the road network graph, G, is defined as a multidigraph, each lane is represented by an arc linking two vertices of G. Hence, two arcs link the same pair of vertices, similarly to lanes. The *congestion* of a lane, associated with arc $uv \in E$, is equal to the number of vehicles present on this lane at a given time step t, and is noted $c_t(uv)$. The relative position of vehicles within a lane $uv \in E$, which is once again computed using the transition function T, is given by the value $P_t(uv)$ (see Figure 5.5).

5.3.3 Transition Function

Since modeling the movement of vehicles on traffic lanes is a complex task, we have, as stated at the beginning of this section, delegated the management of movement, lane-switching behavior, and entry and exit rules of vehicles on the lane to the SUMO simulator and its associated transition function T. The transition function T, which is reminiscent of the MDP transition function $\mathcal{T}$ (see section 3.1.1), applies the following changes to the road network at each time step:

1. All lanes of G and their vehicles are updated according to the traffic model of the SUMO simulator. If a vehicle changes lanes or exits or enters the network, those changes are reflected on the corresponding lanes of E. Similarly, the waiting time of all vehicles of L is increased by one if they wait during the transition. The values of congestion c_t, vehicle position P_t and vehicle waiting time ω_t are updated accordingly.

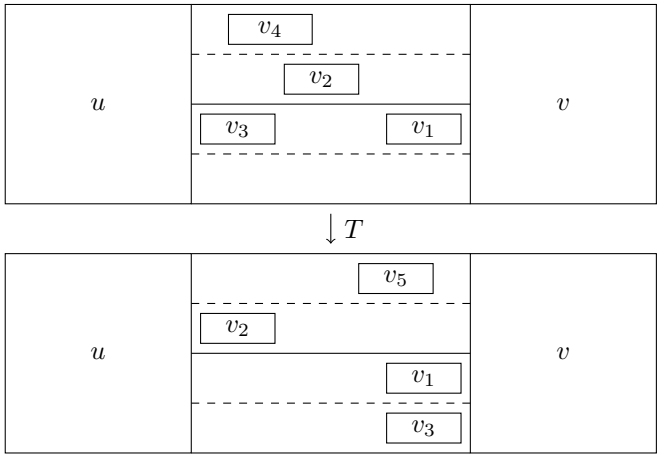

Figure 5.5: Illustration of the evolution of vehicles positions $P_t(uv)$ and $P_{t+1}(uv)$ given by the transition function T. The transition function caused the departure of vehicle v_4 and arrival of vehicle v_5. Transitions also direct lane changes, which is illustrated by vehicle v_3.

2. The signal cycle of all network traffic lights is advanced by one step. Phases are automatically switched if the current phase duration exceeds maximum phase time $d_{\max}$ or if the current phase duration is attained.

3. The current simulation step t is increased by 1.

While many more parameters are considered in actual traffic simulations (see chapter 7), the simplified model presented in this section allows to precisely describe traffic phenomena related to traffic signal control. This model, in turn, helps to properly define the reinforcement learning framework applied to traffic signal control.

This chapter introduced a formal traffic model to help us describe the dynamics of a road network.

This model first defined the structure of the road as a multidigraph in which vertices are intersections and arcs are lanes. It then described the movement of vehicles over the network in order to define how phases and signal cycles are organized over an intersection. Finally, we described the temporal dynamics of the road network. This description includes the movement of vehicles over lanes and the overall network transition, which is managed through a black-box transition function T. This transition function depends on the `SUMO` traffic simulator, which is presented in detail in chapter 7

6 Learning Model

The traffic model defined in chapter 5 allows to easily manipulate traffic-related concepts when applying them in a learning setting. This section builds upon this foundation by formulating the learning problem at hand–routing traffic using traffic signal control–using elements from this traffic model. As we have seen in chapter 3, the standard framework used to represent reinforcement learning problems is a Markov Decision Process. Consequently, this chapter defines each necessary component of our RL-TSC model. It first defines the global objective function $\mathcal{F}$ to be optimized by the agent. It then defines each component of the MDP 4-uple, namely the state space $\mathcal{S}$, the action space $\mathcal{A}$, the reward function $\mathcal{R}$ and the transition function $\mathcal{T}$ used to model the framework and solve the objective function $\mathcal{F}$. As we presented most modeling options used by RL-TSC methods of the literature in section 4.1, this chapter aims to underline the impact of choosing different traffic models to decide which representation is the most adapted to our needs.

6.1 Objective Function $\mathcal{F}$

When applying a reinforcement learning method to a given problem, the first and most crucial question is which objective function the agent should optimize. In the case of traffic signal control, the rather vague term "optimizing traffic" can refer to widely different goals, such as minimizing delay or congestion, but also noise and CO_2 emissions.

6.1.1 Role of the Objective Function

One crucial point to bear in mind is that the objective function has to be the first model component to be defined since all elements of the MDP are dependent on it. Indeed, we have seen that the reward and objective function have to be tightly correlated in order for the agent to learn. Furthermore, state and action space definitions are also highly dependent on the objective the agent is trying to solve. For instance, if a reduction in CO_2 emissions is the main objective of a RL-TSC model, the components of the MDP will have to be chosen to suit this objective. Not only the reward function $\mathcal{R}$ will have to incorporate CO_2-related variables, but the state space definition will also likely incorporate features of the environment that are relevant to this goal. This observation also implies that the different parts of the MDP that we define in this section are chosen with regards to a specific objective function and are not likely to be optimal in other contexts. Also, note that in the case of RL-TSC methods, the objective function of the agent is often directly used as a performance metric to estimate the problem-solving ability of the agent (Mannion et al., 2016). In other words, the better the agent learns how to optimize the objective function, the better the associated performance metric will be.

6.1.2 Choosing the Objective Function

While there is no right and wrong answer when choosing an objective function, some traffic metrics are usually more relevant than others. Both the classical and RL-based traffic signal control literature indicate that there are two main ways of optimizing traffic: through minimization of delays or minimization of congestion (Koonce and Rodegerdts, 2008). Both objectives have their virtues and limitations, and their selection is usually dependent on the goals of local traffic authorities (e.g., some areas favor high-speed traffic flows and minimized delays, while residential areas might favor limited speeding and noise). Since our experimental framework will feature multiple road networks with different geometries, we make the design choice of selecting the most common objective function (i.e., that will be applicable in all traffic scenarios). While congestion reduction might be more beneficial in some specific scenarios, delay reduction has a broad range of applications that will yield satisfactory–if maybe not always optimal–results. This choice is shared in the RL-TSC literature, as almost 50% of the papers surveyed by Noaeen et al. (2021) aimed at minimizing delay or travel time while minimization of queue sizes, maximization of speed, and throughput accounted for 12, 6 and 6% of papers respectively.

The last decision regarding to the model's objective function is choosing which time-related measurement to minimize. As stated in the literature review in chapter 4, there are three main types of time-related variables in vehicular networks: delay (i), travel time (ii), and waiting time (iii). As it turns out, there is little difference in the three measurements when looking at them from an aggregated perspective. The plots shown on Figure 6.1 show that, on the aggregated level, all three time-related performance metrics variables that are obtainable through means of traffic simulation are entirely equivalent. Hence, the choice of the objective function to use depends on the ease of use of the chosen metric and its applicability. Since the time loss and trip duration indicators can only be obtained once the vehicle reaches its destination, both measurements pose a problem when using them to design the MDP's reward function $\mathcal{R}$ as they caused the rewards to be delayed (Van der Pol and Oliehoek, 2016). Hence, vehicular waiting time, which is readily available

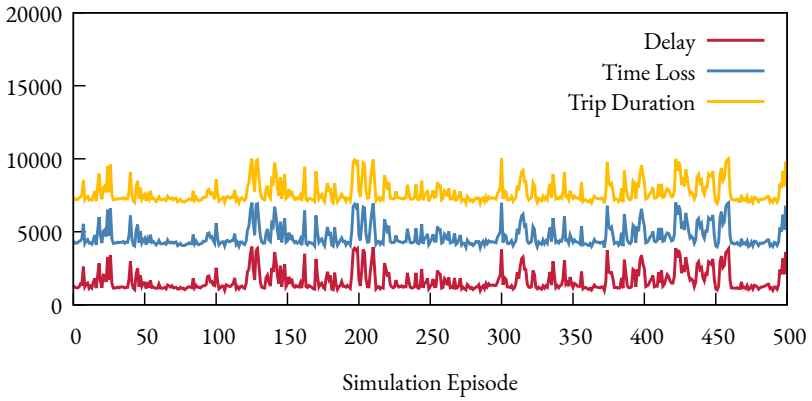

Figure 6.1: Comparison of traffic simulation episodes using three distinct time-related performance indicators. These metrics compute the average trip duration, time loss and vehicular delay for multiple simulation episodes of 500 steps.

in simulation settings at all points of the simulation, provides an adequate objective function for the problem at hand.

6.2 Reward Function $\mathcal{R}$

The first component of the MDP to define, the reward function $\mathcal{R}$, logically follows the definition of the objective function. The reward function aims to help the agent assess whether an observed state-action couple is valuable in maximizing the agent's objective function $\mathcal{F}$. As stated in section 6.1, the chosen reward function $\mathcal{R}$ has to be strongly correlated to the objective the agent aims to maximize as it directly connects the agent's actions to its objective function. Moreover, the influence of an agent acting on a given environment state must have a direct, measurable impact on the reward returned by the system for proper learning to occur. This section investigates which reward function best fits the stated objective of reducing vehicular waiting time.

6.2.1 Choosing the Reward Function

Since the main objective of our RL-TSC model is to reduce the waiting time of vehicles on the network, we logically use the same measurements when defining the reward function $\mathcal{R}$ of the MDP. While the RL-TSC literature showcases an extensive array of possible reward functions, experiments carried out by El-Tantawy and Abdulhai (El-Tantawy and Abdulhai, 2012) comparing the effect of multiple rewards definition on RL-TSC performance found that somewhat simple reward functions performed better. Similar tests used with our experimental setup have also shown the superiority of cumulated delay difference-based rewards. Hence, for an agent placed on intersection $u \in V$, the reward associated with an action a_t is defined as:

$$r_t = \sum_{v \in \Gamma^-(u)} \Big(\sum_{c \in P_t(vu)} \omega_t(c) - \sum_{c \in P_{t+k}(vu)} \omega_{t+k}(c) \Big)$$

In simpler terms, the reward associated with an action a_t for an intersection u is obtained by computing the cumulated waiting time of all vehicles present on lanes directly going from neighbors of u towards u after and before the action was taken (i.e., at time steps $t + k$ and t, where $t + k$ is the first time step at which the agent can decide action a_t) and computing their difference. Such a reward function respects important constraints regarding reward function definition. Measuring differences in cumulated waiting time is obviously correlated with the global objective of reducing the overall cumulated waiting time of vehicles of the road network, and this difference directly quantifies the quality of a given state-action pair. A positive reward indicates that the cumulated waiting time is lower after choosing action a_t, thus indicating a likely correct action choice. Conversely, a negative reward value indicates an increase in cumulated waiting time between those decision points.

6.3 State Space $\mathcal{S}$

In reinforcement learning theory, the state space $\mathcal{S}$ of an MDP is used to describe features of the environment state that are relevant to the agent. Proper state definition is essential in RL problems since the agent uses it to differentiate system states and act upon them accordingly. Since not all features of the environment are relevant to the agent, the main challenge associated with state space definition is choosing *which* features of the environment we should choose and *how detailed* they should be.

6.3.1 Role of the State Space

One of the most commonly faced trade-offs when designing the state space of a RL-TSC model by choosing among the different traffic features is choosing between detailed state representation and exploration efficiency. Indeed, adding more features of the environment in the state space $\mathcal{S}$ potentially leads to better agent performance since it can distinguish different system states better. Still, it also introduces additional dimensionality, which delays the learning process since it increases the size of the state space and hence the duration of its exploration process by the agent. Furthermore, one should note that some components of the environment's actual state are irrelevant for the learning problem at hand or are already embedded in other variables. For instance, the CO_2 emissions of vehicles of the network are likely not relevant when looking at reducing waiting times on the network. The challenge of state representation for RL modelization is hence to identify which features of the environment are relevant when defining the state space $\mathcal{S}$ of the RL problem.

6.3.2 Choosing the State Space

Multiple studies have been made in the RL-TSC literature to measure the impact of state definition on agent performance. El-Tantawy and Abdulhai (2010) compared the performances of an isolated intersection using Q-Learning associated distinct state values (intersection throughput, vehicular delay, and maximum queue length) found that queue and delay-based state representations yielded the best results for cumulative vehicle delay in simulations using traffic data from the city of Toronto. Similarly, Genders and Razavi (2018) have evaluated the effect of state granu-

larity on the performance of an asynchronous advantage actor-critic RL-TSC agent, using three increasingly complex state definitions going from occupancy and speed, to queue and density and finally full DTSE states (see section 4.1.2.3). Their results show that vehicular delay improvements were minimal when using complex state representations and that no differences were observed in throughput or congestion metrics. The authors suggest that increasing state complexity may be beneficial for sufficiently complex function approximation methods such as long short-term memory cells or convolutional neural networks, but not for simpler learning methods.

Similarly to these papers, we compared the performance of multiple state definitions on a wide array of RL-TSC algorithms, ranging from simple classical methods such as Q-learning to deep reinforcement learning algorithms (see section 8.3.3 for a definition). In one case, we used discretized queue data and, in the other case, DTSE occupancy data. The experimental protocol contained an isolated intersection implementing a NEMA-type signal cycle. We found that methods using detailed state representations took two to three times the number of training episodes to reach the same performance levels as methods using simpler state representations. No notable performance improvements were observed once these performance levels were reached. Furthermore, we tested DTSE state representations on even more complex function approximation architectures, such as recurrent or 3DQN networks. We did not observe any significant advantage to using complex state representation. In the light of these findings, our experiments will use discretized state definitions composed of (but not limited to) phase and queue data around an intersection. For a given intersection $v \in V$ composed of n incoming lanes $l_i, i \in [1, n]$, we use the current phase index $\phi_t(u)$, current phase duration $d_t(u)$ and congestion values on lanes around the intersection $c_t(l)$ (see section 5.2.3 and section 5.3) to reach the following state definition:

$$s_t(u) = \langle \phi_t(u), d_t(u), c_t(l_1), c_t(l_2), \ldots, c_t(l_n) \rangle$$

This state definition yields a satisfactory balance between low dimensionality and sufficient granularity by providing both phase and traffic information to the learning agent.

6.4 Action Space $\mathcal{A}$

Selecting the action space $\mathcal{A}$ of a RL problem is equally vital, albeit different from setting its state space. Similarly to the state space $\mathcal{S}$, the action space $\mathcal{A}$ is essential since it defines how the agent can act on the environment to solve its learning task. Hence, different types of action spaces can exist in the case of RL-TSC, event though to a lesser extent than in the state space case.

6.4.1 Role of the Action Space

Interestingly, while being as important as–if not more than–state definition, the effect of action space definition on RL-TSC performance has, to the best of our knowledge, not been studied in the literature. Indeed, multiple types of action spaces are featured in RL-TSC papers. We hence researched and published an in-depth analysis of the effect of action space definition on the performance of RL-TSC controllers (Tréca et al., 2020a) to compare these multiple action types.

6.4.1.1 Effect on Dimensionality

Our analysis compares two types of action spaces: phase-based and step-based actions. Phase-based actions allow the agent to set the duration of the next green phase all at once. When using phase-based actions, the possible action interval for the agent is hence $[d_{\min}, d_{\max}]$, corresponding to the minimum and maximal green phase duration, respectively (see section 5.2.3.2). When using step-based actions, the agent chooses at regular intervals whether to *extend* or *switch* the current green phase. This action space definition hence contains two actions and allows the agent to end the current green phase at any decision point. Note that the state space associated with these two action space definitions is likely to be different. Indeed, step-based actions need to include the duration of the current green phase, $d_t(u)$, in order to know for how long it has been active. This information is not necessary in the phase-based case since no decision is taken while a green phase is active.

Our analysis first compares the effect of these two action space definitions on dimensionality. Since the step-based action space is only composed of two actions, it significantly reduces actions space dimensionality compared to phase-based actions. However, this reduction in action space dimensionality is compensated by an increase in state space dimensionality since step-based actions necessitate the use of the current phase duration d_t in the state definition, contrary to phase-based actions. In conclusion, when using phase-related indicators in the state space of the problem, choosing either step-based or phase-based action types has little influence on the overall dimensionality of the learning problem.

6.4.1.2 Effect on Performance

The second part of our analysis compares the effect of action space definition on the performances of an isolated intersection under different types of traffic demand flows. The SUMO traffic simulator used for these experiments, as well as the protocol used to generate traffic demand data, are described in detail in chapter 7.

This experiment compares a phase-based method to a step-based method which chooses an action at every k step. Additionally, the shape of traffic demand can be changed over the intersection: the overall vehicle arrival rate follows a Poisson process with a fixed arrival rate, but a parameter τ controls the imbalance of arrival rates between the north-south and east-west lanes of the intersection (i.e., a minimum value of τ ensures completely uniform traffic, while a maximum value of τ only allows traffic to occur between the east and west lanes of the intersection). Simulation results have shown that step-based actions are strictly superior to phase-based ones, regardless of the nature of the traffic demand dictated by parameter τ (Tréca et al., 2020a). Furthermore, the analysis of the influence of the step size k between successive step-based actions has shown that smaller step sizes generally yield better performances in even traffic conditions, but that slightly longer decisions windows (e.g., $k = 5$ to $k = 10$) performed better in heavily skewed traffic conditions due to a high parameter τ.

The inherent advantage of step-based methods over phase-based methods can be explained by their very nature: by evaluating whether to extend or switch the current phase every k steps, an agent using step-based actions refreshes its appreciation of the current system state much more frequently than in the phase-based case, in which the agent observes the current system state only once at the beginning of the phase when selecting its action. The decision points shown on Fig-

ure 6.2 illustrate the different rates at which step-based and phase-based controllers get information about the environment. Since the RL-TSC agent observes the current system state much more infrequently than in the step-based case, it cannot adapt as quickly to changing traffic conditions, hence explaining inferior performances.

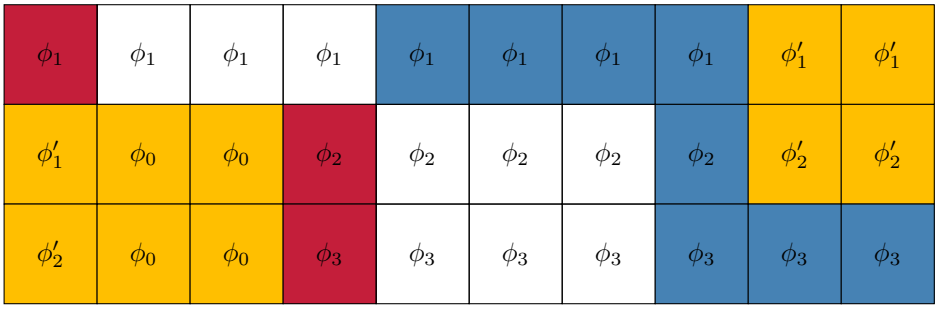

Figure 6.2: Illustration of the decision steps of phase-based (in red) and step-based (with decision interval $k = 1$, in blue) action types on a signal cycle. Yellow steps represent forced transition phases of the signal cycle.

The last phenomenon to explain is the relatively poorer performances of very short interval steps of phase-based actions in heavily imbalanced traffic situations. We attribute these poor performances to the exploration process of the agent. Since the agent favors selecting random actions to explore the state-action space at the beginning of the learning through an ε-greedy policy with a high exploration rate, increasing the rate at which this agent chooses actions through shorter step intervals k mechanically increases its odds of selecting a random action. However, when traffic is heavily imbalanced, the agent should naturally favor longer green phases on east-west lanes and shorter green phases on north-south lanes. By increasing the odds of prematurely ending a normally long green phase through excessive exploration, shorter decision intervals can increase congestion on east-west lanes, impeding overall performance.

6.4.2 Choosing the Action Space

The choice of using a step-based action space over a phase-based one has been motivated by the experimental results presented in the previous section. The step size to associate with step-based action selection we chose is $k = 1$. Indeed, even though longer step intervals performed slightly better in skewed traffic conditions, we have found that the shortest action step interval was the best overall parameter, especially coupled with more advanced reinforcement learning techniques such as deep reinforcement learning (see chapter 8). Hence, the superior results provided by step-based actions coupled with relatively small decision intervals lead us to use a step-based action space:

$$\mathcal{A} = \{0, 1\}$$

in which 0 represents a phase extension action and 1 a phase switch action.

6.5 Transition Function $\mathcal{T}$

The fourth and final point to discuss regarding the modeling of the MDP relating to traffic optimization is the transition function $\mathcal{T}$. The function $\mathcal{T}$ dictates how the environment transitions from one state to the next depending on the agent's action. The transition function, if estimated, offers additional information to the learning agent when selecting an action by estimating the next system state and its potential rewards (see section 3.1.2.5).

6.5.1 Choosing the Transition Model

As stated during the literature review of RL-TSC methods, both model-free and model-based methods have been applied to traffic signal control problems. However, most RL-TSC models choose not to estimate the transition function and are hence effectively model-free. Consequently, while it is technically possible to estimate the transition function of the model's MDP to obtain additional information about the environment, it is commonly accepted that the additional model complexity introduced by switching to a model-free to a model-based method is not worthwhile from a performance standpoint (Mannion et al., 2016). Furthermore, using a model-based method impedes model scalability due to dimensionality issues (El-Tantawy and Abdulhai, 2012). These observations, coupled with the fact that most state-of-the-art RL methods applied to traffic signal control are model-free, make use logically choose a model-free RL-TSC setting in which the transition function $\mathcal{T}$ does not need to be estimated.

This chapter used the traffic model definition of chapter 5 in order to entirely model the MDP components used in our RL-TSC method.

On the basis of the literature review of chapter 4, we established that the objective function of our RL-TSC model was the reduction of the cumulated waiting time of vehicles on the road network. The cumulated waiting time is defined as all the steps for which a vehicle could not advance on the road network, either due to a red traffic light signal or congestion. Consequently, we defined the reward function of the MDP as the difference in cumulated waiting time of vehicles around the lane of an intersection between two successive decision points. We then defined the state space of the MDP as a simple combination of phase-related and congestion features after showing that detailed state representations such as DTSE did not bring increased performance in our model. Similarly, we studied in detail the role of action space definition on agent performance by comparing phase-based and step-based actions. Our analysis has shown that step-based actions were strictly superior to phase-based ones, hence guiding our modeling choice. Finally, like the majority of works of the RL-TSC literature, we chose not to model the transition function of the MDP, resulting in a model-free RL-TSC method.

7 Experimental Setting

The second part of this thesis focuses on modeling how to apply RL methods in a traffic signal control context. In chapter 5, we laid the foundations of a simple mathematical model of traffic signal control, and in chapter 6, we used this mathematical model to define the learning framework in which we apply various RL algorithms on TSC tasks. This chapter focuses on the last element of this iterative modeling work by presenting the tools and methods used to experimentally apply various RL-TSC methods on traffic scenarios. Our experimental setup is composed of two main parts. First and foremost, we use the `SUMO` traffic simulator to simulate traffic scenarios on which we test different traffic signal control methods. Secondly, we designed a simulation library, `carmulator`, in order to integrate a wide variety of RL-TSC systems in `SUMO`. This library also includes many pre and post-processing tools to prepare simulation input and process simulation output for further analysis. Finally, we present how the `SUMO` traffic simulator and the `carmulator` library are used in order to build an experimental protocol used to measure the performance of RL-TSC methods in simulated settings.

7.1 Traffic Simulator

This first section describes the `SUMO` traffic simulator that we use to run our RL-TSC experiments. We first quickly review `SUMO`'s capabilities and features, ending with a justification as to why we chose this simulator. The second section gives an overview of how `SUMO` manipulates data for traffic simulations by looking separately at network, traffic demand, and simulation output data.

7.1.1 Simulator Features

Our traffic experiments are realized using the SUMO traffic simulator (Lopez et al., 2018). SUMO is a microscopic simulator, meaning that each vehicle is managed individually. Furthermore, it is space-continuous and time-discrete, which means that each simulation step in SUMO has a one-to-one correspondence with time steps presented in our traffic model in chapter 5. Finally, SUMO is a multi-modal traffic simulator, meaning that multiple vehicle types and pedestrians can be simulated concurrently.

We decided to use SUMO for multiple reasons. First, SUMO is an actively maintained free and open-source traffic simulator, which means that it is possible to inspect its source code to have insights on some implementation details and that we were able to contribute to its development by submitting bug reports or suggestions to its development team. Second, the SUMO traffic simulator is increasingly popular in the RL-TSC literature. A systematic literature review by Noaeen et al. (2021) shows that the first uses of SUMO in the literature date from 2015, but that 17 out of 27 surveyed papers in 2019 used SUMO as their traffic simulator of choice[1], making it the most popular choice in front of simulators such as VISSIM or PARAMICS. Finally, the SUMO offers unparalleled flexibility when it comes to development and integration with other tools: it offers a Python API to communicate with a running simulation process, `Traci`, and offers a large number of utilities designed to process simulation inputs and outputs, such as traffic demand or simulation logs. The SUMO simulator is written in C++ and uses XML for input and output data format. The simulator relies on two main inputs to run a simulation: a *network file*, which describes the road network over which to simulate, and a *trip file*, which contains traffic demand information over the network. It can also generate several output files and logs in XML format. Additionally to the SUMO simulator, the `sumo-gui` program provides real-time feedback of the ongoing traffic simulation, using speed and visualization options.

7.1.2 Network Data

SUMO uses a network file in order to represent the road network in which simulations occur. These files are composed of a network geometry part, in which the network graph edges are listed, including their length, positions, and the number of lanes. The network's junctions (or vertices) are then listed and are each associated with a traffic light program. This program contains each phase of the signal cycle in a specific state form[2] and the default duration of each phase. Network files can be edited by hand using XML, but SUMO integrates a traffic network GUI editor, `Netedit`, which allows to easily create new road networks and edit traffic light programs on its junctions. The SUMO simulator also provides external tools to convert real-world networks from OpenStreetMaps and convert them to a SUMO XML format.

[1] Quite surprisingly, the same literature review stated that around 16% of the surveyed papers did not state which simulation tools were used at all!

[2] SUMO represents traffic signals in a specific way. For instance, the phase `GGGgrrrrGGGgrrrr` represents the light signal for each intersection's lanes in order. In this example, lanes 1, 2, 3, 9, 10, 11 have a prioritized green signal, lanes 4 and 12 have a permissive left green signal, and other lanes have a red signal.

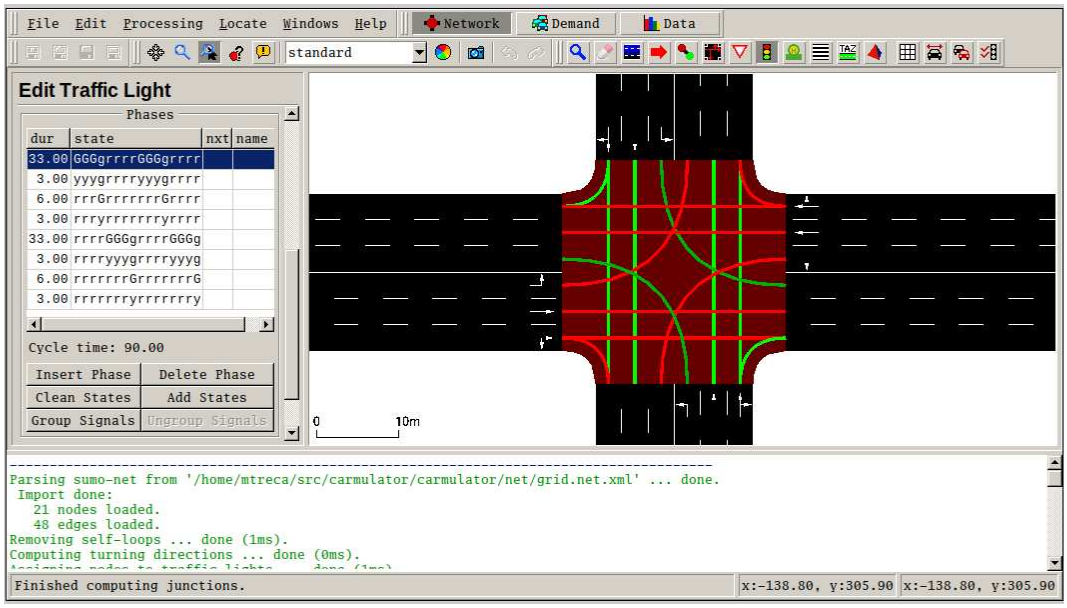

Figure 7.1: The Netedit program, used here to edit the signal cycle of an intersection.

7.1.3 Demand Data

The other central data input needed to simulate traffic is the *demand data*, which indicates the number of vehicles and their trajectories going through the road network. Multiple types of demand data are accepted by SUMO (Urquiza-Aguiar et al., 2019). The first and simplest form of demand data is a *trip*, represented by an origin and end edge of the network and a start time. When simulated, SUMO will compute the shortest path on G (as defined in section 5.1.1) going from the origin to destination edge of the trip using Dijkstra's algorithm and use this route for the vehicle. The second demand data type is a *route*, defined as a trip with a pre-computed route, which hence does not need route computation during the simulation. Finally, SUMO accepts *flow* definitions, composed of an origin and destination edge and probability. During simulation, vehicles will be generated between all pairs according to their respective probabilities and computed similarly to trips.

There are many ways to generate traffic data to these three demand data formats, using a wide range of sources. A simple but cumbersome way to obtain demand data is to either write it by hand to an XML file, use the provided utility in `Netedit`, or use a random trip generation tool packaged with SUMO. Demand data generated in this manner is usually highly unrealistic. Flow definitions can provide a more realistic demand definition, either provided manually or converted by a SUMO utility from origin-destination matrices using real-world data. Other demand data sources are road detector data, which computes trips from observed traffic flows at certain observation points of the network or activity data, generated using the `ActivityGen` utility, which generates traffic flows from activity definitions of the network such as population number or type of neighborhood.

7.1.4 Output Data

The SUMO simulator can log many simulation variables for further use, such as trip duration, time loss, route length, waiting time, or number of stops. Additional logging data can include vehicle emissions, vehicle trajectory, lane changes, noise emissions, or battery use. A simulation process does not need any interaction to complete, as the signal cycles defined in the network data files directly control traffic lights. However, it is possible to use the `Traci` API to control these traffic lights on a step-by-step basis directly. The possibility to control a running simulation process through an API was the starting point of the simulation library we created, `carmulator`.

7.2 Simulation Library

We created the carmulator library to interact with the `Traci` API and directly query and control a running SUMO process. This library was designed for multiple reasons. First, it allowed us to quickly prototype and experiment with RL-TSC controllers within SUMO by establishing simple interfaces between the simulator and prototype methods. Second, the `carmulator` library provides several reference RL-TSC methods found in the literature, which one can use for experimentation or comparison purposes. Finally, `carmulator` provides many utilities to make working with the SUMO simulator easier.

7.2.1 Library Structure

The `carmulator` library is a wrapper around the SUMO simulator allowing direct control of traffic lights during a simulation. On startup, `carmulator` initializes a SUMO process and several library-specific data structures such as a simulation supervisor, traffic lights and signal cycles (one per junction), and a global simulation data record. The simulation supervisor then interfaces SUMO and `carmulator` by, on the one hand, querying the current simulation state and making it available to various `carmulator` traffic controllers, and, on the other hand, by transcribing controller actions into traffic signals applicable in SUMO. 1 provides a simplified description of the simulation supervisor.

Algorithm 1: Simplified traffic supervisor loop in `carmulator`.

```
Initialize controllers and signal cycles;
while Vehicles are still present in the network do
    Query SUMO for the current network traffic, waiting times, and traffic signals;
    for Each traffic controller and signal cycle in the network do
        Advance the signal cycle by 1 step;
        if Signal cycle needs a decision then
            Query the controller for a signal cycle action;
            Pass the controller action to the signal cycle;
        if Signal cycle has changed then
            Change the signal cycle in SUMO;
    Move the simulation by 1 step in SUMO;
Log simulation data to disk;
```

Using this supervision architecture has several advantages. First and foremost, interacting with SUMO in a single class delimits interface code from carmulator-only code and limits the risks of bugs and the number of messages passed from SUMO to carmulator[3]. Second, and most importantly, using this architecture allows quickly defining multiple traffic signal control methods. Indeed, all traffic lights use the same simple interface composed of two methods. The set method allows the controller to execute any necessary operation using the current simulation step and the global carmulator data record (for instance, the controller can count the number of vehicles around its lanes at the current time step, verify which signal cycle was in place on a neighboring junction at the previous time step, or send/read a message to/from neighboring intersections). The get method, executed after set, queries the controller for a new traffic signal choice when needed. The controller can choose this signal with information from the network gathered in the set phase. Defining new traffic control methods using these two methods then becomes extremely easy. For instance, defining a fixed traffic signal that switches phase periodically every ten steps is as simple as defining these two methods[4]:

```
def set(step):
   return

def get(step):
   return step % 10 == 0
```

7.2.2 Traffic Generation

Besides simulating RL methods using SUMO, the second central task of the carmulator library is to generate traffic demand data for these simulations. While it is possible to use hand-crafted traffic demand files or SUMO generated data (either through netedit or programs such as duarouter), carmulator provides multiple high-level flow-based traffic generation methods. As stated in section 7.1.3, flow-based traffic demand associates a given edge pair with a vehicle spawn probability evaluated every simulation step (hence describing a binomial process, which is akin to a Poisson distribution for small probability values, which is often the case in our context). The flow values can be generated in multiple ways using carmulator. The first is to supply an origin-destination matrix automatically converted to a matching flow demand data file for a given network. If no traffic data is available for a network, carmulator can generate flow demand from scratch by generating uniform demand flow across all edge pairs of the network, which ensures traffic stability but is rather unrealistic. Another option provided by carmulator is to generate flow probabilities using an exponential distribution of parameter λ, which are more realistic than uniform flows. The exponential parameter λ defines the mean of the drawn flow parameters and the overall traffic intensity. The overall shape of exponential distributions according to different values of parameter λ can be seen on Figure 7.2.

[3] Message passing between SUMO and the Python API is relatively slow, accounting for 30 to 50% of total simulation time for simple TSC methods.

[4] In this scenario, returning a 0 indicates a current green phase extension of 1 step, and returning a 1 indicates a phase switch.

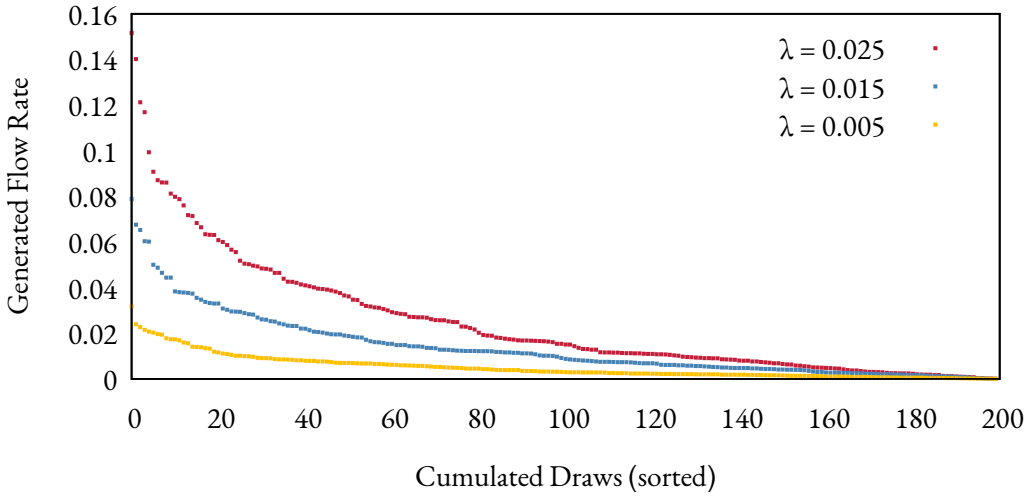

Figure 7.2: Distribution of 200 draws according to an exponential distribution of parameter λ.

7.2.3 Additional Utilities

While easily defining traffic signal control methods and running them using `SUMO` is the primary goal of `carmulator`, the library also provides some additional utilities and methods which proved extremely useful in working with TSC systems.

Comparing multiple TSC methods must be done following the same experimental conditions. In the case of traffic simulations in `SUMO`, the same exact traffic demand file and random number generator seed have to be used to fairly compare methods. Since these constraints are typical in RL-TSC analysis, `carmulator` provides an `experiment_setup` and an `experiment_run` function, ensuring that all methods run for the same amount of episodes the same random seed and demand file are used for each episode across compared algorithms. These methods also generate unique names for all log files of the experiment to be retrieved easily.

7.3 Experimental Protocols

Developing the `carmulator` library has the advantage of making RL-TSC experimentation easy. Indeed, the library provides controller classes that can be used to quickly prototype TSC methods simulation wrappers that can compare multiple TSC methods in the exact same simulation conditions. The `carmulator` library also provides dedicated experimental protocols to analyze RL-TSC controllers. This section covers the three main protocols that are used for the rest of this thesis work in order to compare RL-TSC methods.

7.3.1 Convergence Analysis

We define convergence analysis as the study of the learning capabilities of a RL-TSC agent as learning episodes advance. This analysis is conducted by first generating a set of traffic demand data using the protocol described in section 7.2.2 that will be used across all simulation episodes of the

experiment. This traffic demand is defined in terms of flow, meaning that at each step, each edge pair of the network has a fixed probability of spawning a vehicle following this route. Hence, for the same demand data, using the same random seed will result in the same exact traffic data, while using distinct seeds will result in slightly different traffic data that are still following the general demand pattern. We exploit this property when comparing the convergence of multiple RL-TSC methods. For the same episode index (i.e., the nth simulation episode), the same seed is used across all tested methods, meaning that they all learn using the same exact traffic data. However, between episode indexes, the random seed is changed, ensuring that RL-TSC methods learn on distinct but similar traffic data from one episode to the next.

As for convergence analysis itself, we compute, for each simulation episode, the sum of cumulated waiting times of each vehicle that traveled through the network, giving the total delay of the simulation episode. Plotting these successive delay values from episode to episode, as in Figure 7.3, allow to observe the evolution of the routing traffic capabilities of each agent. For increased accuracy, we usually repeat a convergence analysis over multiple traffic scenarios, each associated with distinct traffic demand data. The plotted result is then the average cumulated waiting time values of these scenarios.

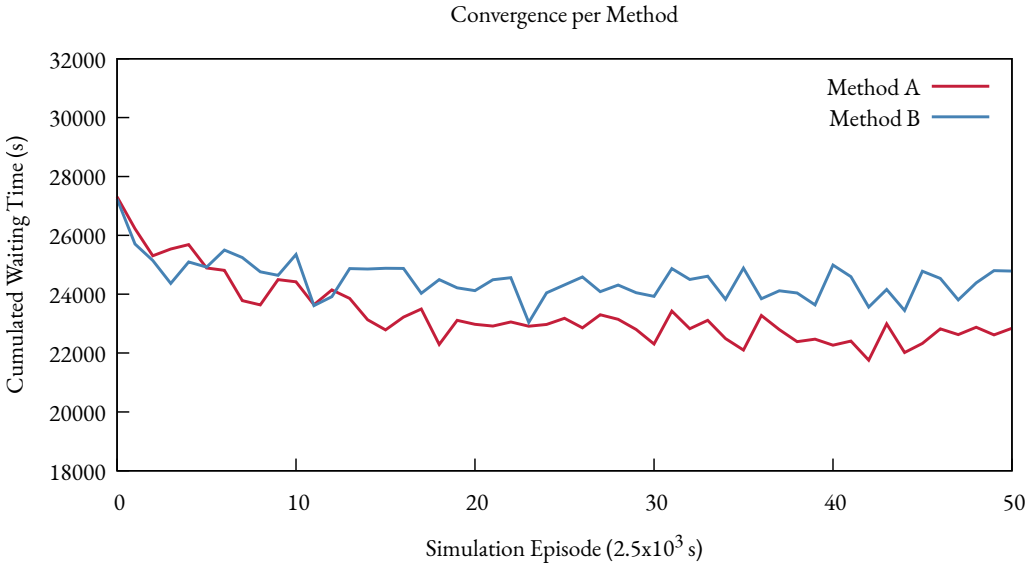

Figure 7.3: Example of a convergence analysis plot.

Note that convergence analysis observes the rate at which a RL-TSC learns *while it is still training*. This implies that convergence analysis is not sufficient to measure the overall performance of a RL-TSC method (which is why we also present performance protocols in section 7.3.2 and section 7.3.3). Furthermore, it also implies that we must decide on a stopping criterion when measuring agent convergence. While there is no hard rule as to when to stop the training of a machine learning model, a generally agreed upon rule is to establish an end of training criteria, usually expressed as a lower bound on the variation of the performance of the agent. Given the unstable

nature of RL-TSC learning, we decide to end agent training if the difference in the average performance of the last n simulation episodes and the n episodes before them is lower than a threshold value κ. The value of parameters n and κ are set, however, on a case-by-case basis since the RL-TSC convergence process greatly differs depending on the traffic scenario and learning method at hand.

7.3.2 Performance Analysis

As mentioned in the previous section, convergence analysis cannot entirely analyze the efficiency of a RL-TSC method. Indeed, this analysis can underline information regarding the learning process of tested methods but can say little about their performance once they have converged. Moreover, learning-specific techniques such as random action selection in the ε-greedy policy (see section 3.1.3.2) introduce sub-optimal action selection choices for the sake of exploration, which can, in turn, affect agent performance. Sub-optimal action selection could introduce a bias in the performance metrics of some RL-TSC methods, especially when one is more likely to explore the state-action space than the other.

We hence evaluate the *performance* of an agent separately from its *convergence* process. While a performance analysis still measures the total cumulated waiting time of vehicles, it measures it *within* instead of *across* simulation episodes. In other words, it plots how the total cumulated waiting time increases as vehicles arrive within the simulation. In order to measure the variability of methods, we plot these metrics over multiple traffic scenarios, each associated with distinct traffic data drawn according to the method described in section 7.2.2. The resulting plot, as displayed on Figure 7.4, features the minimal and maximal cumulated waiting time observed across multiple scenarios for each tested method.

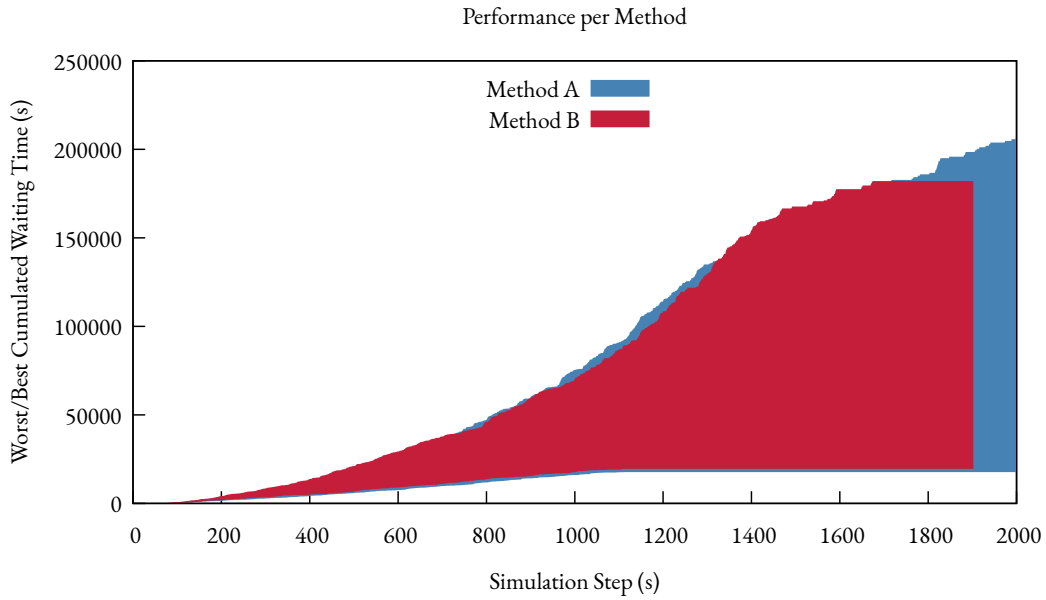

Figure 7.4: Example performance analyis plot.

Note that within a simulation scenario, all vehicles are generated up to a certain time step (e.g., 1500 on Figure 7.4), and the simulation finishes when the last generated vehicle reaches its destination. It is hence possible to have methods associated with longer running simulations, as is the case with method A in the figure above. Finally, performance analysis considers that the RL-TSC methods they compare have finished learning and hence disable their exploration features such as random action selection by setting $\varepsilon = 0$. This ensures that only greedy action selection is in place, which increases agent performance once learning has occurred.

7.3.3 Performance Analysis Under Variable Flows

The convergence and performance experimental protocols allow to study the learning dynamic and post-learning performance of RL-TSC methods. A final experimental protocol of interest to widen our analysis relates to the robustness of these methods. Indeed, even the most advanced RL-TSC methods will not avoid congestion and delays if the traffic demand is superior to a road network's capabilities. However, performant RL-TSC methods can delay congestion and delays as much as possible as traffic demand increases. Similarly, a desirable method can quickly go back to normal traffic conditions once traffic demand decreases. These measurements of robustness hence warrant the constitution of a third experimental protocol that can observe how various methods react to traffic conditions of variable intensity.

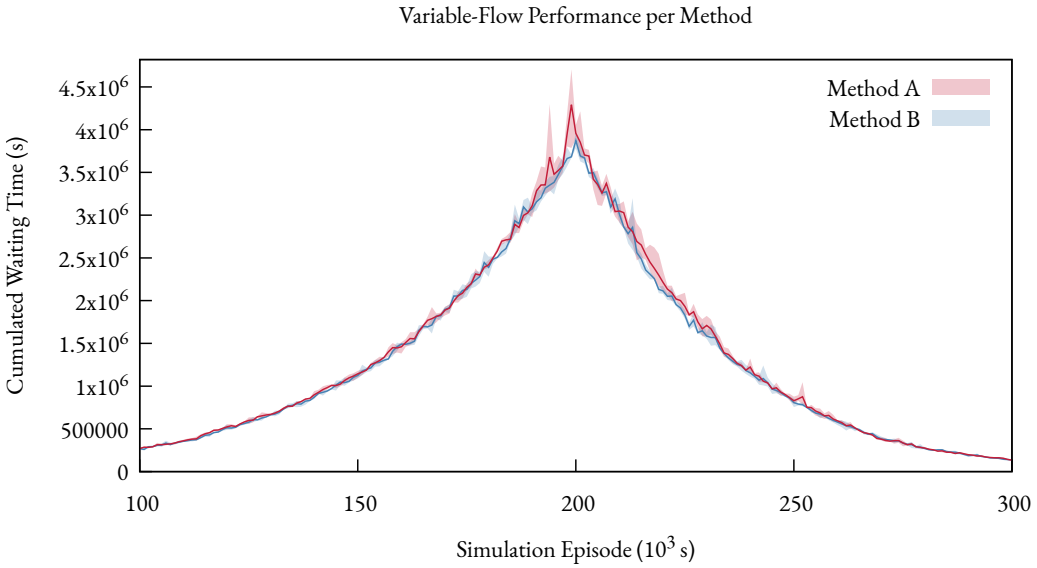

Figure 7.5: Example variable-flow performance analysis plot.

In order to observe both of these phenomenon, we define a variable-flow experimental protocol that is inspired by both the convergence and performance protocols described above. For fixed demand data, all tested methods are first trained similarly to the convergence protocol described above. Then, after these methods converge, the arrival rate λ of each edge pair of the

traffic demand data is increased by 0.8% for each simulation episode over 100 episodes, gradually increasing the total traffic demand while maintaining its overall shape. After the traffic demand peak is reached, the same arrival rate is decreased over 100 episodes by increments of 1% at each episode, returning to a pre-rush hour arrival rate. After running the same experimental scenario multiple times in order to increase accuracy, the resulting plot (see Figure 7.5) shows the best and worst waiting times *per episode* across these distinct scenarios. This final experimental protocol hence combines methods from both the convergence and performance protocols by both looking at inter-episode performance levels (similarly to the convergence protocol) and by plotting best and worst-case cumulated waiting time levels of these episodes across scenarios (similarly to the performance protocol).

In this chapter, we presented the practical tools used to simulate traffic in this thesis work.

We first presented the `SUMO` traffic simulator, an open-source microscopic traffic simulator used to experiment with various RL-TSC methods in this thesis. This simulator was chosen for its availability and flexibility since it offers a Python API that can directly connect to a running simulation process.

We also presented our RL-based traffic simulation library, `carmulator`, which complements `SUMO` by allowing us to quickly develop RL-TSC controllers and test them in traffic simulation settings. This library implements a traffic demand generation utility used to generate demand data over a traffic network. Furthermore, `carmulator` contains a series of experimental protocols used to compare in detail the various RL-TSC methods it implements. The convergence protocol measures the ability of methods to learn across simulation episodes; the performance protocol measures the post-learning performance and performance variability of methods within a simulation episode; while the variable-flow protocol measures the robustness of methods by gradually increasing and decreasing the traffic demand data.

Part III

Method

The third part of this thesis is dedicated to studying different reinforcement learning-based methods used for traffic signal control. Once again, this work is done incrementally as the problem at hand complexifies. We start by discussing multiple isolated traffic signal control methods (chapter 8), featuring both deterministic and learning methods, to analyze each of their components and establish which algorithms and policies are better suited for traffic signal control. Once isolated intersection control is analyzed, we extend our field of study to networks featuring multiple intersections (chapter 9) to study various modes of operations between intersections ranging from independent control to direct coordination mechanisms.

8 Isolated Traffic Signal Control Methods

Isolated traffic signal control methods aim to optimize traffic at the single intersection level, regardless of the actual size of the road network. Since their scope of action is limited, isolated TSC methods are usually simpler to develop and analyze than their coordinated counterparts. This last point leads us to start our analysis of RL-TSC methods on isolated intersections before moving on to coordinated TSC methods in chapter 9. This chapter first covers deterministic isolated TSC methods, which do not use learning to route traffic but will be useful in our experimentations. We then cover multiple classes of classical RL algorithms before looking at function approximation techniques.

8.1 Deterministic Isolated Traffic Signal Control

The first subset of isolated TSC methods is composed of TSC methods which do not use learning mechanisms to route traffic. Deterministic methods regroup, among others, TSC methods implementing a fixed signal cycle repeating itself regardless of traffic conditions, hand-tuned fixed signal cycles designed to maximize intersection throughput, and more complex routing methods.

8.1.1 Fixed Methods

The term fixed traffic signal control methods regroups TSC methods which implement a fixed signal cycle on a given controller regardless of the current traffic situation. The simplest form of fixed

signal control would be an algorithm assigning the same green phase length regardless of phase index or traffic state, even though such an algorithm does not consider the traffic demand around the intersection. During the 1950s, Webster formulated a method to compute the optimal cycle time and split times of an intersection given the traffic demand around it (Webster, 1958). Webster's estimations for optimal cycle time and phase splits rely on computing *critical lanes* for each phase i of the signal cycle, which is the lane with the highest ratio f_i of flow to saturation flow (El-Tantawy and Abdulhai, 2012). Once the critical lanes are identified, the optimum cycle length C in seconds is estimated as a function of the unusable time per cycle L (i.e., amount of seconds dedicated to red phase time) and of the sum of the critical lane flow ratios computed earlier:

$$C = \frac{1.5L + 5}{1.0 - \sum f_i}$$

The optimal green phase time g_i for each phase i of the signal cycle is calculated by distributing the total available green time $C - L$ proportionally to the flow ratio of each phase as

$$g_i = \frac{f_i}{\sum f_i}(C - L)$$

After parameterizing flow values gathered from historical traffic data, the Webster formula allows intersections to implement a fixed signal cycle adapted to their traffic demand. algorithm 2 details the Webster signal cycle formula.

Algorithm 2: Fixed signal timing algorithm using Webster's formula.

for *each step* t **do**
  $i \leftarrow \phi_t(v)$;
  if $d_t(v) < g_i(v)$ **then**
    $a_v \leftarrow 0$;
  else
    $a_v \leftarrow 1$;

While multiple algorithms extend this basis (Rouphail et al., 1998) to more accurately assign green time within an intersection's signal cycle, Webster's original formula provides a good performance indicator of how a real-world parameterized intersection would behave in a traffic simulation setting.

8.1.2 Optimal Method

A particular shortcoming regarding the analysis of TSC algorithms is that there is–to the best of our knowledge–no given deterministic method capable of finding an optimal or near-optimal solution for a given traffic situation at the single intersection level. Furthermore, a common complaint regarding the performance evaluation of learning algorithms is that while it is easy to observe whether a learning method improves over time, it proves more complicated to estimate this improvement with regard to a maximum performance bound for this given problem. A RL-TSC method improving its traffic routing capabilities fourfold through learning iterations does not

measure if this improvement is still far from an optimal–and often unobserved–solution. This issue is common in most papers of the RL-TSC literature since most proposed TSC methods are either compared to fixed or other RL-based methods of the literature (Noaeen et al., 2021), but rarely to state-of-the-art traffic engineering methods used in real-life urban networks. The optimal method we present in this section solves these issues.

8.1.2.1 Optimal Strategy Search

We developed an approximation method that leverages the ability to save and load simulation states in SUMO to alleviate this issue partially. This method considers agent *strategies*, which are binary strings representing successive step-based action choices (see section 6.4.2) by the agent over a certain number of simulation steps. For instance, the strategy `001000` represents two successive extensions of the original green phase, followed by a phase switch and another three successive phase extensions, for a total strategy duration of 15 (a single step per extension action and ten steps for a switch action, corresponding to 5 steps of yellow and red time and 5 steps of minimum green time). The main idea behind this optimal strategy approximation algorithm is the following: when facing an action choice (i.e., whether to extend or switch the current green phase), the algorithm saves the current simulation state to disk and starts testing all possible strategies of length k in a tree-like manner (see Figure 8.1) by successively saving and loading simulation states. After computing all strategies, the algorithm returns the one yielding the best results to the agent, which applies its first h steps. Appendix A provides a complete description of the algorithm.

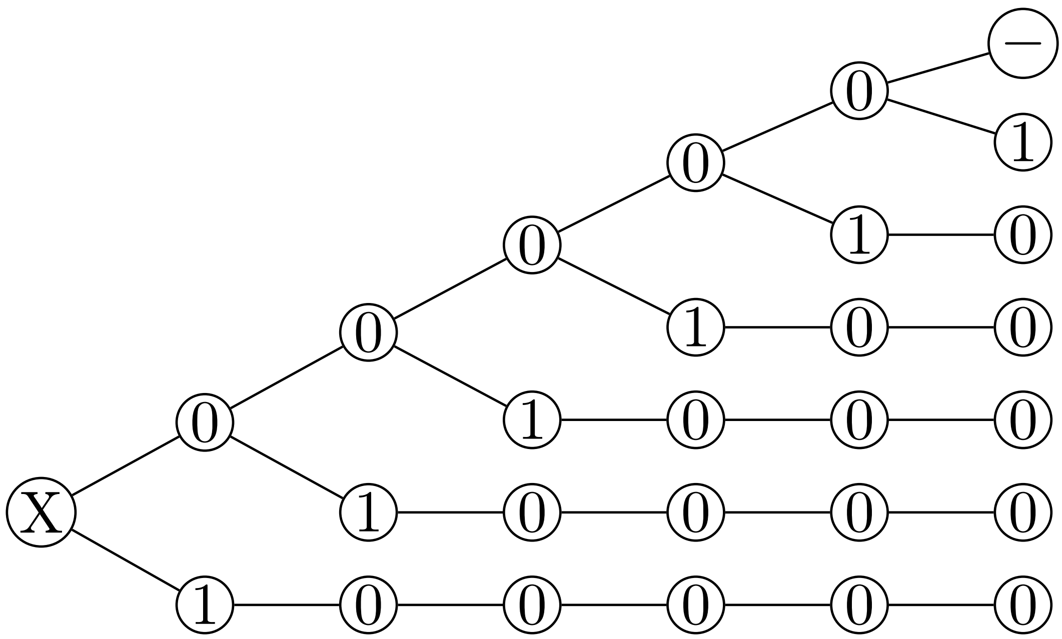

Figure 8.1: Representation of a strategy tree search for a depth $k = 15$. The $-$ character represents 10 successive extend actions for brevity.

8.1.2.2 Dimensionality Issues

A known limitation of exhaustive strategy search methods is their combinatorial explosion when the strategy tree reaches a sufficient length. In the case of traffic signal control, this total number of strategies can be computed by listing all the valid combinations of 0s and 1s that form a strategy string of the desired length and then computing the number of permutations without repetitions in which these symbols can be arranged. Using this formula, we estimate that there are 13 unique strategies of length 20, 3311 strategies of length 50, and around 27 million unique strategies of length 100. Testing the entire strategy tree of a 10.000 steps simulation would require trying 2.13×10^{782} unique strategies, which is entirely above our computational means.

8.1.2.3 Role of Search Parameters

Since an exhaustive optimal search is impossible given the combinatorial explosion of the problem at hand, the role of the strategy depth k as well as the horizon h for which the agent will apply the returned strategy is paramount in finding the right balance between optimality approximation and computational needs for this algorithm. In theory, increasing the strategy search depth k should increase the performance with diminishing returns and increase computation times exponentially (since longer strategies matter less and less regarding the current decision point but dramatically increase the computational search costs). Additionally, increasing the horizon parameter should reduce computational costs and negatively impact agent performance since the entire strategy search process is triggered less frequently. Hence, choosing parameters h and k is a matter of balancing agent performance and algorithm running time. In order to study the influence of both these parameters on the strategy approximation algorithm, we measure the cumulated waiting time obtained in a single-intersection simulation and the total simulation time in seconds for different strategy depth and horizon values for the same traffic and simulation settings.

k / h	1	5	10	15	20	25
10	221	216	228	-	-	-
15	166	226	276	308	-	-
20	161	196	208	220	222	-
25	144	192	145	218	157	157
30	143	167	170	175	222	182
35	138	172	157	167	184	204
40	137	146	167	166	185	188

Table 8.1: Cumulated waiting times according to different strategy depths k and horizon h values.

We present the cumulated waiting time values obtained within the simulations on Table 8.1. We first observe that our predictions regarding the positive influence of longer depths k and shorter horizons h on cumulated waiting time values are respected overall. Nonetheless, some higher horizon values can sometimes outperform lower ones (e.g., horizon $h = 10$ yields a better result than $h = 5$ for $k = 35$), which is likely due to a "lucky run" by specific combinations of k and h; and shows the inherent limitations of optimal strategy approximation methods. The influ-

ence of the horizon parameter also behaves as expected. Increased horizon values above $h = 1$ quickly degrade performance values, even though this degradation is not necessarily ordered or linear for higher values of h. Regarding computational costs, we do notice a substantial reduction in simulation time when increasing agent horizon initially, but this reduction quickly decreases for horizon values above $h = 10$, as one can see on Figure 8.2 for selected values of parameter k.

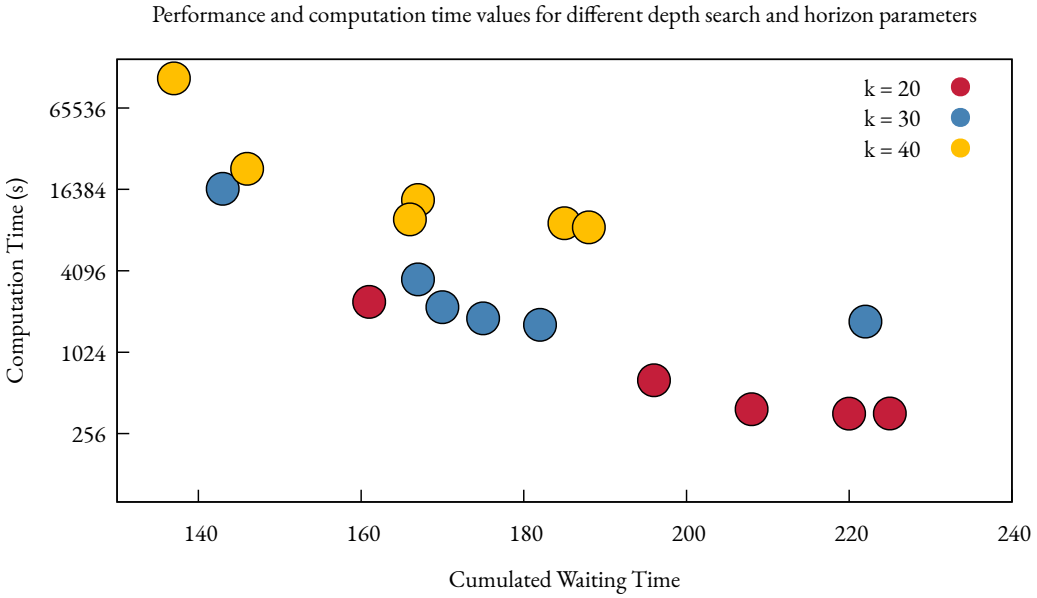

Figure 8.2: Computation time of the optimal strategy approximation algorithm depending on different horizon parameter h values (5, 10, 15, 20, 25, 30, 35 and 40, in increasing order on the above figure).

8.1.2.4 Parameter Recommendations

These results lead us to make the following recommendations regarding the optimal parameter selection. First, we do recommend setting the horizon parameter h to 1. Indeed, as we can see on Table 8.1, setting a minimal horizon parameter is the surest way of obtaining a minimal cumulated waiting time for a fixed depth parameter k, while higher values of h tend to increase the unpredictability of the performance output. Furthermore, we observe on Figure 8.2 that an algorithm of depth k and horizon 1 takes less time to compute and performs better than an algorithm of depth $k + 5$ and horizon 5. Our recommendations regarding the depth parameter k depend on two bounds: a lower depth bound decided by performance, for which we recommend setting parameter k to values of 25 or higher, and a higher depth bound decided by computation time and which depends on the entire length of the simulation. Using results from our test simulations of 500 steps and a fixed horizon parameter of $h = 1$, we estimate that it takes an average of 0.6 seconds per simulation step to run the algorithm for a depth parameter of 10, 4.8 seconds per step for a depth parameter of 20, 33.1 seconds per step for a depth parameter of 30 and 3:37 minutes per step for a depth parameter of 40. Multiplying these estimates by the number of steps necessary to

run a given simulation gives a reasonable estimate of how long a given optimal method will run, which helps select the highest value of parameter k with acceptable computation times.

8.2 Classical Reinforcement Learning Methods

As we have mentioned in chapter 4, multiple types of RL algorithms, such as value-based, policy iteration or actor-critic methods have been successfully applied to TSC tasks (Noaeen et al., 2021). This section hence aims to first present a RL algorithm from each of these classes before comparing them in practice on a single-intersection traffic simulation. Each representative algorithm is voluntarily kept simple to ease the presentation and analysis work of this section; more advanced methods, including function approximation techniques, are presented in section 8.3.

8.2.1 Value-based Methods

Among the three main types of value-based reinforcement learning algorithms presented in chapter 3, only Temporal-Difference learning algorithms are suited to RL-TSC tasks. Indeed, Dynamic Programming methods are *model-based*, meaning that they require prior knowledge or estimations of the transition function $\mathcal{T}$ of the underlying MDP, which is generally considered as a complex modeling task (El-Tantawy et al., 2013; Mannion et al., 2016) and is hence seldom featured in the literature (Noaeen et al., 2021). Similarly, Monte Carlo methods are not used for traffic signal control tasks. Indeed, these methods update their policies and value estimates at the end of an episode, instead of at the end of each step *within* an episode (Sutton and Barto, 2018), making them unfit for tasks like RL-TSC in which fast reactivity within an episode is essential for acceptable performance (El-Tantawy and Abdulhai, 2012).

8.2.1.1 Q-learning

Given its overwhelming presence in the RL-TSC literature and its relatively simple structure, we choose to study Q-learning (Watkins and Dayan, 1992) as the representative value-based method. The Q-learning algorithm estimates the quality of state-action couples of the environment using a general policy iteration technique (see section 3.1.2) and stores these estimates in a Q-table. For $\alpha \in [0, 1]$ and $\gamma \in [0, 1]$ representing the learning rate and discount factor of the agent, the estimated quality of each state-action visited by the agent is successively updated according to the following rule:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a)) \tag{8.1}$$

The intent behind the update rule in Equation 8.1 is rather clear: for a given state s_t and associated chosen action a_t, the estimated quality value $Q(s_t, a_t)$ stored in the Q-table is updated by keeping a fraction $1 - \alpha$ of its old value and a fraction a_t of a newly estimated quality value. This latter term is estimated using the associated reward value r_t (since the long-term reward function is what the quality function approximates) as well as the estimated quality associated with the next system state s_{t+1}, which is computed by estimating the maximal reward the agent could obtain in this new state, $\max_a Q(s_{t+1}, a_t)$. Since this reward is delayed for the agent, a discount factor γ is applied to reflect the agent's decision-making process at step t. algorithm 3 provides an illustration

of the Q-learning algorithm using an ε-greedy policy (see section 3.1.3.2) applied to isolated traffic signal control.

Algorithm 3: Illustration of a standard Q-Learning algorithm with an ε-greedy policy applied to an isolated intersection.

for *each step* t **do**
  Observe s, a, r, s';
  $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$;
  if $X \sim \mathcal{U}(0,1) < \varepsilon$ **then**
    $a' \sim \mathcal{U}(\mathcal{A})$;
  else
    $a' \leftarrow \max_a Q(s',a)$;

8.2.2 Policy Iteration Methods

Policy iteration methods aim at directly estimating the optimal agent policy of a given problem without needing to estimate value or quality functions. Learning automata (LA)–which were originally developed in the field of mathematical psychology (Narendra and Thathachar, 1974)–are functionally equivalent to simple policy iteration RL algorithms (Kaelbling et al., 1996; Nowé et al., 2005). Learning automata were the first RL algorithms to be applied to a TSC problem (Mikami and Kakazu, 1994). Learning automata maintain a policy vector containing the probabilities of selecting a given action in $\mathcal{A}$. After receiving a reward from the environment, the vector p is directly updated to take this feedback into account. Hence, LA circumvent the need for value or quality estimates by directly embedding these values as probabilities in their policy vector. Multiple types of learning automata can be derived from these guidelines and generally differ on three points: their *reward model*, *scheme* and *statelessness*.

8.2.2.1 Learning Automata Parameters

Linear automata usually use three distinct reward models. The P-model is suited for rewards whose values are either 0 or 1, the Q-model when rewards are a distinct collection of symbols, and the S-model when rewards are within a continuous interval (Narendra and Thathachar, 1974). LA schemes also differ on which vector update strategy to apply when receiving a reward. The two most common linear schemes are the linear reward-inaction (LRI) scheme, which only increases probabilities when the reward is positive, and the linear reward-penalty (LRP) scheme, which both increases probabilities if the reward is positive and decreases them if the reward is negative. For $\sigma \in [0,1]$ and $\tau \in [0,1]$ two parameters respectively associated with the reward and penalty components of the linear automaton, these two schemes can be summarized using the update rules below shown on Equation 8.2. Since these update rules guarantee that the policy vectors remain valid probability distributions, the agent policy is directly included in the probability vector in the form of a *stochastic policy*: each action of the action space is drawn according to its weight in the policy vector. While initially using a single policy vector regardless of the system

state (Nowé et al., 2005), linear automata can maintain multiple policy vectors p_s, each associated with a state s of the state space $\mathcal{S}$.

$$p_a \leftarrow \begin{cases} p_a + \sigma r(1 - p_a) - \tau(1 - r)p_a, & \text{if } a = a_t \\ p_a - \sigma r p_a + \tau(1 - r)(\frac{1}{||\mathcal{A}||-1} - p_a), & \text{else} \end{cases} \tag{8.2}$$

8.2.2.2 Modeling Linear Automata for TSC

Regarding the application of LA to RL-TSC problems, several choices have to be made regarding their features. First, state-indexed policies ensure the algorithm behaves differently depending on the current system state, which is crucial in traffic signal control. Hence, the LA applied to traffic signal control will maintain a separate policy vector p_s per system state. The reward scheme used for traffic applications has to feature both negative and positive reward values, according to the reward function defined in section 6.2. We consequently apply a S reward model within the $[-1, 1]$ interval where each reward is linearized to fit in this interval by using the worst and best past observed rewards as the upper and lower bounds of the interval. Finally, the choice of using a LRI or LRP scheme is largely problem-dependent. Our experiences on TSC applications have shown that the LRP scheme was strongly superior to the LRI scheme since it took into account both good and bad action selection decisions. Indeed, by not taking bad action choices into account, the LRI scheme does not learn from wrong traffic decisions even though they are essential in adequately routing traffic. Additionally, the first paper applying reinforcement learning to traffic signal control featured a LRP algorithm (Mikami and Kakazu, 1994). Based on these model choices, and by selecting identical parameters for both rewards and penalties (i.e., $\sigma = \tau$), the LRP algorithm associated with a stochastic policy for isolated traffic signal control is presented in algorithm 4.

Algorithm 4: Illustration of a linear reward-penalty with a stochastic policy applied to an isolated intersection.

for *each step* t **do**
- Observe s, a, r, s';
- **if** $r < r_{\min}$ **then**
  - $r_{\min} \leftarrow r$;
- **if** $r > r_{\max}$ **then**
  - $r_{\max} \leftarrow r$;
- $\hat{r} \leftarrow (r - r_{\min})/(r_{\max} - r_{\min})$;
- **for** a_i *in* $\mathcal{A}$ **do**
  - **if** $a_i = a$ **then**
    - $p_{s,a} \leftarrow p_{s,a} + \sigma\hat{r}(1 - p_{s,a}) - \sigma(1 - \hat{r})p_{s,a}$;
  - **else**
    - $p_{s,a} \leftarrow p_{s,a} - \sigma\hat{r}p_{s,a} + \sigma(1 - \hat{r})(\frac{1}{||\mathcal{A}||-1} - p_{s,a})$;
- $a' \sim p_{s'}$;

8.2.3 Actor-critic Methods

Similarly to policy iteration methods, actor-critic methods establish a policy directly through a policy vector (i.e. the actor); and, similarly to value-based methods, the agent maintains quality estimates (i.e. the critic) and uses them to refine the policy vector iteratively (Grondman et al., 2012). While most actor-critic algorithms used in a TSC context use advanced rainbow-type models leveraging function approximation and other techniques (Gregurić et al., 2020), the general actor-critic framework allows to define simpler schemes that do not rely on continuous state or action spaces or function approximation techniques (Crites and Barto, 1995).

8.2.3.1 Simple Actor-Critic Algorithm

We define an actor-critic algorithm that merges mechanisms from Q-learning and linear automata defined above. Similarly to learning automata, the agent's policy of this actor-critic model is a probability vector p_s associated with a given environment state, s. Additionally, and similarly to Q-learning, this actor-critic algorithm stores quality estimates in a tabular fashion and uses temporal-difference methods to compute these estimates. By using the TD-error formula, which is defined as

$$\delta = r + \gamma q(s', a') - q(s, a) \tag{8.3}$$

where $q(s, a)$ denotes the quality estimate of taking action a in state s, and the critic update rule defined as $q(s, a) \leftarrow q(s, a) + \alpha\delta$ (Crites and Barto, 1995) we obtain the quality function estimate update rule:

$$q(s, a) \leftarrow (1 - \alpha)q(s, a) + \alpha(r + \gamma q(s', a')) \tag{8.4}$$

While similar in appearance to the Q-learning update rule defined in Equation 8.1, it is important to note that in this case, the agent's following action a', or the associated quality $q(s', a')$ cannot be predicted by looking at the quality estimates alone, since they have to take into account the agent's (now separate) policy. It is, however, possible to estimate this future quality estimate by averaging existing state-action estimates weighted by the associated agent policy $p_{s'}$:

$$q(s', a') = \sum_{a \in \mathcal{A}} q(s', a) p_{s', a} \tag{8.5}$$

Equations 8.4 and 8.5 allow to compute quality estimates using a separate policy vector. The policy vector is then itself updated using a linearized TD-error $\hat{\delta}$ on the basis of the maximal and minimal observed TD-error values, $\delta_{\min}$ and δ_{max}. The entire actor-critic algorithm is presented in algorithm 5.

Algorithm 5: Pseudocode illustration of an actor-critic leveraging Q-learning and linear reward-penalty mechanisms with a stochastic policy applied to an isolated intersection.

for *each step* t **do**
  Observe s, a, r, s';
  $\delta \leftarrow r + (\sum_{a \in \mathcal{A}} q(s', a) p_{s',a}) - q(s, a)$;
  $q(s, a) \leftarrow (1 - \alpha) q(s, a) + \alpha \delta$;

  if $\delta < \delta_{\min}$ **then**
    $\delta_{\min} \leftarrow \delta$;
  if $\delta > \delta_{\max}$ **then**
    $\delta_{\max} \leftarrow \delta$;
  $\hat{\delta} \leftarrow (\delta - \delta_{\min})/(\delta_{\max} - \delta_{\min})$;

  for a_i *in* $\mathcal{A}$ **do**
    if $a_i = a$ **then**
      $p_{s,a} \leftarrow p_{s,a} + \sigma \hat{\delta}(1 - p_{s,a}) - \sigma(1 - \hat{\delta}) p_{s,a}$;
    else
      $p_{s,a} \leftarrow p_{s,a} - \sigma \hat{\delta} p_{s,a} + \sigma(1 - \hat{\delta})(\frac{1}{||\mathcal{A}||-1} - p_{s,a})$;
  $a' \sim p_{s'}$;

8.2.4 Performance Evaluation of Classical RL Methods

After presenting methods from three main reinforcement learning algorithm classes, our next task is to measure and compare their traffic optimization performance. This series of experiments compares the Q-learning, linear reward-penalty, and actor-critic algorithm on an isolated intersection. More specifically, the policy and data structures used by these algorithms are compared in order to establish whether one of them is best suited to deal with traffic optimization tasks.

8.2.4.1 Experimental Setup

The comparison of the three classical RL-TSC algorithms of the previous sections is done using the convergence experimental protocol (see section 7.3.1). The traffic demand data is generated using an exponential law of parameter $\lambda = 0.04$ (see section 7.2.2) on a four-way isolated intersection using a NEMA-type signal cycle. Essential simulation parameters, are summarized on Table 8.2.

One could argue that using such a simple road network for these experimentations could render our results meaningless since they differ quite a lot from real-world scenarios. However, we argue that it is exactly because real-world applications are complex that our analysis work should start with simplified traffic scenarios. Indeed, RL-TSC analysis itself seems to suffer from the curse of dimensionality. It is much harder to explain how a given algorithm performs when it is combined with multiple layers of complexity, such as multi-agent learning, agent policy, or function approximation, than when it is used in a simple context. This observation explains why our analysis work is iterative in nature. By using the simplest road network at first, we are able to identify

Parameter	Value
Episodes	500
Steps	2500/episode
Vehicle arrival rate λ	0.04 veh/s
Discount factor γ	0.98
Learning rate α	$0.2 \rightarrow 0.001$
Random action probability ε	$0.9 \rightarrow 0.01$
LRP reward parameter σ	0.5
LRP penalty parameter τ	0.5
Moving window n for stopping criteria	10 episodes
Performance delta κ for stopping criteria	5 sec average

Table 8.2: Simulation hyper-parameters used for classical RL method comparison.

which class of algorithm performs better as well to explain why. Using these results, we can then exclude other RL classes from the analysis as we increase the overall complexity of the model by adding, for instance, multiple intersections or function approximation techniques.

8.2.4.2 Initial Performance Results

We measure a first performance evaluation of the three classes of classical RL-TSC algorithms on an isolated intersection with an overall vehicle arrival rate of $\lambda = 0.04$ vehicles per second, randomly distributed across the incoming edges of the intersection. As one can see on Figure 8.3, the convergence process of all three methods is highly straightforward, all methods triggering their end of training criteria (see section 7.3.1) around the 100th episode (for stopping parameters $n = 10$ and $\kappa = 3$). While each method starts around the same cumulated waiting time levels, they then quickly form a distinct performance hierarchy, with actor-critic (24957 average), linear reward-penalty (24144 average), and Q-learning (22491 average) ranking from worst to best. We note that all the tested methods show an ability to learn to route traffic as simulation episodes advance (although barely in the case of the actor-critic method). If only the classical Q-learning can outperform the fixed Webster controller, it is still far from our optimal method, which has an average waiting time of 19018.

If these results already give us an insight on which classical RL algorithms are adapted to traffic signal control out of the box, they also clearly show the presence of a significant performance gap in performance between otherwise quite similar reinforcement learning algorithms, which raises the question of how one might explain this discrepancy. Two–possibly overlapping–explanations can be given for these discrepancies when looking at RL theory. The first explanation relates to agent policies, since Q-learning, which performs better, uses an ε-greedy policy while both LRP and actor-critic use a stochastic policy. The second explanation relates to the data structure being used by the algorithms in question. Indeed, Q-learning uses a Q-table to store quality estimates, while both LRP and actor-critic use stochastic vector to store information about the quality of state-action couples. Both of these hypotheses are investigated in the following two sections.

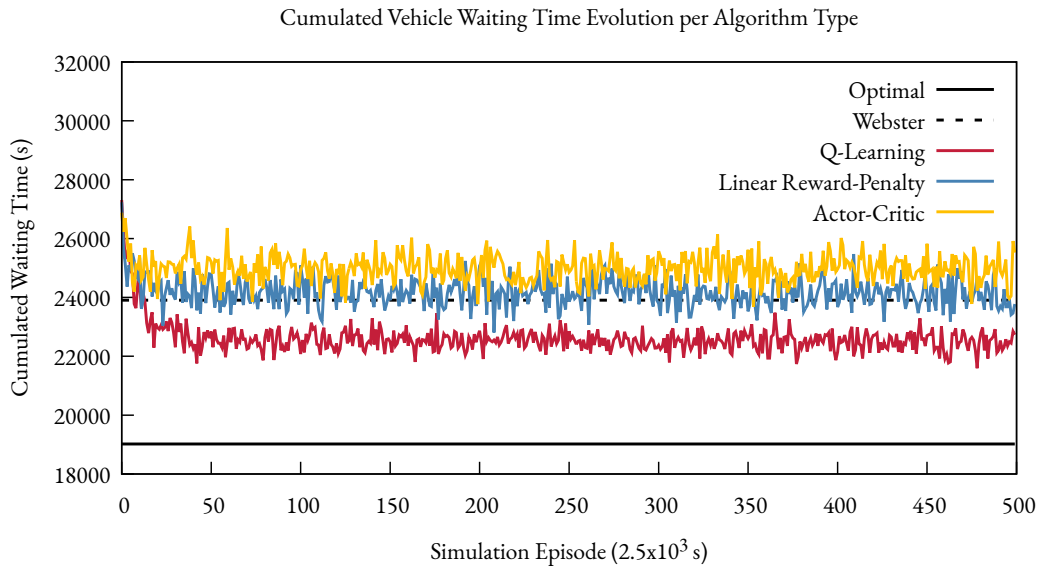

Figure 8.3: Training process of three classical reinforcement learning algorithms on an isolated intersection.

8.2.4.3 Policy Influence Analysis

The first hypothesis we investigate is the different agent policies being employed by these classical RL algorithms. Q-learning uses an ε-greedy policy, while both the LRP and actor-critic algorithms use a stochastic policy which might limit their performances. In order to verify this hypothesis, we run the same traffic scenario as in the previous section while swapping the policies used by the three classical RL algorithms. We use a stochastic policy on a Q-learning algorithm by transforming Q-table rows in probability vectors by linearizing them during action selection. Conversely, we greedily pick the highest probabilities of the policy vectors of LRP and actor-critic policies while maintaining a probability ε of selecting a random action. This experiment aims to estimate whether a stochastic policy is inherently inferior to a greedy-type policy for the three classical RL algorithms in TSC applications.

We plot the results of this experiment on Figure 8.4 by showing the three algorithms using two types of policies: greedy policies are shown in full lines, stochastic policies in dashed lines. These results give contrasted answers regarding our initial hypothesis. On the one hand, switching to an ε-greedy policy sensibly increases the performance of the LRP (in blue in the figure) and–in an even greater fashion–of the actor-critic algorithm (in yellow in the figure). On the other hand, we cannot conclude that stochastic policies are inherently inferior to greedy ones since the Q-learning method (in red in the figure) using a stochastic policy performs slightly better than the greedy version.

Our first experiment, using alternative agent policies to explain the superiority of the Q-learning over LRP and actor-critic methods, cannot fully explain the difference in performance between the three algorithms. However, this same experiment allows to conclude that Q-learning is the

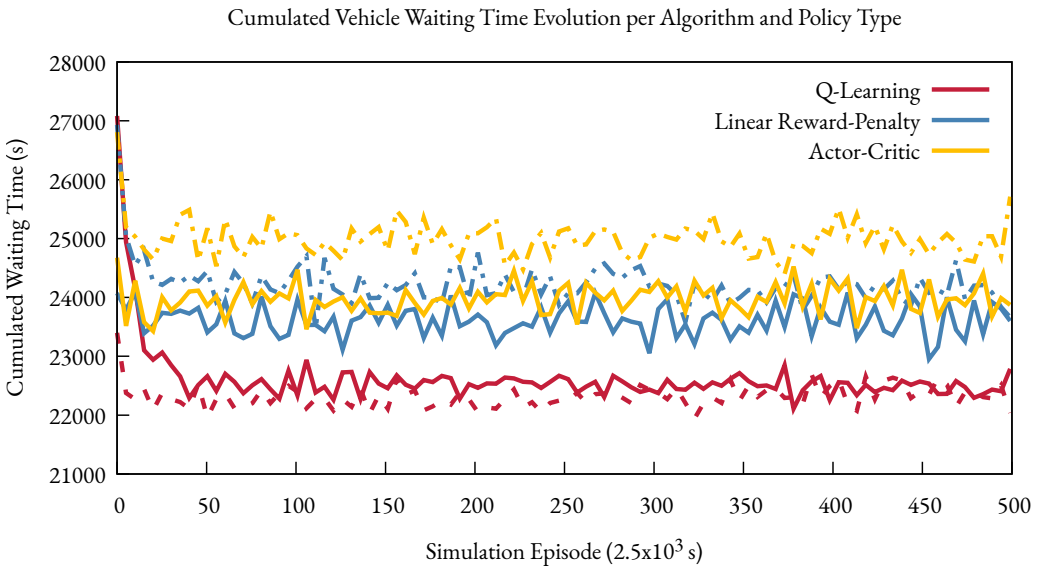

Figure 8.4: Training process of three classical reinforcement learning algorithms using two distinct policy types on an isolated intersection. Dashed lines represent a stochastic policy, full lines an ε-greedy policy. All plots use smoothing splines for readability.

overall best classical RL algorithm for TSC applications given its superiority using multiple agent policies and that it should henceforth be applied to RL-TSC problems.

8.2.4.4 Data Structure Influence Analysis

The second hypothesis we formulated regarding the difference in the performance of the three tested classical RL algorithms is that they employ different data structures to store their learning data. Q-learning stores quality estimates in a tabular fashion, LRP stores probability weights in policy vectors, while the actor-critic method uses both data structures in a hybrid approach. On the one hand, Q-learning stores cardinal values (i.e., an absolute measure of the quality of a state-action pair), while, on the other hand, policy iteration methods store ordinal values (i.e., an order of preference of actions for a given state) in the form of probabilities. We argue that using an RL algorithm using ordinal values introduces several model limitations compared to cardinal values. First, models using ordinal quality estimates are unable to predict the future estimated value of a state, which are expressed in cardinal values, and as represented in the $\gamma \max_a Q(s_{t+1}, a)$ term of the Q-learning formula in Equation 8.1. While the actor-critic algorithm circumvents this limitation by maintaining both a Q-table and a policy vector, the LRP algorithm is unable to determine the quality of a new state and can hence only use the reward value r as a quality indicator of a state-action couple, making it more limited than other methods. Second, one could argue that using a probability vector as a data structure imposes additional constraints on the storage of quality estimates. They must be probability values summing to 1, which forces linearization of the reward and TD-error values, potentially causing information loss.

In order to test this new theory, we run two modified versions of the Q-learning algorithm in the same simulation scenario. A first version disables the role of future values estimates in the computation of quality estimates by setting the discount parameter γ to 0, rendering the learning agent entirely myopic to future rewards. A second version neutralizes the role of future reward estimates and also linearizes the state-action estimates *before* storing them in the Q-table, as if these values were stored in a policy vector, instead of computing them as policy vectors on-the-fly before the action selection process as in Figure 8.4. Finally, we run a modified version of the actor-critic which does not use the policy vector for action selection but tabular quality estimates similarly to the Q-learning, with the exception that these quality estimates are computed using a TD-error formula instead of the classical Q-learning formula. This last method is simulated to verify further if linearization is the root cause for degraded performance in the actor-critic algorithm.

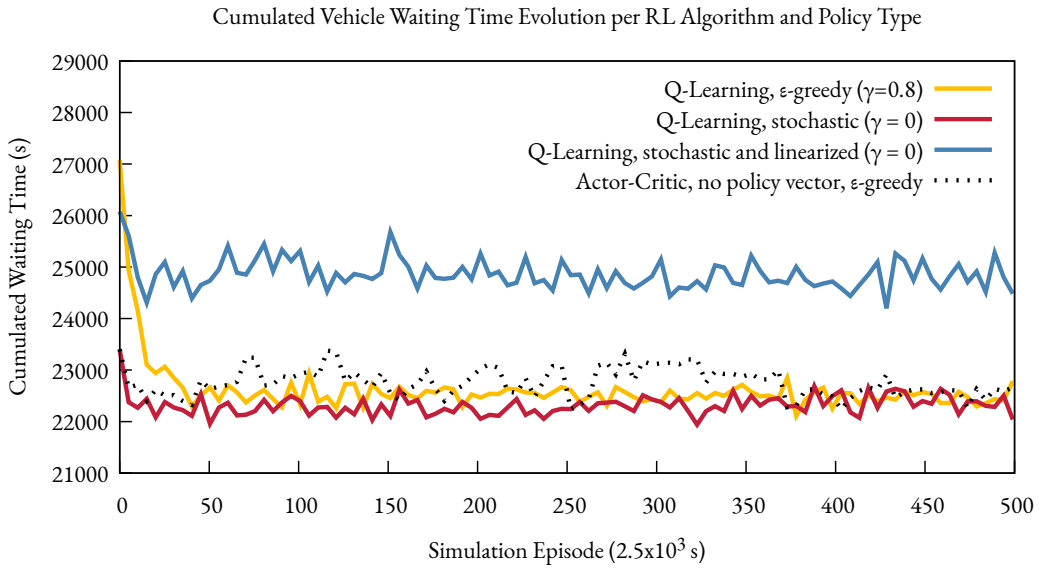

Figure 8.5: Training process of the Q-learning and actor-critic algorithms using data structure variations. Full lines represent multiple variations of the Q-learning algorithm. The dashed line represents the actor-critic policy.

This second experiment, whose results are on Figure 8.5 proves our hypothesis correct. We first notice that the quality estimates of future states, which are neutralized through the discount rate parameter γ, seem to have no influence on the performance of RL-TSC algorithms. Indeed, both the standard (in yellow) and no-future quality estimates (in red) Q-learning algorithms show similar performance levels. This observation shows that, in the context of traffic signal control, the quality of a state-action couple seems only to be dictated by how the chosen action directly impacts traffic and not by how it could make the system transition to a more favorable state. In broader terms, this would imply that most good traffic routing strategies aim at immediately acting on traffic by lowering waiting times instead of aiming for medium or long-term strategies.

The second (and major) result of this experiment is that storing quality estimates in linearized form, effectively switching from a cardinal to an ordinal data structure, dramatically impedes the learning ability of Q-learning. A seen on Figure 8.5, Q-learning using linearized quality estimates (in blue) has performance levels similar to those of LRP on Figure 8.3. Conversely, the actor-critic method using a greedy policy coupled with tabular storage of quality estimates (using a dashed plot on the figure), hence circumventing the use of policy vectors and linearization, performs similarly to Q-learning, further proving the limitations of using ordinal quality estimates for RL-TSC applications. This last observation also shows no significant difference exists between a standard Q-learning estimate rule and the TD-error update used by the actor-critic algorithm. Finally, while this series of experiments has shown that Q-learning is the best classical RL algorithm for isolated traffic signal control, its performance levels are still far behind what the optimal strategy approximation method has achieved in the same simulation setting, with an average cumulated waiting time of 18282 per simulation episode (which is around 20% lower than the standard Q-learning algorithm). This difference in performance underlines the need for more sophisticated reinforcement learning algorithms and techniques, such as function approximation methods, which we describe in the next section.

8.3 Function Approximation Techniques

The role of function approximation techniques, as described in chapter 3, is twofold. On the one hand, function approximation techniques can significantly reduce the dimensionality of a RL task and cause faster learning convergence. On the other hand, they allow the learning agent to generalize past observations to decide how to handle unobserved states efficiently. While dimensionality issues are not central since the state space defined in section 6.3 is quite dimensionality-efficient, the issue of generalization is crucial in traffic signal control problems. For instance, the states `051111` and `051112` are highly similar in practice since they only differ by one vehicle on the last lane of the intersection. If these two states are hence likely to have similar value estimates, they are yet they are entirely unrelated from a learning standpoint, which means that the RL algorithm will have to visit these two states separately in order to estimate their value. This section explores two distinct manners to exploit function approximation to improve the classical RL-TSC methods tested in the previous section. The first one uses a unique pattern that uses function approximation to bootstrap a Q-learning table to accelerate the learning process, while the second illustrates the use of deep neural networks to improve the standard Q-learning algorithm iteratively.

8.3.1 Q-learning Bootstrapping

As stated in the introduction of this section, the main limitation of classical reinforcement learning techniques is their inability to predict the value estimate of a state they have not yet visited. This issue is particularly problematic in TSC applications since intersections have to test a certain amount of low-value state-action combinations, such as switching the current green phase when the associated lanes around the intersection have a large number of vehicles waiting on them. These state-action combinations have to be tested for exploration purposes but can cause significant delays on the road network.

8.3.1.1 Quality Estimate Bootstrapping

A quite unorthodox first-approach solution to the problem of generalization for tabular RL methods is to provide quality estimates to the agent before learning even starts through function approximation (Matignon et al., 2006). In the case of Q-learning, this bootstrapping[1] method would pre-estimate each state-action entry of the Q-table with an estimation of its value. Such a solution requires estimating the impact of each traffic control action (i.e., extend or switch the current green phase) on each possible traffic state in terms of cumulated waiting time difference. This approach is hence on the margin of function approximation techniques: we do use a function to approximate quality estimates of the states of the environment, but we only do so before learning occurs, at which point we use a classical RL algorithm. To the best of our knowledge, using an approximation function to pre-populate the entries of a Q-table in a RL-TSC context had never been done before we published a paper on the matter (Tréca et al., 2020b).

s_t / a_t	0	1
s_1	0	0
s_2	0	0
...	...	...
s_m	0	0

s_t / a_t	0	1
s_1	$\hat{Q}(s_1, 0)$	$\hat{Q}(s_1, 1)$
s_2	$\hat{Q}(s_2, 0)$	$\hat{Q}(s_2, 1)$
...	...	...
$s_{\lvert\mathcal{A}\rvert}$	$\hat{Q}(s_\mathcal{A}, 0)$	$\hat{Q}(s_\mathcal{A}, 1)$

Table 8.3: Comparison of a regular (left) and bootstrapped (right) initial Q-table of an isolated intersection using Q-learning. The $\hat{Q}$ function represents a manual quality estimation of a state-action pair.

8.3.1.2 Estimation Function

Our bootstrapping method relies on estimating the quality of each state-action pair of the environment by computing the reward associated with applying each action to each traffic state of the state space. In practice, computing the reward of each state-action pair requires to estimate the difference in waiting time before and after an action was applied to a particular traffic state, which requires prior knowledge of the environment, most notably traffic demand around the intersection. For phase-based action types, the quality of an action is measured by estimating, for each lane, the number of vehicles that were present in the lane at the beginning of the action (denoted by o) and vehicles that entered the lane while the action is being applied (denoted by n). For both groups of vehicles, it is possible to estimate the number of vehicles that exited the lane (denoted by a $-$) and the vehicles that stayed on the lane (denoted by a $+$) and measure their respective impact on the overall waiting time. By supposing that each vehicle leaving the intersection lowers the cumulated waiting time by T_l, which is the average service time associated with lane l (and which is computed by modeling each traffic lane around the intersection as a `M/D/1/K` queue, as shown in our paper), we can estimate the quality of a specific state-action pair in a phase-based action space to be equal to:

[1] Bootstrapping refers here to the action of pre-filling the Q-table with estimated state-action quality values and is hence different for the concept of bootstrapping in the RL literature which refers to the technique which consists in using values estimates to update other value estimates in methods such as temporal-difference learning.

$$\hat{Q}(s,a) = \sum_{l} T_l \times (n^{-}_{l,a} + o^{-}_{l,a}) - a \times (o^{+}_{l,a} + n^{+}_{l,a}) \tag{8.6}$$

Note that Appendix B provides a complete description of this approximation algorithm in pseudocode form. This approximation method provides several advantages compared to the traditional Q-learning algorithm. First, while not providing exact values for the quality of each state-action pair, these estimates are still much closer to their actual quality function than default values of 0, which are typically used with Q-learning. Second, bootstrapping can occur without starting a traffic simulation and hence operate in a completely offline manner. This last point presents a significant advantage in computational resources since offline computation is significantly faster than online. Finally, if this approximation method has only been tested for phase-based actions, we believe that this method could easily be adapted to step-based actions by modifying the estimation model presented above.

8.3.1.3 Bootstrapping efficiency

These efficiency claims have been tested according to the experimental protocol described in section 7.3.1. The standard and bootstrapped Q-learning techniques are compared on 10 different simulation scenarios on an isolated intersection, each composed of 100 episodes of 10000 steps each. In each tested scenario, the overall vehicle arrival rate is fixed to $\lambda = 0.06$ vehicles per second, but the distribution of this arrival rate between the lanes of the intersection differs between scenarios.

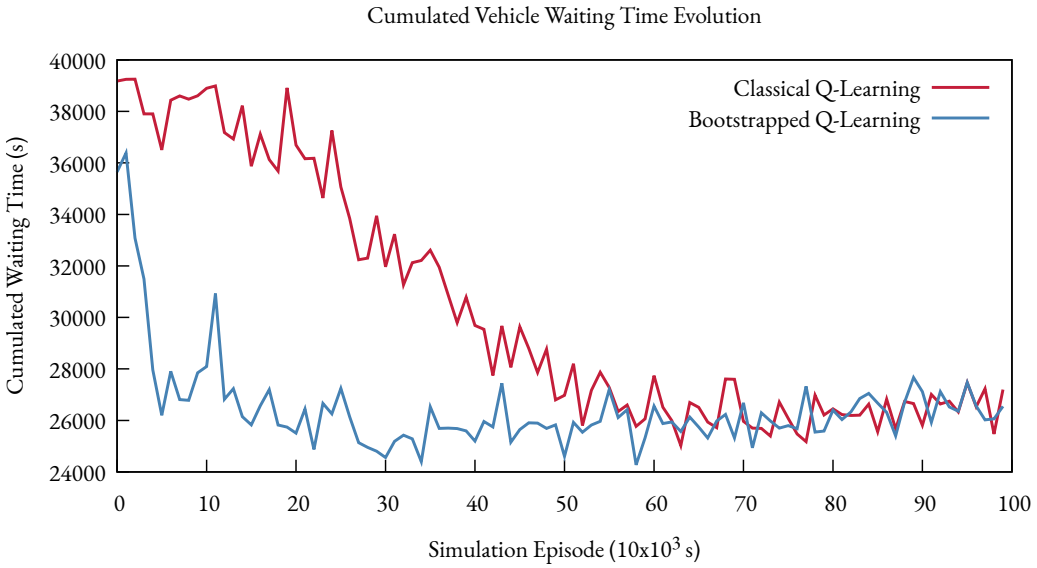

Figure 8.6: Comparative convergence process of a regular and bootstrapped Q-learning algorithm. Average out of 10 simulation scenarios.

As one can see by looking at the waiting time evolution per episode for these traffic scenarios on Figure 8.6, bootstrapping significantly improves the early performance of the Q-learning agent, which is strongly superior to the standard Q-learning algorithm for an average of 50 simulation episodes. The computation time needed to achieve similar results also strongly favors the approximation method: the entire offline bootstrapping process took around 16 seconds, whereas the online simulation training for the standard Q-learning method took around two and a half minutes. A last crucial point to mention is that while this approximation method does speed up the initial convergence process of a phase-based Q-learning agent, it does not improve its overall performance since it reverts to a classical Q-learning algorithm once online learning occurs. This statement can be verified on Figure 8.6 after episode 60 when the standard Q-learning method catches up with the approximation method. Hence, this method does increase the speed at which a Q-learning agent reaches acceptable performance, and it does not make the learning algorithm better at its task, which is a problem more likely to be solved by *bonafide* function approximation techniques, as presented in the next section.

8.3.2 Function Approximation for Q-Learning

The literature review of chapter 4 has shown the plethora of function approximation techniques that can be applied to traffic signal control. Most notably, we have seen in the literature review of chapter 4 the 3DQN method is believed to be one of the best available function approximation methods for a broad range of RL problems (Gregurić et al., 2020; Hessel et al., 2018). This fact, coupled with the conclusion of section 8.2.4 stating Q-learning-based methods are superior in isolated traffic situations, lead us to develop a deep Q-learning model for RL-TSC. This section hence iteratively builds a deep Q-learning method starting from a simple deep Q-network (DQN). We then define a number of RL-specific techniques that are gradually incorporated into this DQN method, explaining their purpose and limitations along the way.

8.3.2.1 Deep Q-learning

The first significant shift from a classical Q-learning method to a DQN method is the introduction of a neural network (see section 3.2.2.2) as a function approximator (Mnih et al., 2015). If the overall components of a neural network (see Figure 8.7) do not change across different models, its *architecture* can greatly vary depending on the problem at hand and the kind of input data it receives. Networks using image data as inputs are likely to feature convolutional layers, while models aiming at establishing temporal links between input vectors are likely to include recurrent features. Regardless of the architecture of the DRL model, the overall learning mechanisms remain the same: the current system state s is used as an input vector of the neural network, which outputs a vector of size $|\mathcal{A}|$ containing the estimated quality of each action available to the agent. Upon applying action a to the environment and observing its actual quality in the form of a reward r, the agent computes the difference between the estimated and the observed quality of the state-action pair using a *loss function* $\mathcal{L}$. This loss is then *backpropagated* to update the weights θ of the neural network using gradient descent methods such as Adam or RMSprop (Zou et al., 2019).

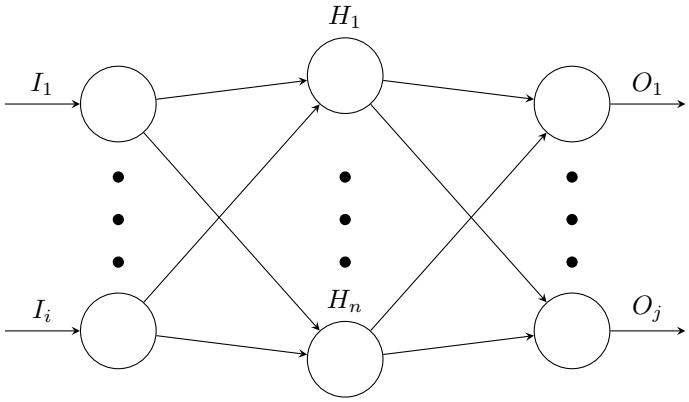

Figure 8.7: Simple illustration of a fully connected neural network. An input vector of dimension i goes through a single hidden layer of dimension n and results in an output vector of dimension j.

8.3.2.2 Experience Replay

One of the leading causes of convergence issues of function approximation techniques seen in section 3.2.2 is due to the fact that the system transitions observed by the learning agent are strongly correlated (i.e., the choice of a state-action couple (s, a) influences the choice of the following one, (s', a')) and that their distribution is non-stationary (i.e., the likelihood of observing a specific state-action pair directly depends on the current quality estimates and policy of the agent). A solution used to alleviate these issues is the use of *experience replay*: the agent maintains an observation buffer $\mathcal{D}$ that stores all system transitions (s, a, r, s') observed by the agent (either all observations or the last N ones) and samples the observations used for learning from this buffer. Experience replay breaks correlation between samples and allows for *batch normalization* (i.e., sampling multiple observations at once, increasing learning stability and convergence speed) when learning (Riedmiller, 2005). A common limitation of experience replay is that observations of the replay buffer $\mathcal{D}$ are sampled uniformly, meaning that the least commonly observed states (which might be the most important) have a low probability of being sampled. The use of *prioritized experience replay* alleviates this issue by sampling observations according to their TD-error value: the highest the TD error, the more likely the agent needs to learn from this specific observation, and the higher its selection probability is (Schaul et al., 2015).

8.3.2.3 Target Network

Using the same neural network to evaluate the value of a given state and for learning can cause it to oscillate due to frequent weight updating, a phenomenon known as the moving-target issue (Hessel et al., 2018). A common solution to these oscillation issues is to use two distinct networks for value estimation and learning: a target network θ^T with frozen weights is used to evaluate the loss while a value network θ^V is used for learning from these evaluations. The weights of θ^V are transferred to θ^T every K steps to limit the moving-target issue (Gao et al., 2017).

8.3.2.4 Double Q-learning

The use of two distinct networks for evaluation and learning also offers the possibility of further separating the role of both networks. Indeed, for a transition (s, a, r, s'), deep Q-learning estimates the quality $Q(s, a)$ using reward r and the value of the next system state s' by estimating the quality of best next state-action couple, $Q(s', a')$. In regular deep Q-learning models, both the best next action a' and the associated quality $Q(s', a')$ are estimated all at once using weights θ, while deep Q-learning using target networks will estimate the best next action and the associated quality $Q(s', a')$ using the value network θ^V. Double deep Q-learning (Hasselt, 2010) is an extension of simple DQN models aiming to avoid overestimations during the Q-learning target update by further separating the role of each network. The best next action is chosen using the current evaluation network θ^V, while the quality estimate of the corresponding state-action pair $Q(s', a')$ is computed using the target network θ^T, increasing estimation robustness and decreasing the likeliness of model over-estimation (Gregurić et al., 2020).

8.3.2.5 Dueling Networks

Dueling networks (Wang et al., 2016) are another extension of DQNs which use two distinct estimators within the neural network to evaluate the value of a state $V(s)$ and quality of a state-action pair $Q(s, a)$ (see section 3.1) separately (Gregurić et al., 2020). More specifically, the second estimator computes the *advantage* of each state-action pair, $A(s, v) = Q(s, a) - V(s)$, which represents the relative value of an action compared to others for a given state. Separating the evaluation of states allows learning which states are intrinsically valuable for the agent regardless of which action is being applied to it, which allows identifying states in which actions do not influence the environment in any meaningful way (e.g., a traffic state in which no vehicles are present), further speeding up the learning process (Liang et al., 2019).

8.3.2.6 DDQN and 3DQN Algorithms

The two reinforcement learning techniques presented in this section, double Q-learning and dueling networks, can be used to form various deep RL methods. A deep Q-network used with double Q-learning forms a double deep Q-network (DDQN), while using a dueling network on a deep Q-network forms a Dueling deep Q-networks (2DQN). Merging both techniques results in the state-of-the-art 3DQN algorithm used in most recent RL-TSC models (Gregurić et al., 2020), which is used in a number of recent RL-TSC models (Calvo and Dusparic, 2018; Wang et al., 2019). Given the recent development of the dueling deep Q network technique (2016), some models feature only double deep Q-learning, or DDQN, models (Gao et al., 2017; Genders and Razavi, 2016; Van der Pol, 2016) while showcasing excellent performance levels. A pseudocode illustration of both algorithms (depending on the underlying neural network architecture) is presented in algorithm 6.

Algorithm 6: Pseudocode illustration of a DDQN/3DQN algorithm.

Initialize θ^T and θ^V with random weights ;
Initialize empty buffers $\mathcal{D}$ and $\mathcal{P}$;
for *each step* t **do**
  Observe s, a, r, s';
  $\mathcal{D} \leftarrow \mathcal{D} + (s, a, r, s')$;
  $\mathcal{P} \leftarrow \mathcal{P} + (|Q(s, a) - r + \gamma \max_{a'} Q(s', a')| + 0.01)$;

  if $|\mathcal{D}| \geq B$ **then**
    Sample $(s_B, a_B, r_B, s'_B) \sim \mathcal{P}$;
    $a^*_B \leftarrow \max_{a'_B} Q(s'_B, a'_B; \theta^V)$;
    $\mathcal{L} = (Q(s_B, a_B; \theta^V) - r_B + \gamma Q(s'_B, a^*_B; \theta^T))^2$;
    Update weights θ^V using $\mathcal{L}$;

  if $X \sim \mathcal{U}(0, 1) < \varepsilon$ **then**
    $a' \sim \mathcal{U}(\mathcal{A})$;
  else
    $a' \leftarrow \max_a Q(s', a; \theta^V)$;

  if $t \mid K$ **then**
    $\theta^T \leftarrow \theta^V$;

8.3.3 Applying Function Approximation to Traffic Signal Control

After presenting the various ways function approximation techniques can be applied and refined on learning problems, our final task is to gauge the effectiveness of these techniques on traffic signal control optimization. This section studies the effect of the function approximation techniques presented above, one by one, on an isolated intersection in order to produce the most efficient isolated reinforcement learning algorithm for traffic signal control.

8.3.3.1 Network Architecture

The first step regarding the construction of an efficient RL-TSC method using function approximation is the choice of its architecture, which is the organization of the hidden layers of its neural network. An important point to underline is that, besides being usually problem-dependent, the architecture of a neural network also largely depends on the type of inputs this network is expected to receive. This point is crucial since papers applying deep Q-learning methods to traffic signal control usually provide complex state information as inputs for their neural networks, usually in image form, which requires the associated network to treat these images using multiple convolutional layers (Calvo and Dusparic, 2018; Van der Pol, 2016; Wang et al., 2019). Our traffic model uses a discrete and relatively compact state definition since they can be used by both classical and deep reinforcement learning methods while ensuring similar performance levels (see section 6.3), which implies that our neural network architecture is likely to be different since convolutional neural networks are not needed in our case.

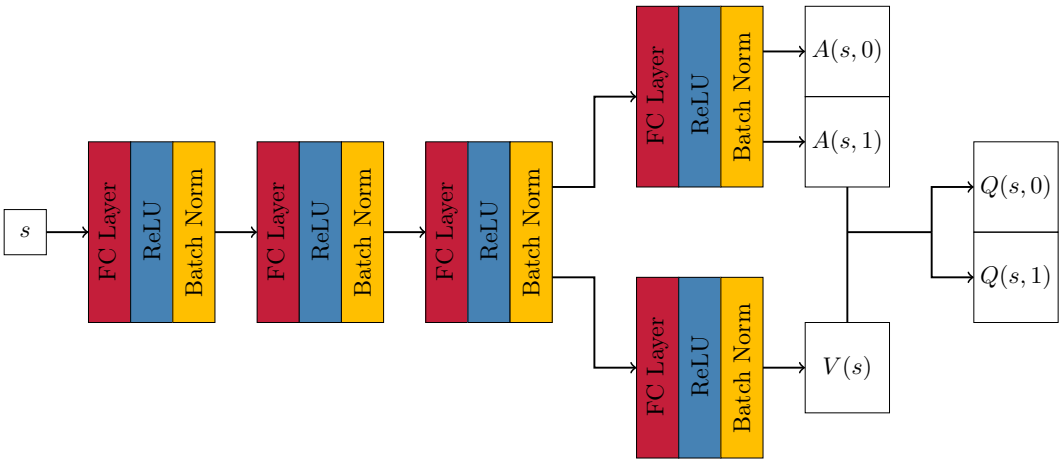

Figure 8.8: Architecture of the 3DQN network.

The neural network architecture used for our experiments, presented in Figure 8.8, features four fully connected layers of 128 neurons each. Each layer is associated with a ReLU rectifier, which is a prevalent activation function in the deep RL literature (Ramachandran et al., 2017), and a batch normalization layer which accelerates learning by normalizing inputs between layers (Ioffe and Szegedy, 2015). Not that we do not include dropout, a mechanism used to randomly drop weights between layers, to our neural network since they are not necessary when applying batch normalization (Li et al., 2019). When dueling networks are implemented (as in Figure 8.8), two distinct layers are used to compute the advantage and state value separately before combining them; if this feature is not implemented, a fourth regular layer directly computing the state-action quality is used instead.

Parameter	Value
Optimizer	Adam (Kingma and Ba, 2014)
Learning rate α	0.0001
Replay buffer $\mathcal{D}$ size	10000 observations
Minibatch size B	100 observations
Target network update interval K	1000 steps

Table 8.4: Deep reinforcement learning-specific simulation hyper-parameters used for function approximation RL method comparison.

8.3.3.2 Deep Q-learning and Experience Replay

This section measures the impact of function approximation alone (i.e., using a deep neural network instead of a tabular data structure) and of function approximation with experience replay on the performance of an isolated intersection with an overall arrival rate of $\lambda = 0.04$ for 250

episodes of 1000 simulation steps each. This experiment once again uses the experimental protocol defined in section 7.3.1, and uses the hyperparameters defined in Table 8.2 and Table 8.4.

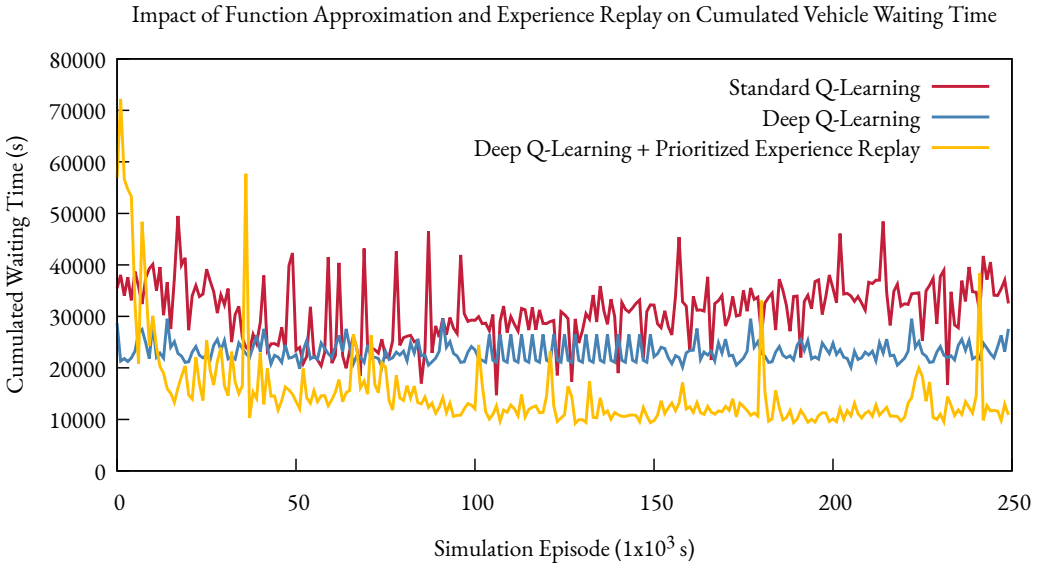

Figure 8.9: Impact of function approximation and experience replay on traffic routing performances. Tested on a single intersection with an arrival rate of $\lambda = 0.4$. Batch normalization is used on deep Q-learning with experience replay.

Results of this first experiment on Figure 8.9 underline the importance of both function approximation and experience replay. Due to the absence of generalization from past experiences, the standard Q-learning method (in red) very quickly converges to acceptable performance levels but reaches a performance plateau due to its inability to fully exploit its interactions with the environment, similar to what has been observed in section 8.2.4. The use of function approximation with a basic deep Q-learning technique (in blue) allows for generalization and hence better performance and increased stability. However, the need for both function approximation and experience replay seems to be crucial in RL-TSC applications. Indeed, the tested method using both techniques (in yellow) greatly outperforms the others. Instead of learning from one observation at a time, learning from batches of $B = 100$ observations allows the agent to revisit the same traffic transition multiple times and quickly converge to improved performance levels. Hence, both function approximation and experience replay is vital in building an efficient RL-TSC method.

8.3.3.3 Double Q-learning and Dueling Networks

After validating the need for deep reinforcement learning coupled with experience replay for efficient traffic signal control, we turn our attention to the additional techniques presented earlier, such as using target networks, double Q-learning, and dueling networks. Since we are confident in the robustness of the DQN methods we are testing, we measure their performance on an isolated intersection with a near-saturation flow rate of $\lambda = 0.6$, once again randomly distributed

across the lanes of the intersection. The following combination of techniques is incrementally tested: DQN with experience replay and target network (DQN), DQN with double Q-learning (DDQN), DQN with dueling networks (2DQN) and, finally, DQN with both double Q-learning and dueling networks (3DQN). While it would seem logical to predict that the most advanced technique should yield the highest performance metrics, using a learning model that is too complex for a given learning task will at best complexify the model for no valid reason (and hence increase training time) and at worse decrease the learning performance the model due to overfitting (Vapnik and Izmailov, 2019).

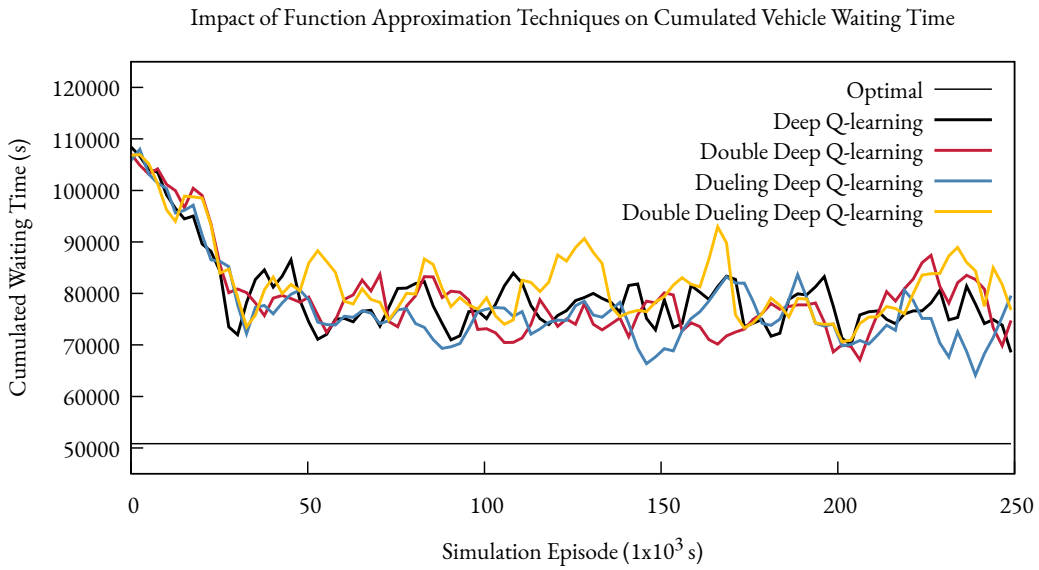

Figure 8.10: Impact of function approximation techniques on traffic routing performances. Average values plotted from 10 simulation scenarios on a single intersection with an arrival rate of $\lambda = 0.6$. Batch normalization, target networks and prioritized experience replay is used on all tested methods. Results are smoothed for readability.

Simulation results of Figure 8.10 show that all the tested methods provide somewhat similar cumulated waiting time levels. Detailed average cumulated values over simulation episodes, shown on Table 8.5, gives, however, insight into the comparative performances of these function approximation methods. The best method, in our experimental setting, turns out to be dueling deep Q-learning (2DQN). Interestingly, double Q-learning, which is often used in the RL-TSC literature, seems to have no effect when used on its own compared to a standard deep Q-learning method and seems to slightly degrade performance when used in conjunction with a dueling network (see the yellow plot on Figure 8.10). While no clear-cut answer can be given as to why such a phenomenon is present since double Q-learning has been developed to limit over-estimation issues in deep Q-learning (Van Hasselt et al., 2016), the fact that the traffic signal control problem at hand is relatively much more straightforward than the Atari games on which the original algorithm was tested or complex traffic state representation using images could explain why such

an addition would be unnecessary in our model and slightly degrade performance. Conversely, the relatively helpful addition of a dueling network in the RL-TSC model seems to indicate that increased discrimination between low and high-value states is beneficial to TSC-related learning.

Algorithm	Episodes 0-250	Episodes 150-250
Deep Q-learning	79381	76823
Double Deep Q-learning	79127	76377
Dueling Deep Q-learning	77257	74469
Dueling Double Deep Q-learning	82188	79659

Table 8.5: Average waiting time per simulation episode and deep reinforcement learning algorithm type.

Based on these findings, we decided to use dueling deep Q-learning (which we abbreviate to 2DQN as to not confuse it with DDQN, which stands for double deep Q-learning) as the deep RL method of choice for traffic signal control and will use this method as a benchmark for tackling agent coordination in the next chapter.

This chapter has analyzed a wide array of RL-TSC methods on isolated intersections. As stated in section 8.2.4.1, we first voluntarily restrict the road network scope to a single intersection in order to analyze the effect of RL algorithms and policies with as little noise as possible before extending this scope to multi-intersection networks in chapter 9.

This chapter first presented deterministic methods in order to use them as reference benchmarks when studying RL-TSC algorithms. First, we described the classical Webster method, which assigns green phases according to the demand profile of each line around an intersection. We then presented a novel near-optimal method using `SUMO`'s ability to save and load simulation states. Both of these methods provide average and optimal benchmarks for isolated RL-TSC performance, respectively.

We then discussed the respective merits and shortcomings of classical (i.e., not using function approximation) RL methods. We compared algorithms for three main classes of RL methods: Q-learning for value-based, LRP for policy iteration, and a hybrid of the two previous algorithms for actor-critic methods. Experiments have shown that Q-learning provides the best performance for our experimental settings regardless of its policy, making it our preferred class of method for the remainder of our research.

Finally, we discussed the use of function approximation techniques for isolated RL-TSC methods. Experiment results have first shown that both function approximation and prioritized experiment replay were essential for proper learning convergence. We then presented a wide array of function approximation techniques, such as target networks, double Q-learning, and dueling networks, in order to identify which combination of these techniques could yield optimal performance levels. The results of this second experiment have shown that combining deep Q-learning, prioritized experience replay, target, and dueling networks in a method known as 2DQN reached the best possible performance in our experimental setting.

9 Coordinated Traffic Signal Control Methods

The previous chapter's traffic optimization study led us to identify dueling deep Q-network (2DQN) as the algorithm of choice for traffic optimization on isolated intersections. As underlined in chapter 4, however, traffic optimization over multiple intersections is essential for proper traffic signal control as single-intersection networks are seldom encountered in real-life traffic scenarios. As the extension of the traffic model to a multi-agent setting is hence necessary, this shift raises the central question of the interactions between the multiple agents of this new model. Since MARL models can both choose to ignore (in the form of independent learning) or model agent interactions (through coordination methods), the first objective of this chapter is to measure whether agent coordination is needed in the context of traffic signal control by comparing independent and coordinated learning methods. If coordination is shown to be beneficial to traffic signal control performance, our second goal is then to explore which forms of coordination are most beneficial for traffic optimization. Since multiple forms of agent coordination–such as indirect and direct coordination in the RL literature (Panait and Luke, 2005) or green waves in traffic engineering,–are applicable in MARL models, these methods will be tested one by one in a multi-agent setting.

9.1 Independent Learning

The simplest form of interaction to consider between agents in a MARL model is independent learning, in which agents do not acknowledge each other while learning in the same environment. While simple in nature, independent learning models have several benefits, such as high scalability (since agents learn independently from each other and can hence be freely added or removed from the road network) and excellent performance for relatively low complexity, as seen in the simulation results of the previous chapter. Preventing the modeling of agent-to-agent interactions does not, however, allow for agents to share observations or learned policies, which proves helpful (and sometimes crucial depending on the learning task) in accelerating their learning process (Tan, 1993). The use of independent learning for the study of coordination methods hence plays a central role in the evaluation of coordinated RL-TSC methods since it is not only considered as a coordination option to route traffic but also as a benchmark used to generally measure the added benefit of agent coordination in the specific case of RL-TSC tasks.

9.1.1 Optimal Method in the MARL Case

As explained in chapter 3, the shift from a SARL to MARL model has a number of consequences on the RL-TSC model at hand. The first consequence of this switch is that the approximation method defined in section 8.1.2 is likely to not guarantee optimality anymore. Indeed, in the isolated case, the intersection can easily compute an approximation of the optimal strategy since it is the only agent affecting the network. However, in the MARL case, multiple intersections will aim at computing an optimal strategy step by step without explicitly knowing the strategy of its neighbors. In other words, since agents computing their optimal strategy cannot guess which strategy their neighbors are going to apply, the resulting strategy has no guarantee to be optimal or near-optimal. Note that it would be possible to design a multi-agent version of the optimal strategy approximation algorithm, which computes an optimal joint strategy at the cost of much higher complexity. However, we emit the hypothesis that neighboring intersections' influence on the optimal strategy of an intersection has little influence on the overall performance levels of the algorithm in practice. This hypothesis is based on an experiment shown in section 9.3.2.3, which shows that vehicles coming from neighboring intersections have little effect on the performance of an intersection in the short term. It should, however, be noted that while we apply the same near-optimal strategy search algorithm, its computational costs are much higher since each intersection has to compute its optimal strategy approximation algorithm separately.

9.1.2 Independent Learning Performance

The primary consequence of extending the learning model to a multi-agent setting is the introduction of non-stationarity and higher learning instability due to concurrent agent learning. Since independent learning methods do not provide explicit mechanisms to deal with these issues, the introduction of multiple learning agents on a road network could influence the hierarchy of function approximation methods defined in section 8.3.3, which concluded in the superiority of the 2DQN controllers for traffic signal control in an isolated intersection setting. This section hence presents a control experiment checking whether the results obtained on an isolated intersection still hold true in a multi-agent setting. This experiment compares the 2DQN method with the

3DQN method, which has also been used in a multi-agent MARL scenario (Calvo and Dusparic, 2018) and a standard DQN algorithm in a simple multi-intersection network in order to verify if their performance hierarchy changes with the introduction of multiple agents.

9.1.2.1 Convergence of Independent Methods

The first manner in which these independent methods are compared is, as done previously, by measuring their average convergence trajectory over different traffic scenarios. This protocol is explained in detail in section 7.3.1. We test these methods on a two-by-two grid network composed of four intersections, each implementing an independent version of their RL-TSC algorithm. Traffic demand generation is done according to the protocol detailed in section 7.2.2. The arrival rate is fixed to $\lambda = 0.015$ on average per entry-exit edge pair on the network. The learning hyperparameters used for this experiment are listed in Table 8.2 and Table 8.4.

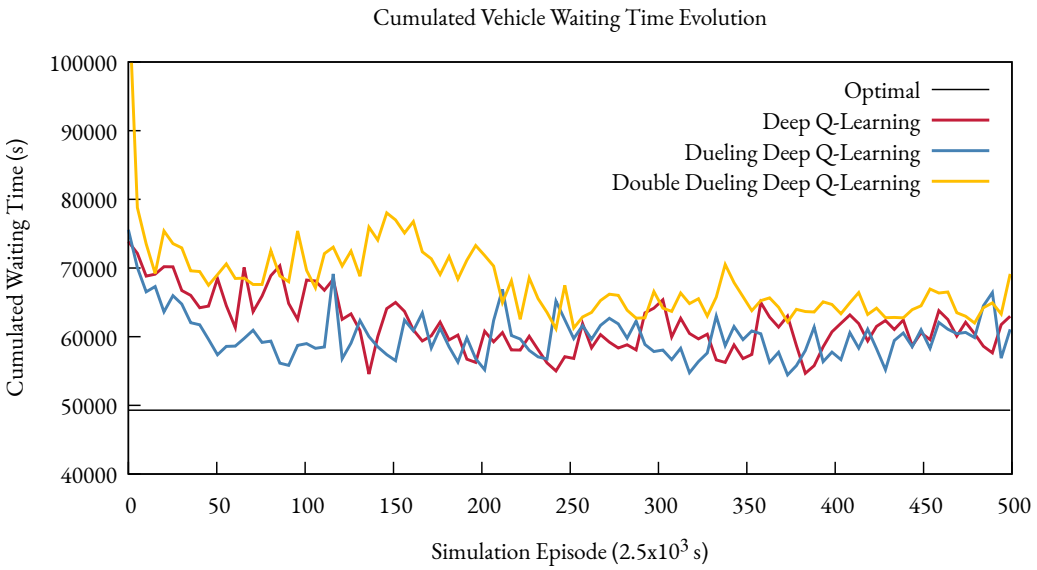

Figure 9.1: Comparison of multi-agent independent function approximation methods. Average results over 5 simulation scenarios.

Results displayed on Figure 9.1 show the cumulated waiting time evolution for these methods averaged over five distinct traffic scenarios, for 500 episodes of 1000 steps each. First, these results confirm that the relative efficiency of these methods does not seem to be affected by the shift to a MARL model, as they are similar to the single intersection results of Figure 8.10. Indeed, the 2DQN algorithm still yields the best overall performance in terms of cumulated waiting time, while the 3DQN method still displays a relatively high learning instability as observed in the isolated intersection scenario of section 8.3.2. Overall, these results show that while DQN and 2DQN are each relatively close to the (unattainable) optimal strategy performance levels, the 2DQN algorithm is slightly more efficient on average, making it our preferred method for inde-

pendent multi-agent learning. In its independent multi-agent version, this chosen algorithm will be referred to as independent dueling deep Q-network (I2DQN).

9.1.2.2 Performance of Independent Methods

While the analysis of the convergence process of the various methods in the previous paragraph already underlines essential information regarding their efficiency, it does not reveal their entire traffic routing capabilities. Indeed, as mentioned in section 7.3.1, the convergence analysis of RL-TSC methods do not entirely measure their abilities for multiple reasons. First, these methods are still in the learning phase when being compared and use associated mechanisms such as state-space exploration (i.e., by using a random action selection policy parameter $\varepsilon > 0$). Second, since convergence performance measurements are computed over entire simulation episodes, there is little information about the performance variability of these methods within a simulation episode or across different simulation scenarios.

Since performance variability of RL-TSC methods is a characteristic we wish to observe, we compare the independent function approximation techniques of the previous section according to the performance protocol described in section 7.3.2. Multi-agent performance evaluations measure the cumulated waiting time evolution *within a single simulation episode*. More specifically, we measure the minimal and maximal cumulated waiting time of the DQN, 2DQN and 3DQN methods over 20 distinct traffic scenarios of 1000 steps each. These simulation results are shown on Figure 9.2. The experimental parameters are similar to those of the previous section.

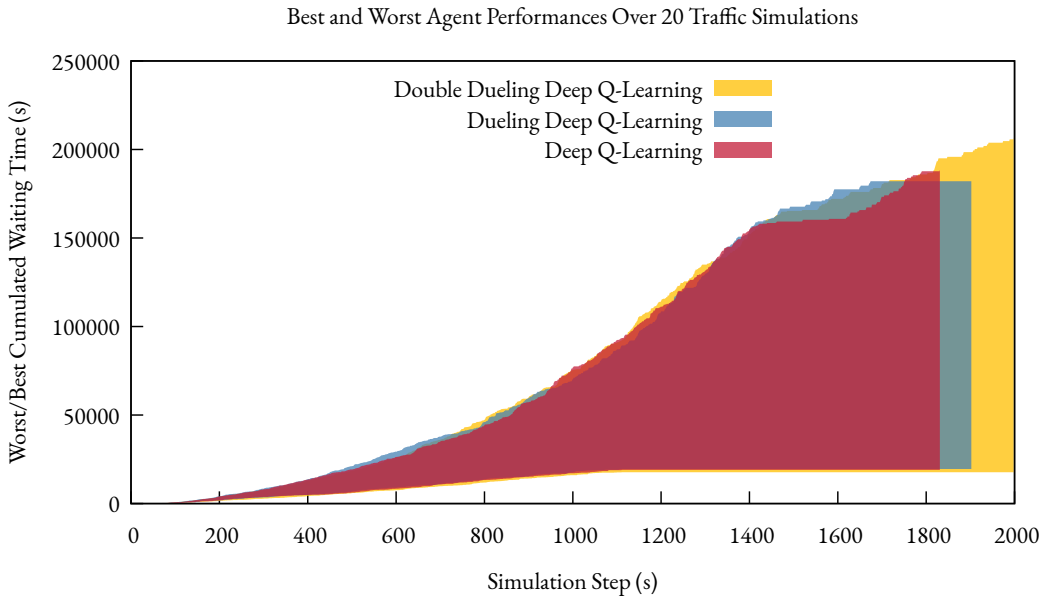

Figure 9.2: Performance comparison of the DQN, 2DQN and 3DQN algorithms over 20 traffic scenarios.

Performance results confirm our intuitions regarding the independent methods we tested. All methods display a large variability regarding overall cumulated waiting times, which indicates

a substantial variety of the simulation scenarios we tested (cumulated waiting times vary from around 20 000 to 190 000 depending on the traffic scenario). The mean cumulated waiting times for these methods are relatively close to each other (77 718 for DQN, 72 993 for 2DQN and 73 764 for 3DQN). While displaying the most extensive variability of all, by simultaneously yielding the best and worst overall simulation results depending on the tested scenario, the 3DQN method is inferior to the 2DQN method from a mean performance standpoint, which is considered essential since reduced performance variability should ensure a more stable convergence of the learning process on a broader range of traffic scenarios. Consequently, our choice to use the 2DQN method we formulated in chapter 8 is maintained in multi-agent settings.

9.2 Green Wave Coordination

Explaining *why* communication and coordination between intersections should be used is a surprisingly complex issue that arises when reading the RL-TSC literature. Indeed, many literature papers proposing modern and coordinated traffic signal control methods claim that interactions between intersections of a road network are beneficial in optimizing traffic, usually proving this claim with plots showing the superior performances of these multi-agent methods. However, these papers seldom explain *how* coordination makes optimizing traffic easier[1]. A satisfying answer to this question can nonetheless be found in the traffic engineering literature, which has the benefit of directly describing coordination methods that have been applied in real-world contexts for decades. Consequently, the first coordination mechanism we study are *green wave* coordination techniques, which are extremely common in real-world traffic applications, and have the advantage of being quickly developed in traffic simulation settings (as opposed to many proprietary traffic routing methods whose source code is not accessible).

9.2.1 Green Wave Coordination Mechanisms

The main goal of green wave coordination is to allow for continuous vehicle movement along an arterial or major street by properly offsetting green phases on their traffic controllers. When correctly executed, green waves decrease the number of stops and delays along these arterials. A major point to note is that green wave coordination is not always desirable. Indeed, the US traffic signal timing manual states that intersections must be close to one another[2] and share the same cycle time, and that significant traffic must occur between them for coordination to be beneficial (Koonce and Rodegerdts, 2008). Green waves are designed around three key parameters: cycle time, offset and split time. First, the cycle time of all intersections within the green wave must be identical for synchronization purposes and is hence computed to best fit the traffic demand of all these intersections, for instance, using Webster's formula (see section 8.1.1). The offsets between intersections represent the delay with which they will successively apply the same green phase along the arterial, hence creating a green wave. Offsets are equal to the ratio of the authorized speed divided by the length of the streets of the arterial, which approximates the time it takes

[1] A notable exception is featured in the paper of Wei et al., which aims to equalize queue pressure across intersections, which has been proved to result in optimal intersection throughput (Wei et al., 2019b)

[2] the Manual on Uniform Traffic Control Devices recommends a maximum length of 800m between two intersections for coordination

a vehicle to travel from each intersection of the arterial to the next. Finally, split time designates the organization of the remaining phases of the intersections within their respective signal cycles, with the constraint that the arterial green phase repeats at fixed time intervals to preserve the green phase offsets. The traffic signal settings giving way to a green wave along a specific path are typically represented using a *time-space* diagram, as shown on Figure 9.3.

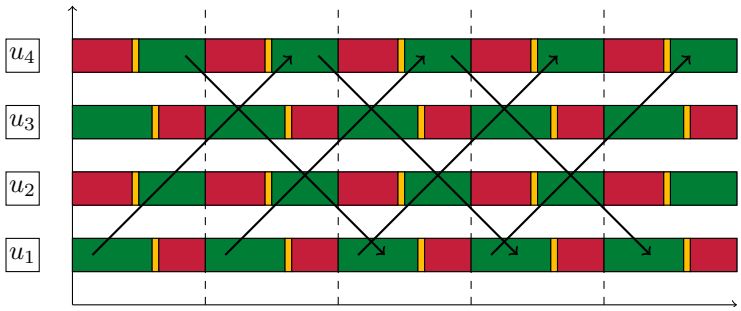

Figure 9.3: Example time-space diagram on a four intersection arterial. The x-axis represents time and the y axis distance. Intersections of the arterial are numbered from u_1 to u_4. Vehicle movements across the arterial are represented by black arrows. The signal cycles of the four intersections, represented across the time axis next to its intersection, are computed so that green waves can occur in both directions of the arterial.

9.2.2 Green Wave Methods

This section proposes two distinct traffic control methods relying on green wave coordination to test their efficiency. As noted previously, green wave coordination requires additional preprocessing compared to other forms of traffic light coordination since key arterials have to be identified on the network. Their associated cycle time and offsets have to be computed beforehand.

9.2.2.1 Fixed Green Wave Method

The first green wave coordination method is a fixed method akin to most real-world implementations. Once one or more arterials are identified on the network, and their associated traffic demand obtained, we compute the optimal cycle time of each intersection using Webster's formula (see section 8.1.1) and use its average value per arterial as the arterial's cycle time. We then manually compute offsets and apply them along the arterials. The resulting method is a fixed green wave traffic signal method that is not adaptive but provides a good performance baseline.

9.2.2.2 Adaptive Green Wave Method

The second green wave coordination method is an adaptive extension of the first, which relies on the 2DQN method instead of Webster's formula to compute the splits within the signal cycle. Since green wave coordination relies on fixed-time signal cycles to ensure coordination along the arterial, the adaptive green wave method can only compute the relative length of each phase within this signal cycle. The action space of this method is hence necessarily phase-based, as the agent

has to choose the entire length of the major arterial green phase in advance due to signal cycle constraints. While this method suffers from the limitations associated with using a phase-based action space (see section 6.4), it should, however, benefit from built-in coordination mechanisms inherent to this method which do not have to be learned, and of the efficiency associated with 2DQN methods.

Algorithm 7: Illustration of the coordinated green wave algorithm. This algorithm is implemented on intersections featuring two green phases but can be adapted for a larger phase amount. Variable C represents the total cycle time, which is equal on all intersections along the arterial. Learning, sampling, and target network update operations of 2DQN are omitted for brevity.

Initialize g_v to $\emptyset$ for each intersection v of the network;
for *each step* t **do**
  for *each intersection* v **do**
    Observe s_v, a_v, r_v, s'_v;
    if $g_v = \emptyset$ **then**
      if $X \sim \mathcal{U}(0,1) < \varepsilon$ **then**
        $a'_v \sim \mathcal{U}(\mathcal{A})$;
      else
        $a'_v \leftarrow \max_a Q(s'_v, a_v; \theta_v^V)$;
      $g_v \leftarrow a'_v$;
    else
      $a'_v \leftarrow C - g_v - 2d_{min}$;
      $g_v \leftarrow \emptyset$;

9.2.3 Green Wave Performance

This section evaluates the performance of the two green wave methods described in the previous section, especially regarding the usefulness of green wave coordination. In order to do so, we compare the fixed green wave method to some of its non-coordinated counterparts, such as fixed signal cycles or signal cycles computed using Webster's formula. To isolate the coordinated feature of the adaptive green wave method, we pitch it against the I2DQN method of section 9.1, which represents our best RL-TSC method so far. This comparison is made according to three distinct axes: convergence analysis, performance analysis under normal traffic conditions, and performance under saturated traffic conditions.

9.2.3.1 Green Wave Convergence

We first test the green wave coordination methods and I2DQN controllers by looking at their convergence according to the protocol defined in section 7.3.1. These methods are tested on a simple 4-intersection network which features a main arterial, as pictured on Figure 9.4. Traffic data is generated specifically to create higher traffic demand along the main arterial of the network: each edge pair at the edge of the network has a base flow rate value of $\lambda = 0.06$ vehicles per step.

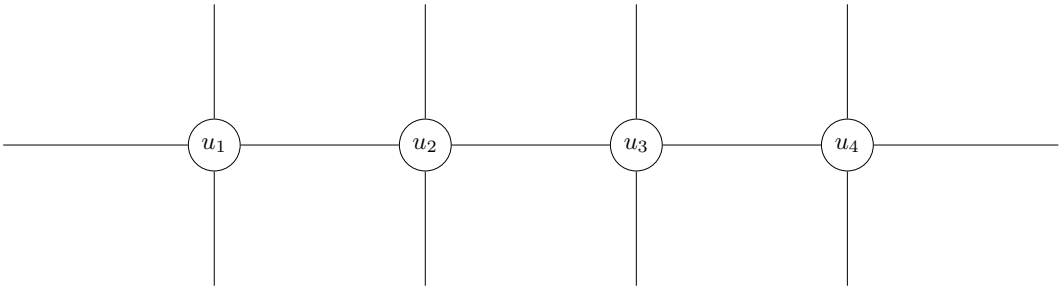

Figure 9.4: 4 interesection network use to emulate green waves along an arterial. All intersections share the same offset time of 16 steps and the same cycle time of $C = 30$.

If the starting edge of the pair is located on the arterial (i.e. horizontally next to intersections u_1 and u_4 on Figure 9.4), this flow rate is increased by 0.01. Similarly, if the ending edge is located on the arterial, the flow rate parameter is also increased by 0.01. Once the flow rate of each edge pair is computed, traffic is generated according to the protocol of section 7.2.2.

We first compare the convergence trajectories of the 2DQN and coordinated green wave methods over 500 episode runs of 1000 steps each, averaged over five distinct traffic scenarios. Figure 9.5 showcases the cumulated waiting time evolution per episode of each method. Multiple key points can be deduced from these results.

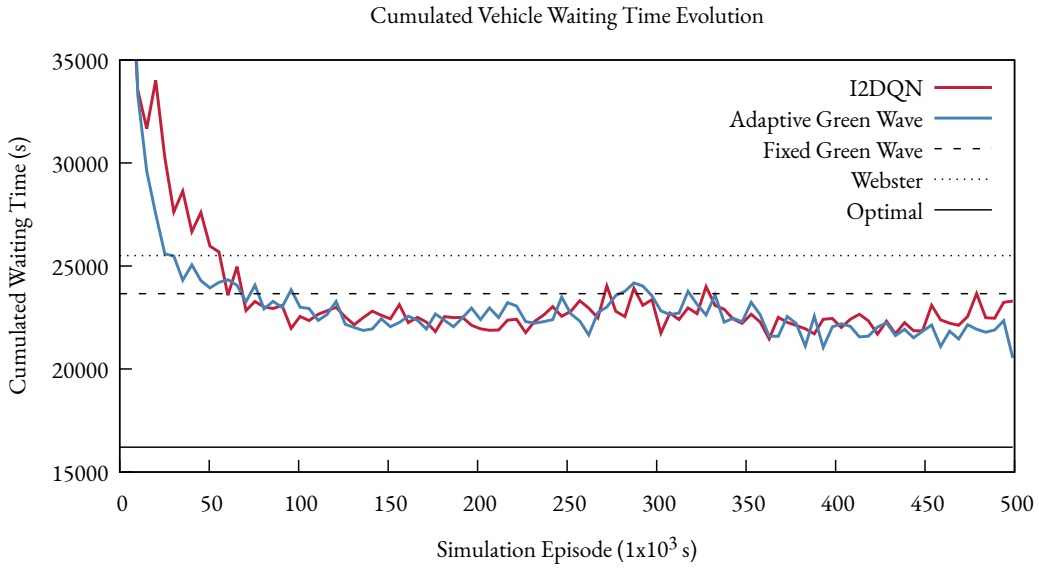

Figure 9.5: Performance evolution of green-wave and DQN-based TSC methods on the line network. Fixed methods are represented as dotted lines in the following order (from worst to best): simple fixed method, Webster fixed method, green wave fixed method and optimal method.

First, a clear hierarchy can be established regarding fixed, non-adaptive methods: the green wave Webster algorithm outperforms the regular Webster algorithm quite significantly, which highlights the usefulness of green wave coordination for fixed methods. This comparison obviously does not include the optimal method, which is used as a lower performance bound that cannot be beaten by RL-TSC methods.

Regarding the two adaptive methods, the first point to notice is that both methods converge to similar performance levels, eventually outperforming the fixed green wave coordination method. We do, however, notice that the convergence process is faster in the green wave coordination case, which is likely because the green coordination mechanism of this method is not learned but forced on the intersections, giving this method an advantage in the early simulation episodes. This observation also proves our earlier hypothesis stating that these built-in coordination mechanisms would influence traffic performance more than the use of phase-based actions, which are less optimal than step-based ones (Tréca et al., 2020a). Table 9.1 illustrates the evolution of the convergence process of both methods and shows the slight superiority of the green wave coordination method throughout simulation iterations.

Average Waiting Time	0-500	0-250	250-500	400-500	450-500
Deep Q-learning	23619	24690	22548	22391	22369
Adaptive Green Wave	23440	24493	22386	21826	21796

Table 9.1: Average waiting time per simulation episode according to episode intervals.

9.2.3.2 Performance in Normal Traffic Conditions

As stated in the previous section, studying the convergence process of a RL-TSC method is not always sufficient to study its effectiveness. Indeed, performance evaluations allow comparing multiple RL-TSC methods post-training while eliminating sub-optimal action choices due to exploration. Such a comparison is even more necessary when comparing I2DQN and the green wave coordination method since both methods do not choose actions at the same rate (since the I2DQN method is step-based and the green wave method phase-based), which might cause an additional bias since the former chooses actions much more frequently than the latter, and is hence more likely to choose random actions often. We hence measure the performance of the I2DQN and coordinated green wave method using the performance protocol defined in section 7.3.2. This experiment is conducted over 20 distinct traffic scenarios using a non-saturating base flow rate of $\lambda = 0.06$.

The performance outputs of the I2DQN and adaptive green wave methods of Figure 9.6 confirm the superiority of the green wave method. Indeed, one can observe that traffic scenarios using the adaptive green wave method (in blue in the performance plot) are shorter, as symbolized by the shorter size of the plot along the x-axis, indicating that vehicles generated up to simulation step 1000 are reaching their destination faster. Furthermore, the adaptive green wave method shows better average performance than the I2DQN algorithm, as one can see with the relative position of both curves while suffering from less performance variability, as shown by its smaller surface area on Figure 9.6. It is, however, essential to bear in mind that the superior performances of the

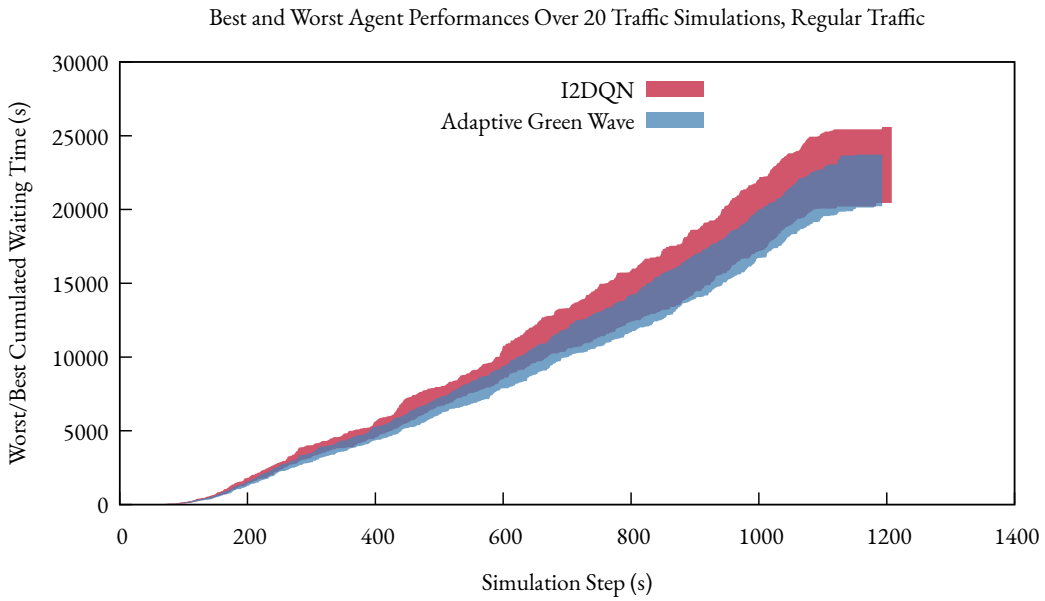

Figure 9.6: Performance spectrum comparison of Deep Q-learning and Green Wave Coordination methods over 20 traffic scenarios. Base traffic arrival rate fixed to $\lambda = 0.06$.

green wave coordination method were obtained in a traffic scenario featuring optimal parameters for the use of green wave coordination. Indeed, our experiment features a main arterial composed of multiple close intersections with a relatively normal traffic demand along the arterial, which encourages green waves along the arterial.

9.2.3.3 Performance in Saturated Traffic Conditions

Given their strongly different nature, the last point to consider when comparing the I2DQN and green wave coordination methods is their resilience to saturated traffic conditions. Indeed, while the I2DQN method should learn to adapt regardless of the traffic conditions due to its adaptive and independent nature, the built-in coordination mechanism of the green wave method is not guaranteed to function if the traffic is saturated or over-saturated as bandwidth solutions often result in poor performances in these situations (Koonce and Rodegerdts, 2008). In order to evaluate whether these limitations affect the adaptive green wave controller, we run a second performance evaluation in near-saturated traffic conditions with a base arrival rate of $\lambda = 0.08$ vehicles per step. The rest of the simulation parameters remain similar to those of the previous section. Results of this second evaluation are on Figure 9.7.

Performance evaluations in a saturated road network of Figure 9.7 confirm our original hypothesis regarding the poor robustness of the green wave coordination method in near-saturated traffic conditions. Both the I2DQN and green wave methods suffer from worse performance levels which are mechanically due to higher traffic demand. However, the increase in cumulated waiting time is much higher in the green wave case (250%) than in the I2DQN case (142%). Furthermore,

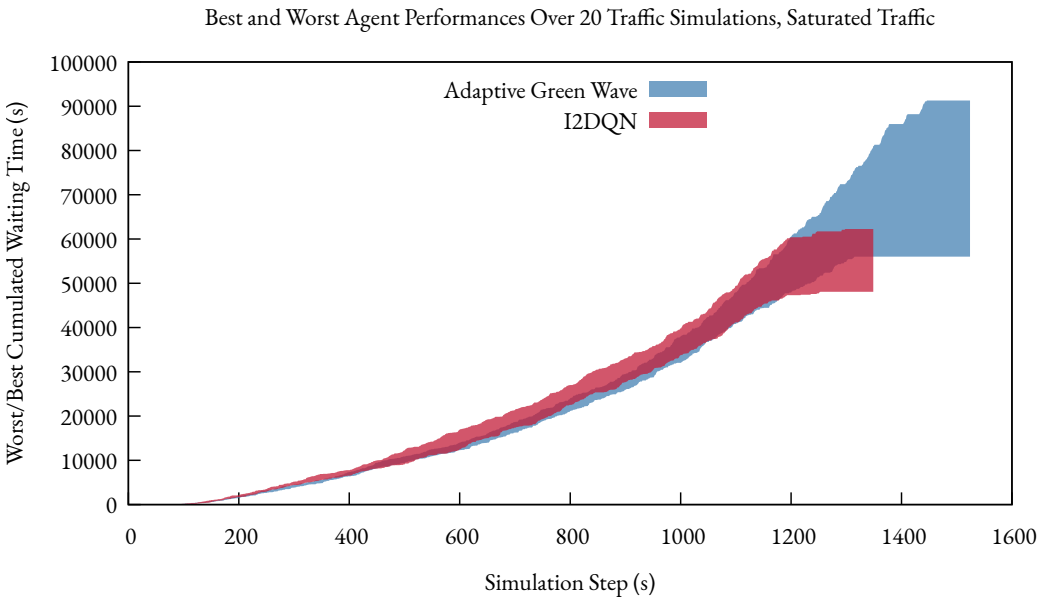

Figure 9.7: Performance spectrum comparison of the I2DQN and adaptive green wave methods over 20 traffic scenarios. Base traffic arrival rate fixed to $\lambda = 0.8$.

the performance variability of the green wave method significantly increases in near-saturated conditions (63% between the best and worst traffic scenario), while the I2DQN performance variability remains in line with normal traffic flow levels (29%), showing greater flexibility.

9.2.3.4 Overall Applicability of the Green Wave Coordination Method

Given the multiple experiments conducted in this section so far, our conclusions regarding green wave coordination effectiveness are contrasted. While our deep learning-based green wave coordination method is overall more efficient than the I2DQN controller, these results only in an experimental setting featuring an arterial with pre-computed offsets between intersections and regular traffic demand going mostly along this arterial; all of which highly favor the green wave coordination method. These superior results do not hold as soon as these specific settings are changed, as illustrated by the subpar performances of the green wave coordination methods in near-saturation traffic flows in the last section. This remark underlines the relative utility of the green wave coordination method since it requires a specific traffic demand and network topology to provide optimal performance. This point has been confronted before by Wagner et al. (2019) in their paper questioning the overall usefulness of direct agent coordination in TSC systems:

> While arterial coordination can be demonstrated to yield gains in efficiency under fairly mild conditions, the coordination of a whole transport system is not as simple. In addition to the mathematical and organizational challenges that come with this task, it is also not clear what can be gained. So, an optimum solution might turn out

to be just a few percent or so better in reducing delays, emissions, and even crashes, leading to the question whether it is worth the effort.

While we tend to agree with these conclusions concerning our experimental results, which explains why we do not pursue green wave coordination techniques further in this thesis work, it should be noted that an I2DQN method that could automatically switch to green wave coordination when specific traffic demand conditions are met (e.g., non-saturated traffic flows along arterials of a road network) would provide the best of both the I2DQN and green wave coordination methods. However, some computational challenges (e.g., implementing step-based green wave coordination, which is a non-trivial scheduling task, or automatically detecting arterials on a large road network and computing its offset values) would have to be tackled to implement such a hybrid method.

9.3 Indirect Coordination

In their state-of-the-art paper regarding cooperative multi-agent learning, Panait and Luke (2005) define indirect communication methods as "those which involve the implicit transfer of information from agent to agent through modification of the world environment.". In the realm of RL-TSC, the modification of the world environment usually occurs through modifications of the state space of the learning agents, as action space modified is associated with direct coordination (see section 9.4). Hence, indirect coordination applied to TSC relies on letting agents receive information besides their immediate local state without explicitly coordinating and making them exploit this additional data during their learning process.

9.3.1 Indirect Coordination Mechanisms

All MARL TSC methods using indirect coordination rely on state augmentation. A straightforward manner to augment the state space of a coordination-free TSC method to achieve indirect coordination would be to directly include features from the state information of neighboring intersections into the state space of each intersection of the network, hence increasing the knowledge of the true system state of each intersection. While such an approach is theoretically feasible and has even been applied in practice (Nishi et al., 2018; Wei et al., 2019b), it is subjected to dimensionality issues when the number of external state features increases. Even when parsimoniously including neighboring agent state features in an agent's state representation, the exploration process can become unbearably slow due to the curse of dimensionality (Genders, 2018). This increase in dimensionality is also the main obstacle to using centralized learning in RL-TSC applications, since the concatenation of all intersection-level state representations of an entire road network would make the state space of the problem skyrocket, making exploration impossible in practice for the centralized agent (Yau et al., 2017).

While indirect coordination methods suffer from certain limitations due to dimensionality considerations, they remain the most popular form of agent coordination in the RL-TSC literature mostly due to their flexibility and ease of implementation. Indeed, such methods rely on letting agents observe more features of the environment than what they would observe in an independent setting; and this additional data can be exploited in many different ways depending

on the underlying RL-TSC method, ranging from state augmentation to more complex joint action computation (see next paragraph). Consequently, indirect coordination methods are easier to develop and implement than direct coordination methods, requiring message passing between agents and explicit coordination mechanisms.

9.3.1.1 MARLIN-IC Algorithm

A prime example of indirect coordination circumventing dimensionality issues applied to traffic signal control is found in the MARLIN-IC algorithm designed by El-Tantawy and Abdulhai (2012) and presented in chapter 4. The MARLIN-IC algorithm is a model-based indirect coordination method that maximizes the utility of each agent of the network by first estimating the optimal joint policy of each agent and its neighborhood according to the principle of the locality of interaction then computes the associated optimal action using modular Q-learning. Given the prevalence of MARLIN-IC in the RL-TSC literature and its reported efficiency, this method has been ported to the `carmulator` library for comparison purposes.

The primary mechanism behind the MARLIN-IC algorithm is the computation of optimal joint action states between an intersection and its neighbors. For a given local state around an intersection v, s_v, the intersection computes the associated joint state (s_v, s_n) by observing the local state of each of its immediate neighbors in the network. The intersection then estimates the actions a_n each of its neighbors will take given this joint state action (by keeping a table of observations of past joint state and joint actions) and computes its optimal action based on the actions each of its neighbors are expected to take. This rather complex algorithm, which necessitates a Q-learning and past observation table for agent-neighbor couple, leverages indirect coordination by observing neighboring states and actions and strategy modeling through the estimation of neighboring actions without direct communication.

9.3.1.2 Deep MARLIN-IC Algorithm

The original MARLIN-IC algorithm has long been considered to be a state-of-the-art coordinated TSC method, which has showcased excellent results on large-scale traffic scenarios using real-world traffic data (Brys et al., 2014; Mannion et al., 2016; Yau et al., 2017). However, the field of RL-TSC has dramatically evolved since its original publication in 2012 and has most notably adopted more sophisticated function approximation techniques in order to improve agent performance and learning efficiency. The superiority of deep learning over traditional RL algorithms has already been demonstrated in chapter 8 and is also the central thesis of a 2018 paper co-written by one of the MARLIN co-authors (Shabestary and Abdulhai, 2018). We consequently decided to adapt the original MARLIN algorithm to newer function approximation techniques by using deep instead of regular Q-learning as the learning algorithm. If the adaptation of the original algorithm to its deep Q-learning variant (referred to as deep MARLIN) is straightforward, a couple of points should be noted. First, using a function approximation technique on MARLIN-IC could potentially break the theoretical guarantees of modular Q-learning. However, experimental results obtained with the deep MARLIN method show that this theoretical result has little importance in practice. Second, the addition of a function approximation technique on top of an already rather complex coordination method initially resulted in a volatile learning process, which is a common issue with indirect coordination techniques applied in non-stationary and complex

environments (Nowé et al., 2012). However, the addition of a single-agent learning layer, which is described in detail in the next paragraph, has entirely alleviated these issues.

9.3.1.3 Modified MARLIN-IC Algorithm

Since its source code is not (to our knowledge) publicly available, we have re-implemented the MARLIN-IC algorithm in Python to the best of our abilities from its description in the thesis manuscript of El-Tantawy and Abdulhai (2012). While most of the algorithm has been ported as-is without any problem whatsoever, we have noticed an undefined behavior in its original implementation. In some cases, an intersection has to pick its next action but has neighbors who do not choose any action for this time step because they are within a yellow or red phase. While the unavailable neighboring intersection can be skipped in most cases, this situation is problematic if the intersection choosing its action only has a single neighbor or in the rare cases in which all its neighbors are unavailable simultaneously since no joint actions can be computed. Furthermore, skipping some of the unavailable neighbors causes information loss, which is likely to degrade agent performance. In order to deal with this edge case, we decided to modify the MARLIN-IC algorithm structure to add a regular SARL learning algorithm below the joint-action coordination layer. Each intersection first learns from local states and actions similarly to any isolated RL algorithm (using a classical or deep learning form) and then augments the resulting value function through neighbor coordination as in the original MARLIN algorithm. The resulting modified MARLIN-IC algorithm, with this modification, is shown on algorithm 8.

Algorithm 8: Outline of the original MARLIN algorithm as described by El-Tantawy and Abdulhai (2012) with an additional learning layer. O_{vn} is an observation table listing the actions taken by agent n given the current joint state s_{vn} and $P_{a'_n}$ a function computing the probability of neighbour n choosing action a'_n based on the current joint state and this observation history.

for *each agent* $v \in V$ **do**
  Observe s_v, s'_v, a_v, r_v;
  $Q_v(s_v, a_v) \leftarrow (1-\alpha)Q_v(s_v, a_v) + \alpha(r_v + \gamma \max_a Q_v(s'_v, a_v))$;
  for *each neighbour* $n \in \Gamma(v)$ **do**
    Observe s_n, s'_n, a_n;
    $O_{vn}(s_{vn}, a_{vn}) \leftarrow O_{vn}(s_{vn}, a_{vn}) + 1$;
    $Q_{vn}(s_{vn}, a_{vn}) \leftarrow (1-\alpha)Q_{vn}(s_{vn}, a_{vn}) + \alpha(r_v + \gamma \max_a Q_{vn}(s'_{vn}, a_{vn}))$;
  $a'_v \leftarrow \max_{a*_n} Q_v(s_v, a_v) + \sum_{n\Gamma(v)} Q_{vn}(s'_{vn}, a^*_{vn}) P_{a'_n}(O_{vn}, s'_{vn}, a^*_v))$;

The modified MARLIN-IC algorithms, both in their classical or deep form, both strongly benefit from the introduction of this single-agent learning layer. In order to measure the benefits of this change, we tested four variations of the MARLIN-IC algorithm: two in its original form (one featuring this additional layer, one without), two in its deep learning form (one featuring this additional layer, one without). These four RL-TSC methods are tested on a two-by-two network featuring four intersections, with an overall arrival rate parameter of $\lambda = 0.015$.

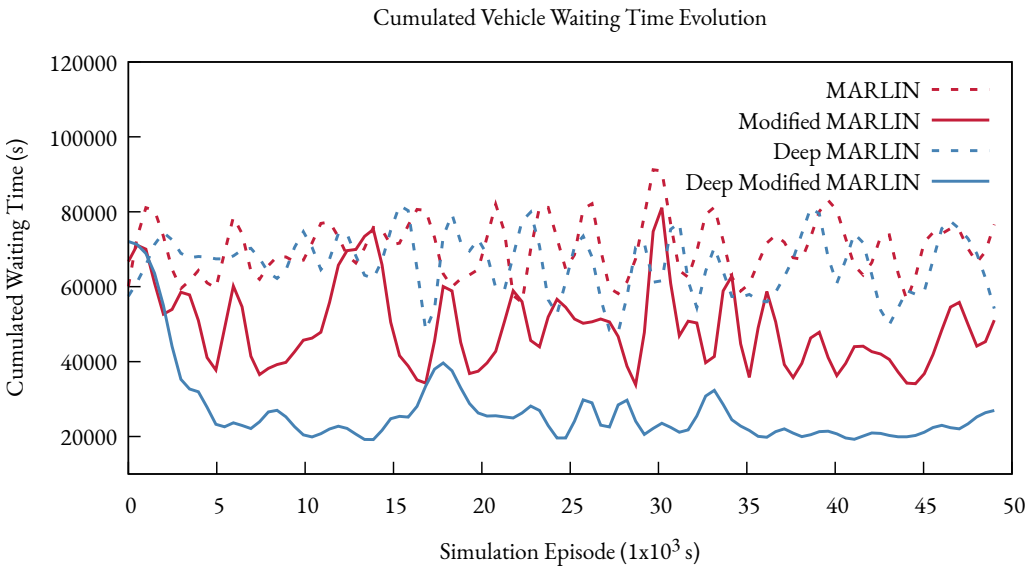

Figure 9.8: Performance comparison of the regular and modified MARLIN-IC methods in their classical and deep forms. Classical MARLIN methods are in red; deep MARLIN methods are in blue. Dashed lines represent the original MARLIN algorithm, full lines our modified version. Results are averaged over five simulation scenarios.

Our testing lead to two key observations. First, the addition of a single-agent learning layer greatly improves performance for both the classical and deep MARLIN methods, as seen when comparing plots of the same color on Figure 9.8. Second, we see that the deep MARLIN algorithm significantly outperforms the classical MARLIN algorithm in their modified versions, as seen when comparing the full plots of the same figure. Most notably, we notice that the deep MARLIN algorithm using a single-agent layer achieves remarkable stability quite early in the convergence process. On the basis of these results, we retain the modified versions of the MARLIN algorithm, both in its classical and deep form, given their superior performances. These algorithms will be referred to as MARLIN and deep MARLIN from now on for simplicity's sake.

9.3.2 Measuring The Impact Of Indirect Coordination

The aim of the MARLIN algorithms we are testing–exploiting joint state and action observations to maximize utility at the neighborhood level–is clear. We now need to evaluate its relative efficiency compared to independent methods such as the 2DQN algorithm tested in the previous section in order to establish if, and possibly why, such a form of coordination is beneficial to traffic optimization.

9.3.2.1 Indirect Coordination Convergence

The first efficiency measurement is done regarding the convergence process of the original and deep MARLIN algorithms, which are compared to an independent I2DQN method and the optimal and Webster deterministic methods for comparison purposes. These methods are tested on a two-by-two grid network composed of 4 intersections by using the convergence protocol defined in section 7.3.1. Since indirect coordination techniques do not have the special requirements seen with green wave coordination methods, there are no additional constraints regarding traffic generation or intersection cycle time. Traffic flows between each pair of edges of the network are generated using an exponential law of parameter $\lambda = 0.015$, which corresponds to low to regular traffic demand over the network.

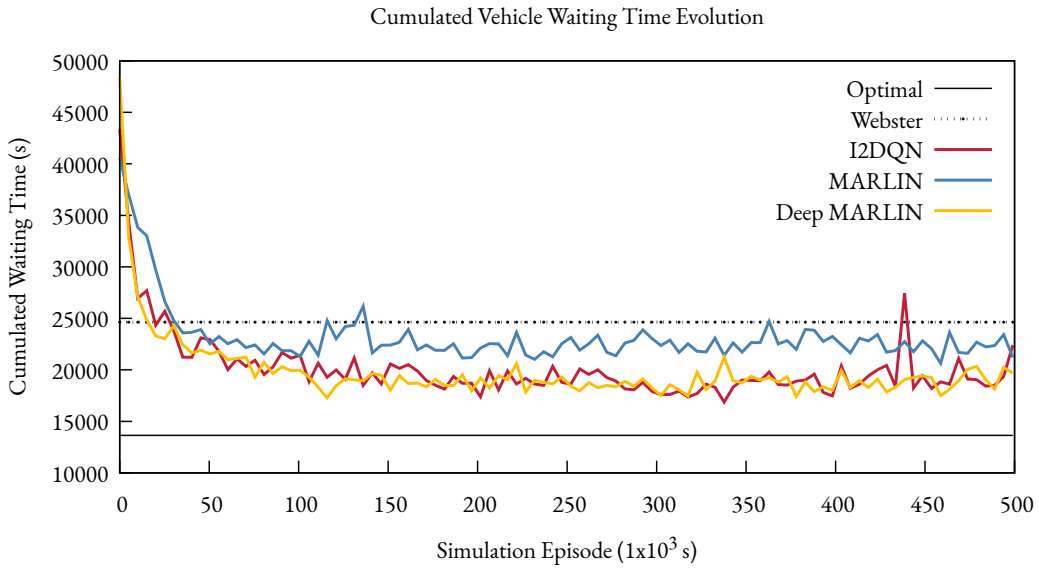

Figure 9.9: Convergence process of the I2DQN, MARLIN and Deep MARLIN methods. Webster and Optimal fixed methods are included for comparison purposes. Average values over 5 simulation scenarios. Smoothed results.

We first analyze the convergence process of our selected methods for 500 episodes of 1000 steps each. The results of these traffic simulations, presented in Figure 9.9 show first and foremost the strong convergence stability of both MARLIN algorithms and, to a lesser extent, of the I2DQN algorithm. Similarly to tabular value function algorithms tested in section 8.2.4, the original MARLIN algorithm (in blue in the figure) is unable to improve early on, hinting at its inability to learn further from its environment due to the absence of function approximation mechanisms. Conversely, both methods featuring function approximation techniques quickly converge to higher performance levels. Additionally, we notice that the two-layer approach to coordination we offered to limit convergence instability in the deep MARLIN case performs exceedingly well given that the deep MARLIN methods we tested without this approach were more unstable than the I2DQN algorithm shown on Figure 9.9. While these initial results confirm the intuition that

the classical MARLIN method does not perform as well as deep reinforcement learning methods leveraging neural networks for state generalization, low and steady traffic demand scenarios do not allow to measure how adaptive and resilient both deep RL methods are in order to compare them truly.

9.3.2.2 Indirect Coordination Under Variable Traffic Flows

Since the I2DQN and deep MARLIN method showcase quite similar performance levels in the experiments of the previous section, we design an experiment aiming to test their capabilities under changing traffic conditions. The experimental protocol we designed in section 7.3.3 compares the robustness of both methods by gradually increasing traffic demand over multiple simulation episodes, testing their robustness in the process. For this experiment, we generate traffic data using an exponential law of parameter 0.015 (using the protocol defined in section 7.2.2). This traffic generation results in randomly generated arrival flow rates over each entry-exit edge pair of the network. After running a hundred simulation episodes using these regular weights (in order to make both methods converge), we gradually increase the arrival rate of each edge pair of the network by 0.8% each step for 100 steps, reaching an overall arrival rate of around 0.04, before decreasing by 1% each step for 100 steps, returning to a pre-rush hour traffic demand. Hence, each TSC method will learn to route vehicles in increasingly saturated traffic conditions while ensuring that the traffic demand imbalances that exist in the network are maintained. Furthermore, once peak-hour traffic conditions are passed, gradually lowering traffic demand will allow to observe which methods can quickly return to pre-rush hour performance levels, denoting greater adaptability.

We present these simulation results in Figure 9.10. As mentioned in section 7.3.3, the areas plotted in this figure correspond to the performance spectrum of a given RL-TSC method delimited by its best and worst observed cumulated waiting times for a given simulation episode. Additionally, a solid line plots the average cumulated waiting time across all simulation scenarios and represents the average performance level of the TSC method for this given traffic network and demand. Results of Figure 9.10 do show that while the low-traffic demand situations of the first simulation steps result in somewhat equivalent performance levels from both I2DQN and deep MARLIN (even though the latter continues improving beyond the former after iteration 70, which is not visible in Figure 9.10), increasing this traffic demand allows differentiating both methods further. The initial increase in traffic demand immediately decreases the stability of the deep MARLIN method (see sub-plot 1), while the I2DQN method maintains greater stability during these initial steps. As congestion keeps increasing, however, the I2DQN also suffers from increased performance variability, and to a greater extent than the deep MARLIN method (see sub-plot 2). This causes I2DQN to display lower performance levels compared to the deep MARLIN method, as seen on sub-plots 2, 3 and 4 of Figure 9.10. Finally, we note that the deep MARLIN method displays superior resilience after sustaining a brutal increase in traffic, as its variance in performance quickly decreases when traffic demand levels go back to normal (see sub-plot 4).

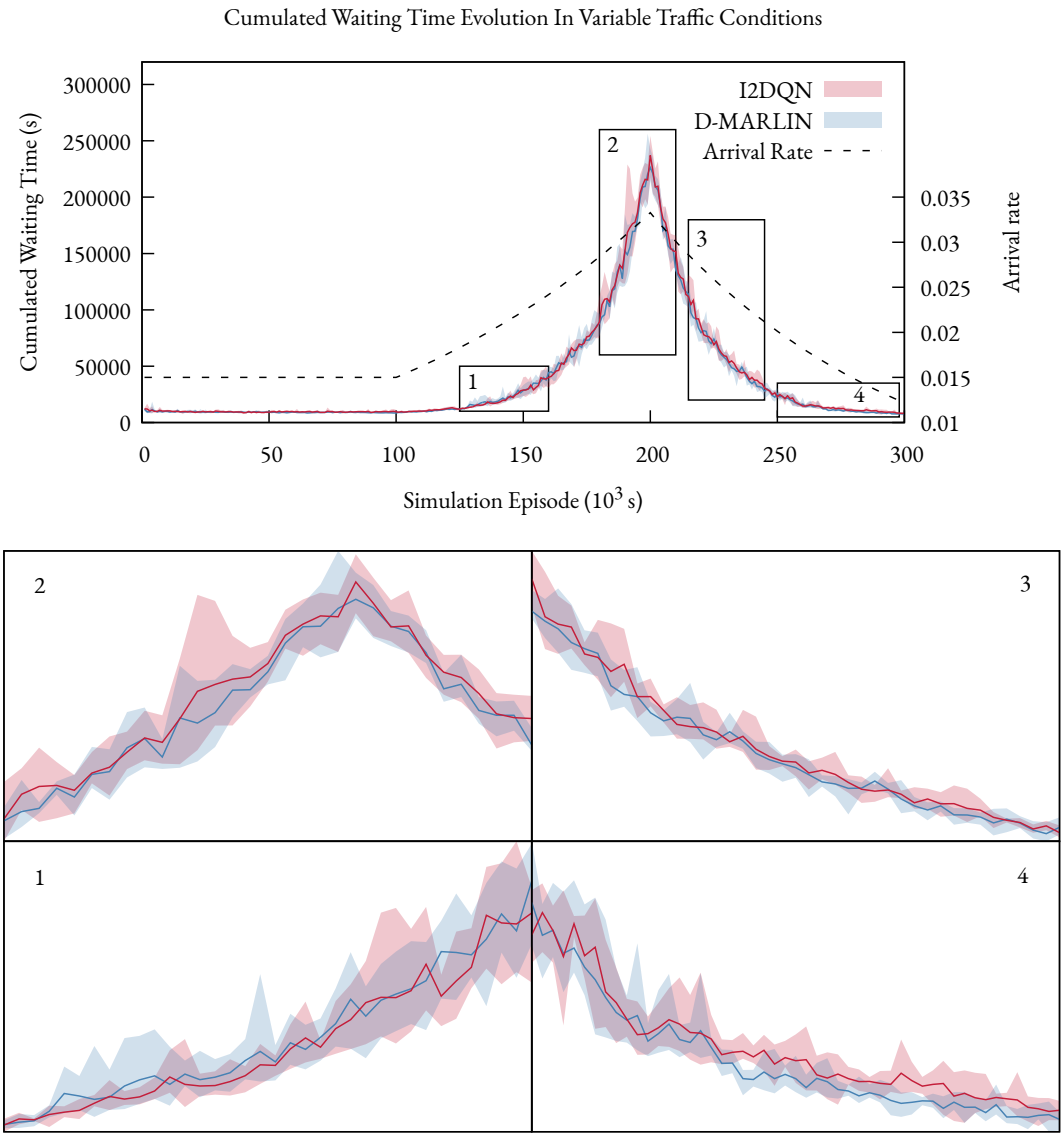

Figure 9.10: Analysis of I2DQN and deep MARLIN under variable traffic conditions.

In conclusion, the deep MARLIN algorithm provides better performance overall in variable traffic demand conditions than its independent counterpart I2DQN, even though deep MARLIN seems more susceptible to performance instabilities for minor variations of traffic demand (see subplot 1 of Figure 9.10). Moreover, deep MARLIN also proves more resilient to these traffic changes as it quickly stabilizes its performance levels once traffic demand goes down.

9.3.2.3 Measuring the Influence of Joint State-Action Modeling

The previous subsection has shown that indirect coordination between intersections of a road network allowed from similar to superior performances compared to independent methods. This

section aims to explain *how* such coordination mechanisms provide an advantage to independent methods.

MARLIN Coordination Mechanism The critical coordination mechanism behind the MARLIN and deep MARLIN algorithms is the joint state and action modeling between an intersection and its neighbors. When computing quality estimates associated with different actions, intersections do not only use the local traffic state but also take into account the probable future action of each of its neighbors, given their local traffic state. While such an approach has empirically proven its efficiency (El-Tantawy and Abdulhai, 2012), no clear-cut explanation has been given as to why observing the traffic state of neighboring intersections can improve the learning abilities of an intersection.

Joint Action Modeling We argue that modeling neighboring intersection is often useless since their actions do not have time to impact the local traffic state from one step to the next. Indeed, it should take around 7 seconds (or steps in our case) for a vehicle to travel from one intersection to the next in the quad network used in our experiments (intersections are spaced 100 meters apart). While it should logically be argued that a vehicle influences an intersection–through state and reward computation–the moment it enters one of its incoming lanes, a single step is not long enough for a vehicle to entirely travel the crossing area of its origin intersection. This means that the influence of an intersection on its neighbors should range from null to minimal between two successive time steps. This observation should, in theory, imply that neighboring intersection state and action modeling has virtually no effect on the computation of quality estimates for the "extend" action since its effect is measured from one time step to the next. However, neighboring states and actions influence the quality estimate of the "switch" action whose effects are measured around ten steps after the action has been taken, long enough for neighboring traffic to reach the local intersection.

Measuring the Influence of Neighboring Features We designed a specific experiment in order to test this hypothesis. We first trained a regular deep MARLIN controller under normal traffic conditions and extracted one of the resulting neural networks from one of its controllers. Using the neural network associated with the joint state-action modeling between intersections u and v, we are able to measure the estimated quality value $Q_{uv}(s_{uv}, a_{uv})$ measuring the quality of any given joint state-action couple (s_{uv}, a_{uv}) given as an input (see algorithm 8). This allows, among other things, to measure how much a change in the local state s_u or neighboring state s_v influences the quality values $Q(s_{uv}, a_{uv})$ of actions "extend" and "switch". In practice, we sample multiple states that have been encountered during the learning process from the memory replay buffer $\mathcal{D}$. Eached sampled state s_{uv} has the form:

$$s_{uv} = \langle \phi(u), d(u), c(l_{u1}), \dots, c(l_{un}), \phi(v), d(v), c(l_{v1}), \dots, c(l_{vn}) \rangle$$

which is a concatenation of the local states of intersections u and v as defined in section 6.3. For each sampled state s_{uv}, we first measure the quality values $Q(s_{uv}, \text{"extend"})$ and $Q(s_{uv}, \text{"switch"})$ associated with both actions available to the intersection u. Then, we change the values of $\phi(u)$ and $\phi(v)$ in s_{uv}, which represent respectively the current phase index of intersection u and v.

Since intersections in the quad network have two main green phases (i.e. east-west and north-south), this change switches the currently active green phase from one to the next. Using these alternative state definitions, we measure the new associated quality values in order to measure how they differ from the original values $Q(s_{uv}, \text{“extend”})$ and $Q(s_{uv}, \text{“switch”})$.

The idea behind these measurements is the following. The more a given feature of the state space influences the reward of an agent, the more a change of its value will change the quality value associated with it, *ceteris paribus*. In the case of our experiment, the feature of the state space indicating the currently active phase of an intersection is essential in choosing the next action. If the lanes along the north-south axis of an intersection are heavily congested, the quality of the “switch” action is likely to be high if the currently active phase is east-west and very likely to be low if it is north-south. Hence, measuring the difference in quality values after changing a feature of the state space gives an idea of how much this feature matters to the agent and its reward. By measuring these differences, we hope to show that switching the local phase index $\phi(u)$ highly matters for intersection u, while switching the neighboring phase index $\phi(v)$ does not for the “switch” action.

Experimental Results Results of Table 9.2 show these differences in quality values after sampling 5000 different states. The values μ in the table represent the average difference in quality after a phase switch, while σ represents the variance in average difference in quality after a phase switch. These results underline, as expected, the massive difference between local and neighboring state changes. As we can see in Table 9.2, a change in the neighbor state has a close to zero impact on the “extend” action of the local controller (both the average difference in variance difference being close to 0), while having a significant impact on the “switch” action. Furthermore, we can see on the first row of this table that the local phase index feature of the state space significantly impacts the quality value of both actions, indicating its significant influence on the agent's decision.

	μ(extend)	μ(switch)	σ(extend)	σ(switch)
Local phase change	3.43	1.19	41.25	4.86
Neighbor phase change	0.35	1.01	0.54	7.47

Table 9.2: Average difference (μ) and variance in average difference (σ) of value estimates for 5000 states.

This experiment hence confirms that the advantage of the deep MARLIN method over independent algorithms such as I2DQN resides in its ability to compute quality estimates of phase-switching actions better since the traffic neighboring intersections will influence local traffic while this action is being applied. Conversely, our findings indicate the joint state-action modeling of deep MARLIN is useless when computing quality estimates of the phase extension action since the neighboring traffic does not have the time to influence the local traffic, which represents both a missed opportunity to extract additional information from neighboring states and a method weakness since it introduces unnecessary computation and potential instability.

Delaying Joint Action Modeling A logical argument that could be made regarding the inability of deep MARLIN to properly take neighboring states into account from one step to the next would be to delay the time step at which the intersections receive neighboring states. One could imagine that modeling the neighboring state from a couple of steps prior would leave time for this anterior state representation to affect the local traffic state. This solution would, however, only displace the issue. Since intersections using MARLIN learn through joint-action modeling, the delayed state representation of neighboring intersections would already be incorporated in the local intersection state, still rendering the delayed neighboring state representation unable to influence the "extend" action quality estimates. The direct coordination method presented in the following subsection aims to provide quality estimates for both action types through message passing rather than through joint state modeling.

9.4 Direct Coordination

Direct agent coordination, also known as *explicit coordination* (Busoniu et al., 2008) pushes agent interaction further than indirect coordination by allowing for direct exchange of information while learning. The main difference between indirect and direct coordination methods is that the latter does not only receive information from other agents of the environment but also directly take other agents into account in their decision-making process through explicit message passing mechanisms or joint action computation.

9.4.1 Direct Coordination Mechanisms

Similarly to indirect coordination methods, a wide variety of algorithms can be used for direct coordination of traffic lights since their only requirement is the direct exchange of information between learning agents. Hence, while multiple direct coordination mechanisms exist for RL-TSC systems, such as the max-plus algorithm (Kok and Vlassis, 2005; Van der Pol and Oliehoek, 2016), we focus here on two different coordination algorithms: the MARLIN-DC algorithm, which is the direct coordination version of the MARLIN-IC algorithm that we studied in the previous section, and the RIAL and DIAL algorithms (Foerster et al., 2016) which features self-learning communication between agents of the same environment.

9.4.1.1 MARLIN-DC Algorithm

The MARLIN-DC algorithm (El-Tantawy and Abdulhai, 2012) leverages direct negotiation between agents in order to compute optimal joint policies. Similarly to MARLIN-IC, each agent maintains a Q-table with each of its neighbors containing quality estimates according to the joint state-action of the intersection and its neighbor. When choosing an action, an intersection does not only compute its optimal action according to the joint state-action space with each of its neighbors (similarly to MARLIN-IC) but also estimates the optimal action of each of its neighbors by directly using their policies. Using this additional information, the agent can then compute its best-response action with regard to the actions of its neighbors and estimate the difference in utility between its original optimal action and this best-response action. After this first computation step, intersections directly coordinate themselves by broadcasting their difference in utility to

their neighbors: the intersection with the maximal utility difference in its neighborhood is allowed to change its original action to the best-response action. By repeating this process by descending order of difference in expected utility, the MARLIN-DC algorithm reaches an equilibrium that is expected to maximize the joint expected reward of a neighborhood of intersections. While this algorithm offers a novel direct coordination method to optimize traffic signal control, experimental results presented by El-Tantawy and Abdulhai have shown that MARLIN-DC provides similar to slightly worse performance levels compared to MARLIN-IC, while increasing its computation time fivefold. These limitations motivated El-Tantawy and Abdulhai to only study MARLIN-IC in large-scale simulation scenarios and hence prevented us from implementing and testing the MARLIN-DC algorithm in `carmulator`.

9.4.1.2 Reinforced and Differentiable Inter-Agent Learning

A promising technique for agent coordination known as reinforced inter-agent learning (RIAL) has originated in a paper by Foerster et al. (2016). In its original version, the RIAL algorithm is applied to a fully cooperative, partially observable, and sequential multi-agent learning problem in which communication is essential. Without any communication protocol defined beforehand, agents must learn to communicate through limited channels during each step of the game in order to maximize their shared rewards. Agents must not only learn to solve their tasks, but they must learn and agree on a common communication protocol. During learning, each agent chooses two distinct actions using two distinct neural networks: a traffic action a and a message action m to send to the other agents. The selected actions and messages of each agent are then observed by all other players on the next step as part of the state space definition.

The original RIAL algorithm has been extend by the differentiable inter-agent learning (DIAL) algorithm which not only shares messages across agents, but also gradients used to reward communication actions. A single neural network is shared by all agents for choosing the communication action. Each agent using the communication neural network has a unique index variable as an input, allowing them to specialize. Such a method requires centralized learning since learning parameters cannot be shared through limited communication channels. Since centralized computation is the norm in traffic simulations, parameter sharing through the use of a single neural network between agents is possible in RL-TSC applications, and has already been benchmarked in that manner (Chu et al., 2020; Vanneste et al., 2021). While results from these papers have been encouraging (they have both beaten independent learning baselines), we believe that the DIAL algorithm is not optimal for RL-TSC tasks. The remainder of this chapter explain why this is the case, and how our proposed method, DEC-DQN, addresses these issues.

9.4.1.3 Adapting The Architecture

The first issue regarding the application of DIAL to traffic signal control comes from the neural network architecture being used in the original algorithm. Indeed, the learning task of choice of the original DIAL paper is a switch-riddle game that is both relatively short (in terms of learning episodes) and simple (in terms of state-action space definition). Additionally, since taking a wrong decision can cause the game to end early, tracking previous states of the game was deemed essential and was done using a gated recursive unit network, which is a form of recurring neural network.

If this neural network architecture fitted this type of game, traffic signal control would be entirely different for multiple reasons. First, traffic optimization is neither short (since the learning task goes indefinitely) nor simple (since state-action spaces are a magnitude more complex than the switch-riddle games). These factors significantly increase the complexity of gradient computation in recurring neural networks since the input data is larger (states are complex) and wider (episodes are longer), which considerably lengthens the learning process of each agent. Additionally, and perhaps more importantly, we have found that keeping track of past system states using a recurrent neural network mattered little in our TSC setting, implying that only the immediate system state and the action applied to it were influencing the utility of an agent. While a few papers of the literature use recurrent neural networks in their deep learning models (Chu et al., 2019; Ma and Wu, 2020; Shi and Chen, 2018; Xiong et al., 2019; Zeng et al., 2018), they also use complex DTSE state representation (see section 4.1.2.3) through image inputs, which justifies the use of recurrent neural networks alongside convolutional layers in order to learn from image input data.

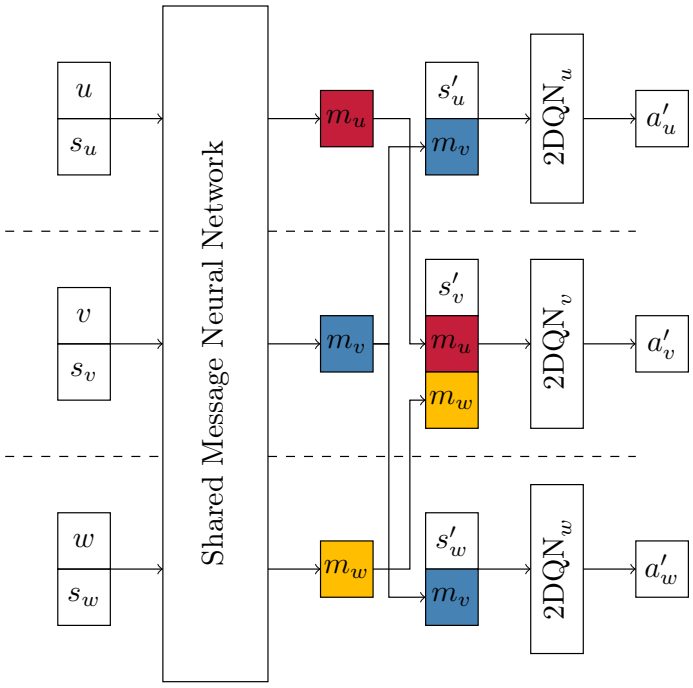

Figure 9.11: Illustration of the modified DIAL architecture applied to a road network of three aligned intersections, u, v and w. At each step, each intersection chooses a communication action m on the basis of its local state and unique intersection index on the shared communication neural network. The resulting message is then sent to neighboring intersections at the next time step and is used by neighboring intersections to chose a traffic signal control action a.

In the case of simpler state representation, such as ours, testing multiple architectures of recurrent neural networks (long short-term memory and gated recursive unit) in the isolated and coordinated cases has resulted in subpar performances in all cases. Hence, since recurrent neural networks provide both inferior performance results and a significant increase in computational needs for learning, we retain the dual network architecture of the original DIAL algorithm but re-

place the recurring components by simple fully connected layers similarly to the I2DQN method. We represent the chosen architecture on Figure 9.11.

9.4.1.4 Rewarding Communication Actions

The second challenge and main caused by applying the DIAL algorithm on a traffic signal control task comes from associating rewards to message actions taken by agents at each step. This adjustment issue is again due to the type of task on which the original DIAL algorithm was applied. In the switch-riddle game, the same reward is used to both environment and message actions a and m since the game is fully cooperative and can only end up in the death or liberation of all prisoners. Hence, the same reward is shared across all agents, and both actions are working towards the same optimization goal. In the case of traffic signal control, rewards are neither shared between agents nor impacted in the same way by environment and message actions. Each agent locally optimizes traffic through action a to maximize its local utility and sends a message m (in the form of an integer), which will be received at a future time step by its neighbors so that they can maximize their local utility. While local traffic related-actions can hence still be rewarded directly using traffic delay-related measurements, message-passing actions are much harder to estimate since agents have no innate mechanisms to estimate *if* a message they sent has been taken into account by a neighboring intersection, and, if so, *how* this message has affected their local utility. Furthermore, since intersections with multiple neighbors receive multiple messages each turn, computing the individual reward associated with each of these messages poses an issue of *credit assignment* (Panait and Luke, 2005; Sutton and Barto, 2018) since we do not know how to divide the reward between each neighbor. These issues underline the fact that applying a DIAL-type algorithm in a traffic signal control context requires to define a reward function specifically designed to reward communication actions.

We design the direct-evaluation communication DQN (DEC-DQN) method to address this challenging task. The DEC-DQN algorithm features a reward function specifically geared towards communication actions which directly estimates how agent communications affect those who receive them. This estimation is made possible by the fact that traffic simulations allow for centralized training and parameter sharing. It is hence possible, within a simulation episode, for an agent not only to access the shared neural network used for message action selection but also to access the neural networks used by neighboring intersections in order to choose their traffic-related actions. By supposing, furthermore, that agents can observe all communications passed between agents, the reward function used in the DEC-DQN leverages an idea similar to state feature estimation in section 9.3.2.3. The reward of a communication m made by an intersection v is computed by estimating the opportunity cost of sending message m for each neighbor n of v. This opportunity cost is obtained by plugging the state of neighbor n containing the original communication action m into its neural network in order to observe the maximum attainable quality estimate that neighbor n can reach. Then, this quality value is compared to all other *potential* maximum quality estimates neighbor n *could have gotten* had agent v sent a different communication action m. The higher the difference between the quality estimate associated with the best potential communication action $m*$ and the sent communication m, the higher the opportunity cost. In other words, the more an intersection makes its neighbors miss on high-quality estimates due to a given communication action it sent, the higher the loss associated with this communica-

tion should be. The detailed pseudocode illustration of the loss computation of communication actions of the DEC-DQN algorithm is illustrated in algorithm 9.

Algorithm 9: Algorithmic illustration of the loss computation for message actions in the DEC-DQN algorithm. The reward associated with the message m_v chosen by intersection v is computed by estimating, for each intersection neighboring v, the maximal expected reward this neighbor could attain when receiving message m_v and the maximal expected reward it could attain by receiving any other message from intersection v. The higher the difference between these two maximal expected rewards, the more intersection v should have chosen a different message, and the higher the associated loss is.

for *each agent* $v \in V$ **do**
  $\mathcal{L} \leftarrow 0$;
  Observe m_v;
  for *each neighbour* $n \in \Gamma(v)$ **do**
    $M \leftarrow (m_1, \ldots, m_{v-1})$ for $[1, v-1] \in \Gamma(n) - \{v\}$;
    $s' \leftarrow (s'_n, M, m_v)$;
    $V_{\max} \leftarrow \max(V_{\theta_n}(s'))$;
    for $m_{alt} \in \mathcal{A}_m - \{m_v\}$ **do**
      $s'_{\text{alt}} \leftarrow (s'_n, M, m_{\text{alt}})$;
      if $\max(V_{\theta_n}(s'_{alt})) > V_{\max}$ **then**
        $V_{\max} \leftarrow \max(V_{\theta_n}(s'_{\text{alt}}))$;
    $\mathcal{L} \leftarrow \mathcal{L} + ||V_{\max} - \max(V_{\theta_n}(s'))||$

9.4.1.5 Choosing DEC-DQN Parameters

The proper tuning of parameters is essential in most deep learning models, and this is perhaps even truer in the DEC-DQN case when compared to other deep RL-TSC models such as I2DQN. Indeed, on top of sharing its model parameters with I2DQN, the coordination-specific mechanisms of DEC-DQN need to be correctly parameterized to pass messages between intersections of the network efficiently. This section hence aims at finding proper values for two parameters of the DEC-DQN algorithm for TSC. First, this section studies the effect of the size of communication channels between agents, also defined as the size of the action space for communication actions $\mathcal{A}_m$. Finding the correct communication channel size is a trade-off. Too narrow of a channel might not be able to express sufficiently different messages, limiting the usefulness of agent communication. Conversely, too large of a channel increases dimensionality and prevents the agents from converging on a common communication protocol, limiting once again the usefulness of agent communication. The second parameter we aim to estimate is the effect of delay between the emission of a communication action by an intersection and its reception by its neighbors, which was a fundamental limitation of the indirect coordination mechanism of MARLIN, as seen in the previous section.

Communication Channel Size The size of the communication channel $|\mathcal{A}_m|$ is paramount in proper communication between agents of the DEC-DQN algorithm. Messages are sent by

agents as integers to their neighbors in order to convey information about their local situation without explicit constraints as to what these messages represent. As this communication protocol is learned, the size of these communication channels represents the depth, or richness, of what these messages can convey. As such, a small message action space, similarly to a small state space $\mathcal{S}$, conveys less information to the agent but is likely to converge faster due to its reduced dimensionality. Conversely, a large amount of communication channels allows the agents to exchange more precise data at the cost of a longer convergence process. Since the optimal size of the communication space is highly likely to be problem-dependent, we experiment with various communication channel sizes in a simulation setting. We compare the convergence process (as described in section 7.3.1) of three DEC-DQN algorithms with different communication channel sizes on a two-by-two grid network in order to estimate their influence on the agent's convergence process. Note that this experiment is carried out over a relatively long number of simulation episodes, 1000 instead of the usual 100, in order to observe the very-long term convergence of DEC-DQN methods using large communication channels.

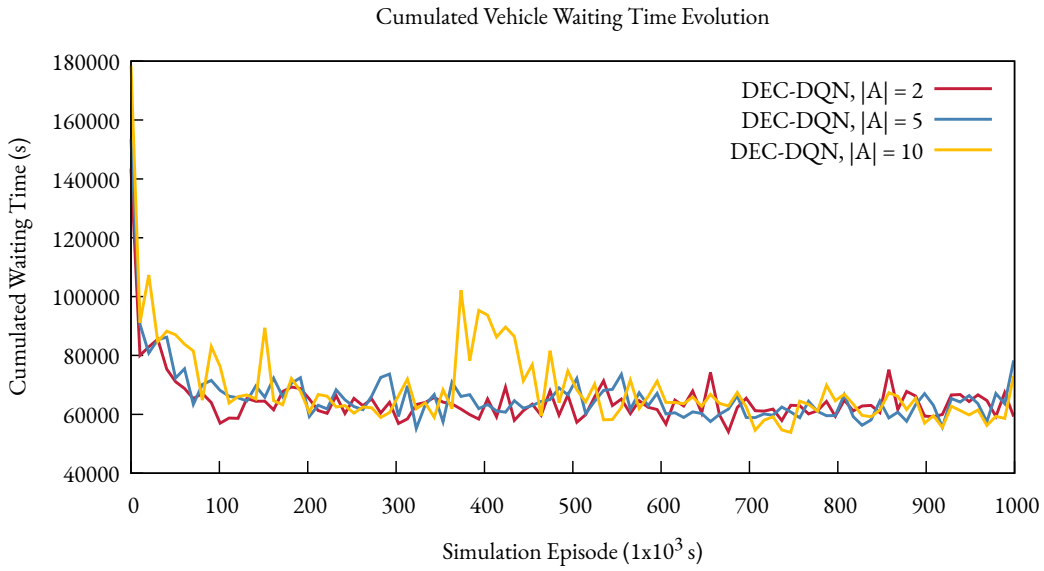

Figure 9.12: Convergence process of the DEC-DQN algorithm per comunication channel size.

Results of this experiment, averaged over five traffic scenarios, are shown on Figure 9.12. The first observation is that communication channel size does not seem to have a significant effect on very long-term convergence, as all three DEC-DQN methods show similar performance values in later simulation episodes. However, we do observe that the DEC-DQN algorithm using the largest communication channel size (in yellow in the figure) displays greater convergence instability, which is noticeable around episode 400 on Figure 9.12. Since increasing the channel size of communication actions does overall not seem to yield specific rewards for this specific learning problem but does increase learning instability, we choose the simplest message action space in order to reduce dimensionality, which is $|\mathcal{A}_m| = 2$.

Message Delay As underlined in section 9.3.2.3, the influence of the delay at which intersections receive information from their neighbors is crucial for adequately computing the quality estimates of traffic actions at its disposal. Since, as opposed to the MARLIN algorithm, the delay between the emission and reception of a communication action is configurable in the DEC-DQN algorithm, it is hence essential to analyze the effect that such a delay may have on quality estimates. To this end, we repeat the experimental protocol described in section 9.3.2.3 which allows to estimate the impact of messages on the quality estimates of an intersection by directly measuring it on their neural networks. In this setup, we train ten distinct DEC-DQN agents using increasingly larger message reception delays Δ_t ranging from 1 to 10 on quality estimates similarly to what has been done in the indirect coordination case. Each agent has exactly one neighbor who sends one of two communication actions, since $|\mathcal{A}_m| = 2$, as stated in the previous section. We then sample observations from the replay buffer $\mathcal{D}$, and for each observation, measure the effect of flipping the original message sent by the neighboring intersection on the associated quality estimates of the agent. The average difference in quality estimates for a message and its flipped variant, $\mu(a)$, is then computed for a given action a. The results of these computations, for multiple message delays Δ_t, can be found in Table 9.3.

	$\Delta_t = 1$	$\Delta_t = 2$	$\Delta_t = 3$	$\Delta_t = 4$	$\Delta_t = 5$
μ(switch)	75.91	60.33	63.64	54.98	41.68
μ(extend)	55.03	59.05	13.86	31.15	9.05
	$\Delta_t = 6$	$\Delta_t = 7$	$\Delta_t = 8$	$\Delta_t = 9$	$\Delta_t = 10$
μ(switch)	116.83	72.49	65.57	70.35	92.41
μ(extend)	128.6	44.48	45.05	51.5	62.57

Table 9.3: Average difference of quality estimates for different messages and delay values Δ_t.

The results from these experiments raise two critical points. First, the communication actions of a neighbor influence the quality estimates of all actions by the agent. Indeed, μ values of Table 9.3 are all significantly different from 0, indicating that a switch in the neighbor's communication action has a substantial impact on the agent's expected reward. This first observation shows that DEC-DQN solves the primary issue associated with deep MARLIN: its inability to influence a neighboring intersection from one step to the next. Since DEC-DQN has by a minimal delay of 1 step by construction, it does not suffer from this shortcoming. Furthermore, we observe that the average difference in quality estimates does not tend to substantially change as the message delay parameter Δ_t increases. While we could interpret this phenomenon as proof that communication does not affect agent performance (although upcoming experimental results show otherwise), we understand it as a proof of DEC-DQN's adaptability. Since the communication protocol is learned from scratch by agents, messages passed between agents do not have to represent fixed, time-dependent, state information like in MARLIN but can represent any feature of the environment. Hence, we suppose that the features that are chosen to be included in these communication protocols are likely to change depending on the chosen delay value (e.g., lower communication delays are, for instance, likely to favor features that are more likely to impact neighboring intersections immediately). However, its impact on agent performance does remain

somewhat constant. These findings hence tend to indicate that while the message delay parameter Δ_t dictates which kind of state features are used by agents to craft their communication protocols, they all tend to have a similar impact on resulting agent quality estimates. In light of these findings, we hence opted for the most straightforward message delay parameter value, $\Delta_t = 1$.

9.4.2 Measuring the Impact of Direct Coordination

This section is dedicated to analyzing the performances of DEC-DQN in a TSC setting. This analysis compares the DEC-DQN algorithm to a baseline independent deep Q-learning using the I2DQN algorithm. We first measure the convergence of both methods before evaluating their performance capabilities in regular and saturated traffic conditions.

9.4.2.1 Direct Coordination Convergence

The first experiment compares the convergence of both methods according to the protocol defined in section 7.3.1. We generate traffic demand data on a 2x2 grid network according to the protocol defined in section 7.2.2, using a constant arrival rate averaging $\lambda = 0.018$ vehicles per step. The convergence process of the I2DQN and DEC-DQN algorithms is plotted on an average of 10 distinct traffic scenarios, each running for 500 episodes of 1000 steps each. As mentioned in the previous section, the chosen DEC-DQN parameters are a default message reception delay of $\Delta_t = 1$ and a communication action channel of $|\mathcal{A}_m| = 2$. The learning hyperparameters used for this experiment are similar to those listed in Table 8.2 and Table 8.4.

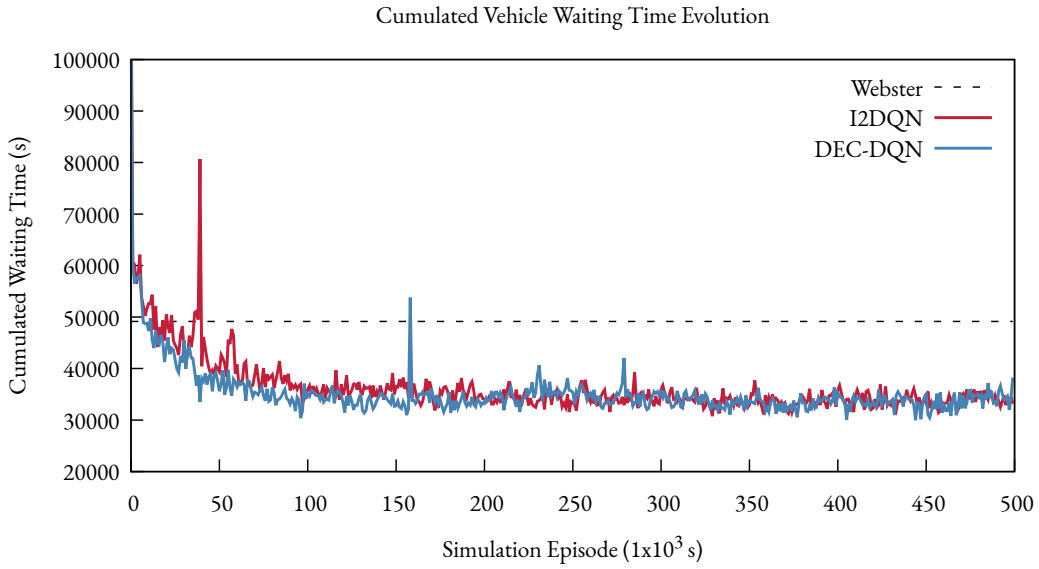

Figure 9.13: Convergence process of the I2DQN and DEC-DQN algorithms.

The results of these experiments, as shown on Figure 9.13, tend to differentiate both methods slightly more than in the indirect coordination case. While both the coordination-free and di-

rect coordination-based methods once again end up achieving similar performance levels in the later stages of simulation on all tested scenarios, the DEC-DQN algorithm converges significantly faster in the early traffic simulation episodes (we measured an average difference in performance of 9.76% during the first 100 episodes on Figure 9.13). While displaying the relative superiority of direct coordination to independent learning, this result is somewhat surprising. Since DEC-DQN uses a larger state space than I2DQN, and since it necessitates to train both a traffic-related and communication-related neural network, we would have expected a longer convergence process than the independent learning method, eventually reaching superior performance levels. However, Figure 9.13 seems to show that agents, in this simple traffic scenario, manage to quickly converge on a common communication protocol which enables the agent to reach excellent performance levels quickly.

9.4.2.2 Direct Coordination Under Variable Traffic Flows

An extensive way of measuring the overall efficiency of a RL-TSC method is to analyze its behavior under variable traffic conditions. To this end, we replicate the experimental setup of section 9.3.2.2 using increasing and decreasing traffic flows to compare the I2DQN and DEC-DQN methods using the same experimental parameters. This experiment is designed to estimate how agent coordination can help intersection using DEC-DQN to adapt to changing traffic conditions.

The results on Figure 9.14, averaged over 5 distinct traffic scenarios, confirm our initial observations. Subplots 1, 2, and 3 of this figure, respectively associated with an increase, peak, and decrease in traffic arrival rates, all show that the DEC-DQN algorithm (in blue) is superior to I2DQN (in red) both in terms of average performance (as shown by the solid-colored lines) and in terms of variance (as shown by the colored areas on the plot). Most notably, the I2DQN method suffers from high variance in performance when traffic flows start to increase (as seen in subplot 1) and features an extremely poor simulation episode near the peak of traffic flows (as seen on subplot 2), probably due to a single disastrous simulation episode, showing the potential instability of the I2DQN algorithm under saturated traffic flows. More importantly, these simulation results do not showcase the inefficiency of the I2DQN method, whose results are similar to those found in Figure 9.10, but rather the extremely good resilience of the DEC-DQN algorithm even under highly saturated traffic conditions.

These simulation results seem like definitive proof that, when properly orchestrated, coordination between intersections for traffic signal control can significatively increase agent performance and globally reduce cumulative waiting times over a road network. Our results have shown that, in the case of our experimental protocol, direct coordination is likely to be superior to indirect coordination since information transmitted by neighboring intersections is likely to influence the value estimates of both the "switch" and "extend" actions, as opposed to the latter only in MARLIN-type algorithms. Since these observations have been made, however, on relatively small road networks containing a few intersections, we address the direct comparison of the three main methods of interest (I2DQN, deep MARLIN, and DEC-DQN) on a synthetic large-scale road network in the next and final traffic simulation of this thesis.

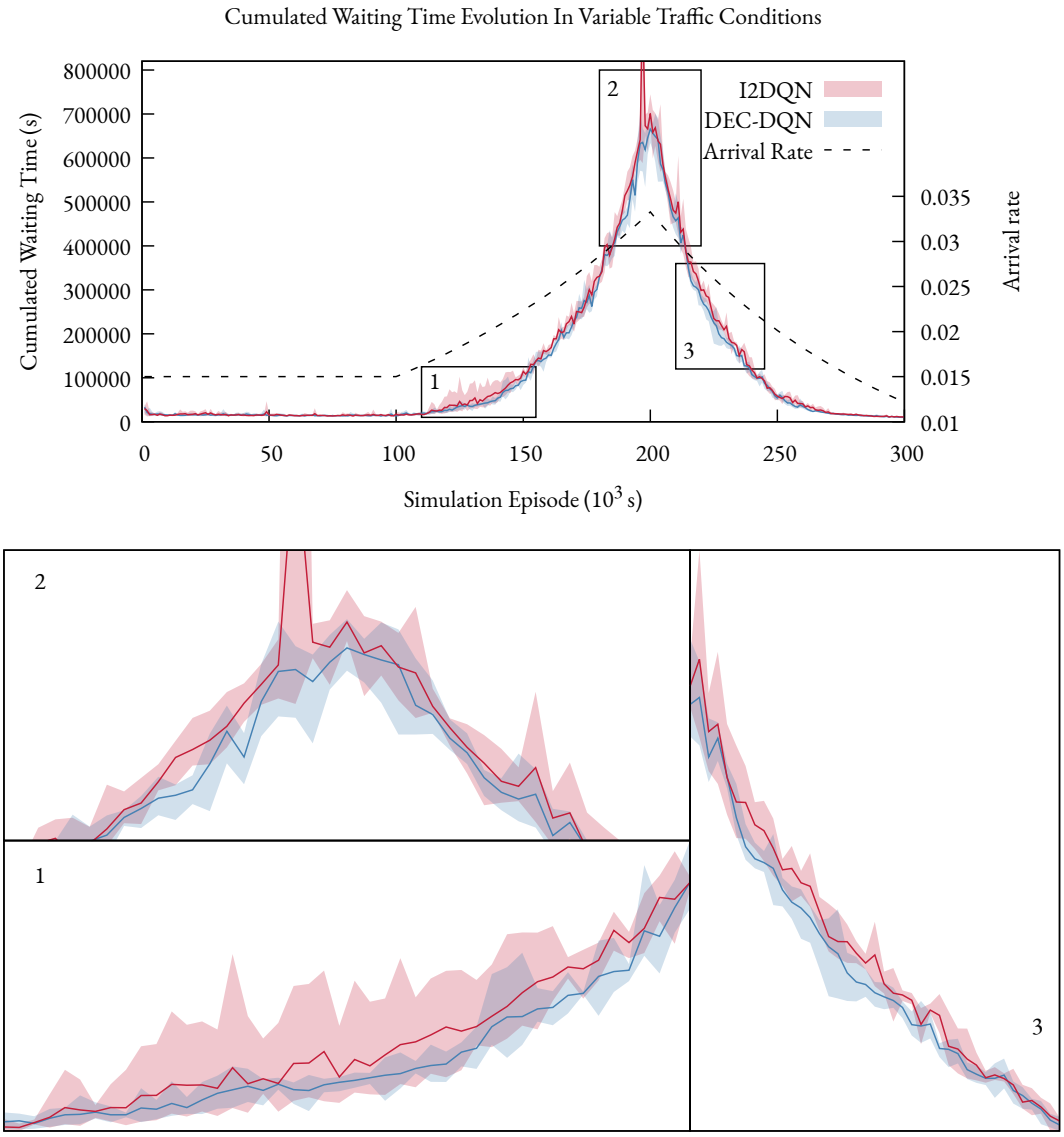

Figure 9.14: Analysis of the I2DQN and DEC-DQN algorithms in variable traffic conditions.

9.5 Agent Coordination on Large-Scale Traffic Networks

While the results observed in the previous section and their subsequent conclusions are sufficient to prove that agent coordination is effective in increasing traffic optimization performance through message passing, the effect of the direct and indirect coordination methods have been tested on road networks featuring a limited amount of intersections. This last section, which contains experiments on coordination methods on a larger scale of operation, has two main objectives. First, it aims to analyze whether such a change in scale has any effect on the coordination mechanisms that were previously observed. In other words, we want to know if increasing the number of coordinated intersections changes the behavior of the independent, indirect, and direct coordination methods we have used so far. Second, this experiment on a larger-scale network is used to directly compare the three most promising RL-TSC methods we have tested so far: I2DQN, deep MARLIN, and DEC-DQN, in order to draw more decisive conclusions regarding their respective merits and shortcomings.

9.5.1 Synthetic Large-Scale Road Network

The network chosen to represent a large-scale traffic simulation is composed of 77 nodes (57 of which are intersections controlled by traffic lights) and 240 edges. While not based on the network graph of a real-world urban area, this road network aims to recreate features commonly seen in urban areas, such as the use of a two-by-two lane outer ring road and north/west and east/west arterials going through the network center.

In order to generate realistic traffic flows over the network, the default shortest path algorithm used by vehicles to select a route on the network (through the `duarouter` program) is modified in two ways. First, the edge length used in the computation of the shortest part is weighted by a factor of the number of lanes the road has: for two routes with the same weight, a vehicle will take the route containing edges with a higher number of lanes, which is akin to preferring arterials and highways instead of single-lane streets. Second, a random factor r is introduced in `duarouter` and represents the upper bound for which sub-optimal routes can still be chosen by a vehicle. For a parameter $r = 1.2$ (which we pick), a vehicle can select any route going from its origin to its destination as long as its travel cost is at most 1.20 times the cost of the shortest route. This factor introduces randomness and personal driver preferences (which might be sub-optimal) in travel route selection. Finally, and similarly to the previous sections, the Poisson vehicle arrival rate of each origin-destination pair of the network is chosen according to an exponential law of parameter λ, according to the experiment protocol defined in section 7.2.2. The convergence process of the RL-TSC methods are first tested on a non-saturation arrival rate, and their stability is then tested by increasing and decreasing this arrival rate.

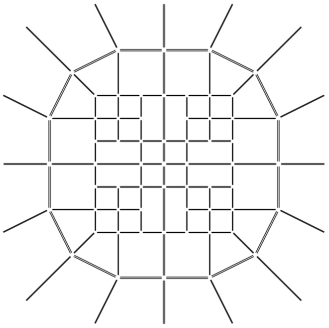

Figure 9.15: Synthetic city network.

9.5.2 Performance Under Fixed and Variable Arrival Rates

After describing the road network over which our experiments take place, we now observe the effect of scaling the experimental road network up on the convergence process and adaptability to changing traffic conditions of the I2DQN, deep MARLIN, and DEC-DQN traffic signal control methods. Given their overall scale, and given the fact that they compare the best RL-TSC methods we have developed in each category (i.e., independent, indirect, and direct coordination), this final series of experiments give a complete overview of the respective merits and shortcomings of these RL-TSC methods.

9.5.2.1 Convergence Under Fixed Traffic Flows

As seen in the previous section of this chapter, the first manner in which coordinated RL-TSC methods can be analyzed is through the observation of their convergence process according to the protocol defined in section 7.3.1. This convergence is tested on the synthetic city network for a near-saturation fixed arrival rate of $\lambda = 0.1$. For this experiment, the performance spectrum of each method over the ten tested traffic scenarios (defined in section 7.3.3 as the area showing the best and worst cumulated waiting times of a given method over the tested traffic scenarios) is displayed alongside the usual average performance plot.

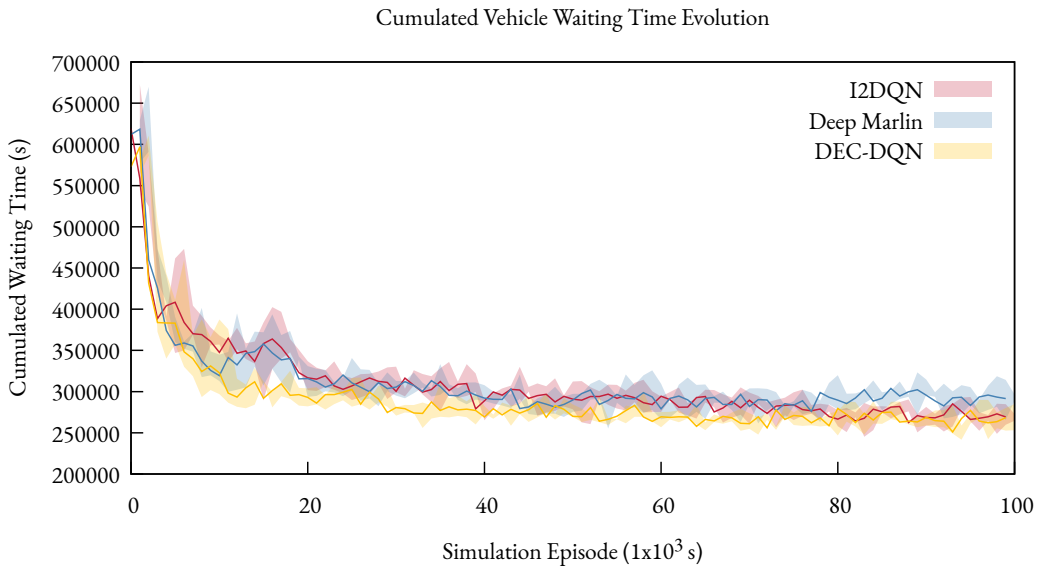

Figure 9.16: Convergence of the I2DQN, deep MARLIN and DEC-DQN with a constant arrival rate on the synthetic city network.

The convergence values of Figure 9.16 show results similar to those observed in small-scale networks. Most notably, we observe that the DEC-DQN coordination method (in yellow in the figure) converges around 10% faster than other methods in the early learning iterations and maintains its performance advantage throughout the simulation runs. Another notable point regard-

ing these simulation results is the respective evolution of the I2DQN and deep MARLIN methods compared to the previous small-scale experiments. While converging slower at first, I2DQN (in red) ends up performing better than deep MARLIN (in blue) on average towards the end of the simulation, reaching DEC-DQN-like performance levels.

The convergence analysis of these three RL-TSC methods hence provide valuable information regarding their comparative merits on large-scale traffic simulations with regards to their convergence. Similarly to earlier experiments, the DEC-DQN coordinated method provides the best of both worlds by performing better on early simulation iterations and by maintaining these superior performances in the long run. Hence, the introduction of a large number of learning agents, each using the same neural network in order to learn to communicate, seems to accelerate their convergence towards a common communication protocol, explaining the early convergence of DEC-DQN due to efficient coordination through proper communication. Conversely, the deep MARLIN algorithm seems to suffer from the introduction of a larger number of learning agents. While the deep MARLIN method yielded similar performance levels with lower variance to those of I2DQN on simulations done on a smaller scale network (see Figure 9.9), the switch to a large-scale network decreases the maximum performance metrics of the deep MARLIN algorithm, which becomes strictly less efficient than the I2DQN method. The reason for this loss in efficiency is not entirely apparent since coordination, in the deep MARLIN case, still yields benefits in the form of accelerated convergence in early iterations when compared to I2DQN.

9.5.2.2 Performance Under Variable Traffic Flows

The final simulation scenario we run in order to comprehensively analyze the three coordinated RL-TSC methods compares their respective performance under variable traffic conditions in the large-scale synthetic road network defined in section 9.5.1. This analysis, coupled with the results of the fixed traffic flows of the previous subsection, should provide a complete comparison of these three methods. This last simulation scenario is by far the most costly to run from a computational perspective since it features both a high number of intersections and a high number of vehicles when traffic demand is increased (see section 9.5.2.3 for the complexity analysis of each method). Consequently, we used five different traffic scenarios to generate the data used to plot Figure 9.17, instead of the usual 10.

Results of this final experiment, as shown on Figure 9.17, strengthen the observations we made under fixed traffic flows. First, the deep MARLIN method (in blue in the figure) is the worst average performer as traffic flows increase (see subplot 1), similarly to what was observed towards the end of traffic scenarios of Figure 9.16. As it also features low-performance variance, the deep MARLIN method seems to indicate its limitations once again when applied to many intersections. However, the somewhat decentralized architecture of the deep MARLIN method (since each neighboring intersection pair is associated with an independent neural network) seems to increase its robustness, as demonstrated by the low variance of its performances once traffic demand decreases (see subplot 3). These results indicate that the deep MARLIN method is best suited in relatively small networks with variable traffic conditions, such as seen in the traffic scenario of section 9.3.2.1. Comparatively, the I2DQN method (in red) features relatively good performances–superior on average to those of deep MARLIN–although at the cost of higher variance and lower robustness (see subplot 3). These results underline the fact that the I2DQN method is suitable

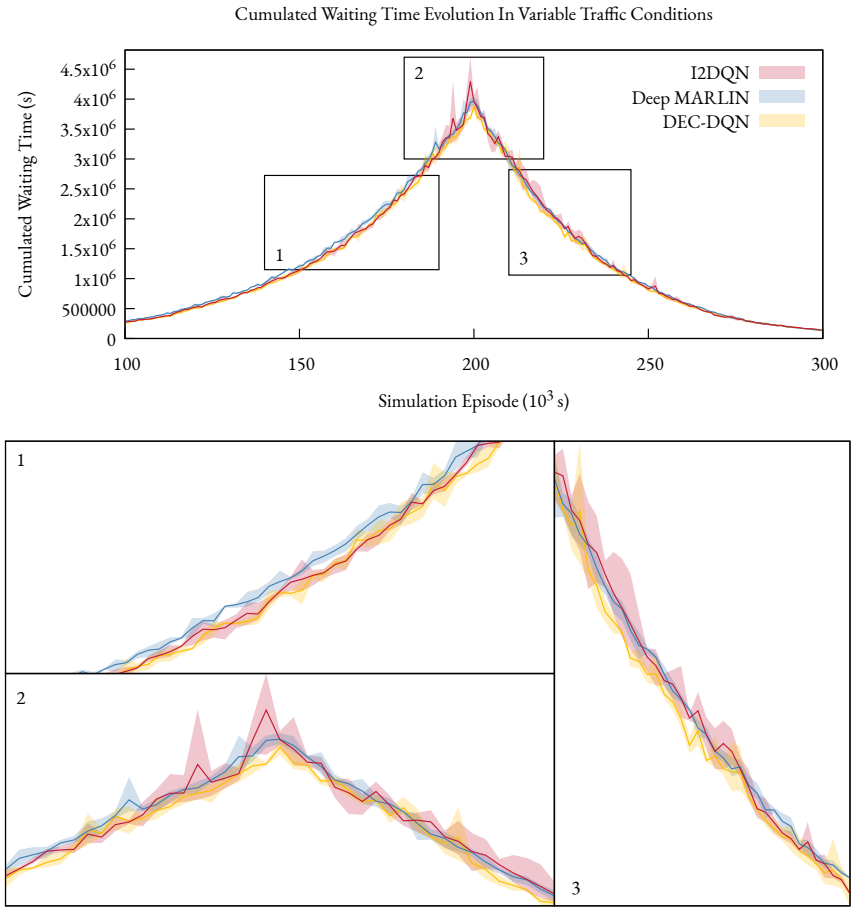

Figure 9.17: Comparison of the I2DQN, deep MARLIN and DEC-DQN methods in variable traffic conditions on a large scale traffic network.

for any road network size (since no coordination is used) but is preferable in networks with relatively low traffic demand variation. Finally, our novel DEC-DQN method provides once again the best of both worlds by featuring both the best average performance levels for almost all points of the simulation episodes, all while featuring low variance levels. Such a method is preferable for all types of networks and all types of traffic demand, provided that direct coordination between intersections is possible.

9.5.2.3 Complexity Comparison of RL-TSC Algorithms

A final point of comparison worth analyzing regarding coordinated RL-TSC methods relates to their computational and memory requirements. Indeed, while these constraints do not play a significant role in the desirability of each method in simulated scenarios, they can be major factors in their feasibility in real-world applications. We compare two principal metrics for each method. First, we look at the average number of operations executed by each algorithm for each simulation step, which broadly represents their average computational needs. The average number of operations is preferred to a traditional complexity analysis as it gives a more detailed appreciation of each method's relative computational costs. The complete calculations used to obtain this average number of operation per simulation step is given in Appendix C. The second metric we use is the average memory requirements for learning data, which can directly be obtained by measuring the file size of the learning data file we obtain at the end of a traffic simulation using `carmulator`.

Method	Computation (op/s)	Memory (kb)
I2DQN	239162880	409144
Deep MARLIN	717663744	2069528
DEC-DQN	1435765248	459716

Table 9.4: Computational and memory requirements of various RL-TSC algorithms.

Computational and memory requirements shown on Table 9.4 underline a typical pattern related to coordination for traffic signal control methods: there are strongly diminishing returns when aiming at improving a well-parameterized independent TSC method. The limited to notable improvements regarding agent convergence or performance, in the respective case of deep MARLIN and DEC-DQN, are either associated with a steep increase in memory (a fivefold increase) or computational (a sevenfold increase) requirements, respectively. This last observation underlines the high costs associated with implementing effective agent coordination for traffic signal control, whether in terms of the general complexity of the algorithms at play or in their associated costs. Nonetheless, given the speed at which all these algorithms are executed in simulated traffic conditions, these costs should not be prohibitive regarding their potential application in real-world traffic scenarios.

This chapter undertook an in-depth analysis of various RL-TSC methods used on road networks featuring multiple intersections. More specifically, we analyzed different modes of coordination between agents to establish which form is most beneficial for traffic signal control. Each of

these coordination methods is compared to an independent method, I2DQN, which was identified as the most capable non-coordinated RL-TSC method at the end of chapter 8.

We started by defining coordination based on the concept of green wave coordination from traffic engineering. In its adaptive form, the green wave coordination method adds offset constraints on intersections along an arterial. Each intersection has a fixed total cycle time and hence chooses phase-based actions to route traffic. On this basis, the green wave coordination method uses an 2DQN neural network to learn how to split green phase time within this fixed signal cycle. Experimental results have shown that this novel method significantly outperformed the independent I2DQN controller on arterial streets under normal traffic conditions. However, this performance hierarchy entirely inverts when traffic is saturated along the arterial, making the green wave coordination method unfit for general use on road networks.

The second type of coordination mechanism we analyzed is indirect coordination, which relies on indirect information passing between neighboring intersections. We presented the MARLIN-IC algorithm, which is a trademark method of the coordinated RL-TSC literature. This method relies on joint state-action modeling in order to compute optimal action choices for each agent of the network. We proposed a modified version of the MARLIN-IC algorithm featuring an additional single-agent decision-making process and used a deep neural network instead of a classical Q-learning algorithm for agent learning. This adapted method has shown moderate improvements compared to the I2DQN algorithm. An in-depth analysis of the neural networks used by agents of the MARLIN-IC algorithm has shown that these improvements are held back by the fact that joint state-action modeling is not an efficient manner of communicating information between neighboring intersections, since the information being sent from an intersection does not immediately affect the neighboring intersection's traffic.

The third type of coordination mechanism we analyzed is direct coordination, which lets agents directly exchange information to maximize their reward. We developed a novel coordinated RL-TSC method, DEC-DQN, based on the coordination mechanisms of the DIAL algorithm. This algorithm enables agents working towards an optimization goal to communicate in order to solve their tasks. The novelty of this algorithm is that agents do not only use reinforcement learning to learn how to solve their tasks but also to settle on a common coordination protocol. This novel approach is highly desirable in RL-TSC applications since this communication does not have to be designed beforehand and can hence be potentially applied to any type of road network. Experiments conducted on small-scale networks have shown that DEC-DQN, which we adapted for traffic signal control, performed significantly better than the baseline I2DQN method.

The final section of this chapter pitted the three best RL-TSC methods designed during this thesis work in a large-scale simulation scenario. These experiments were carried out in order to identify the relative strengths and weaknesses of each type of coordination: independent in the case of I2DQN, indirect for deep MARLIN, and direct for DEC-DQN. Simulation results have shown that the performance of the deep MARLIN algorithm was degraded due to the scale of the road network but could adapt well to varying traffic demand, implying that this method is better suited for small-scale networks with variable demand. Conversely, the I2DQN method featured a performance level similar to those of deep MARLIN, although at a much lower complexity cost. However, its low robustness makes the I2DQN algorithm better-suited for road networks with low traffic demand variance. Finally, the DEC-DQN RL-TSC method once again displayed excellent convergence speed and great robustness to changing traffic conditions. Its ability to deal

with varying traffic demand regardless of network size makes it, in our opinion, the best RL-TSC method featured in this thesis work, and our recommended coordinated RL-TSC method, provided that direct agent coordination is feasible in the traffic optimization problem at hand.

10 Conclusion

The central aim of this thesis was to develop a state-of-the-art coordinated traffic signal control method for traffic optimization. Our objective was to carefully develop this method from the ground up to justify all of our model choices and provide guidelines for future research in this area.

Research Work of This Thesis

We carried out this research goal in iterative stages. The first step was to give a general presentation of what traffic optimization using reinforcement learning *is*. We separately presented the fields of traffic engineering and reinforcement learning in chapter 2 and chapter 3 respectively. These chapters introduced the main concepts and terminology of traffic signal control and gave a general overview of how reinforcement learning algorithms aim at solving a task through learning using an algorithm and policy on a Markov decision process. This necessary concept introduction paved the way for a presentation on how these two fields merged in the reinforcement learning for traffic signal control literature in chapter 4. This chapter gave a comprehensive overview of how reinforcement learning algorithms optimize traffic flows. Moreover, it introduced crucial concepts such as agent coordination and function approximation and explained how the literature tackled these challenges. These three chapters form the first part of the thesis.

We then developed the general framework in which our novel RL-TSC method could later be constructed, tested, and validated. We first defined a model of traffic flow in chapter 5, which allowed us to mathematically define traffic engineering concepts to apply them in a reinforcement learning context. Such an application takes place in chapter 6, which analyzes the multiple ways in which the traffic optimization problem can be modelized as a Markov decision process and establishes the optimal one, which we later use for our research work. This chapter notably underlines inefficient MDP models regarding state or action space representation. Finally, chapter 7 closed this general RL-TSC framework presentation by describing the traffic simulator used to develop and compare traffic signal control methods to answer our research question. This framework is composed of the `SUMO` traffic simulator and our research library, `carmulator`. The chapter ended with an in-depth presentation on *how* the performance of a RL-TSC controller could be accurately measured: through convergence and performance analysis protocols under fixed or variable traffic flows.

After establishing the concepts and framework needed for our research, we tackled the traffic optimization problem on isolated intersections in chapter 8. This chapter first presented a near-optimal fixed traffic signal control method used for benchmarking purposes before comparing three types of classic RL algorithms. After proving that temporal-difference learning coupled with greedy policies are most efficient for traffic optimization, we underlined the crucial role of function approximation for acceptable performance. This observation led to the analysis of vari-

ous modern deep reinforcement learning algorithms applied to TSC, singling out I2DQN as the best method for isolated traffic signal control. These findings enabled us to extend our research to multi-intersection road networks in chapter 9. We distinguished four distinct modes of agent coordination (independent, green wave, indirect and direct) and developed and tested a novel RL-TSC algorithm for each of these categories. We first developed a deep learning-based coordination method based on green waves over arterials, which performed better than independent learning under rather specific conditions. We then adapted a major algorithm of the RL-TSC literature MARLIN-IC in order to compare it to other forms of coordination. By slightly modifying its structure and using a neural network instead of a classical RL algorithm, we made the MARLIN-IC method outperform the I2DQN algorithm over a small-scale road network. Finally, and perhaps most notably, we developed the DEC-DQN direct coordination method, which allows intersections of a road network to coordinate through direct message passing. The novelty of this method is that no communication protocol is defined beforehand, meaning that intersections learn to both route traffic and settle on a common communication protocol. Our experiments have shown that the DEC-DQN direct coordination method outperforms all other tested methods and state-of-the-art performance levels. Moreover, these results have been confirmed on a large-scale simulated traffic network featuring more than 50 intersections.

Findings and Contributions

Our research has produced several significant contributions regarding RL-TSC, all of which are listed in section 1.4. Furthermore, our analysis work on isolated and coordinated TSC allowed us to formulate multiple key observations.

We tried to the best of our ability to not only show on experimental results that agent coordination was beneficial for RL-TSC but to explain *how* it improved traffic optimization tasks. This endeavor was, for instance, at the basis of the creation of the green wave coordination method since traffic engineering could formally prove that such a form of coordination form could improve throughput alongside an arterial. As RL methods are by nature much harder to formally analyze, our aim was to at least identify which parts of a coordination method made it superior to the baseline independent RL-TSC method. These attempts have resulted in the analysis of section 9.3.2.3 stating that the deep MARLIN algorithm could only partially influence its neighbors. This observation has, in turn, prompted us to adapt the DIAL algorithm into DEC-DQN due to its unique structure. Instead of trying to impose an explicit model of agent coordination, such as joint state-action modeling in the case of deep MARLIN, the ability of agents using DEC-DQN to learn a common communication protocol meant that such a model was no longer necessary. This difference is, in our eyes, quite similar to the model-based and model-free distinction in RL models as described in section 3.1.2.5. Instead of trying to create an imperfect model of the unknown mechanisms of traffic coordination, we could let a learning algorithm figure it for itself. The experiment conducted in paragraph 9.4.1.5 confirms that such a coordination mechanism entirely influences its neighbors through the unique reward function for communication actions we designed in section 9.4.1.4. This is, in our opinion, the strongest result of this thesis work, alongside the performance gains associated with using such a method.

The second major finding of this thesis work relates to our original goal of finding the "best" algorithm for RL-TSC. As section 9.5.2.2 as shown, this initial goal might have been wrongly

formulated given that each tested method has relative strengths and weaknesses. For instance, the I2DQN algorithm showcases good results overall (except in highly saturated conditions), on different network types, for an overall low complexity cost. Conversely, the DEC-DQN algorithm provides excellent results, even in highly congested scenarios, but comes at a high complexity cost and necessitates a specific infrastructure (i.e., a central controller being able to train agents) which might not be feasible in some situations. Hence, we would argue that each method presented in this thesis has specific areas in which their application would be relevant.

Finally, while experimental results of section 9.5.2.2 have shown that coordination is undeniably beneficial to traffic optimization, we were surprised to find out that modern non-coordinated RL-TSC algorithms such as I2DQN provided excellent performance for low complexity and computational costs. Indeed, we have established in section 9.5.2.2 that it is preferable to use an isolated algorithm such as I2DQN on a large-scale network instead of a more complex and variable algorithm such as deep MARLIN. More generally, this thesis work has shown the diminishing returns that are strongly associated with traffic optimization, as increases in method complexity yield smaller and smaller performance gains. This observation stands both for the traditional TSC methods studied in chapter 2 and for the RL-based methods of chapter 9.

Future Works

Even though we believe that this thesis offers both a broad and in-depth analysis of traffic optimization using reinforcement learning, we also believe that it could benefit from additional research and experimentation.

Some short-term experiments could be carried out regarding the large-scale network simulations using real-world data. While the large-scale network we designed in section 9.5.1 has interesting properties such as an outer ring road and high-speed lanes making its analysis worthwhile, using real-world traffic data and the associated urban network could legitimate our simulation results even further. While some open data sets containing traffic flows exist[1], selecting data that is compatible with our simulation settings, reworking the datasets, and recreating the associated road networks in `SUMO` could not be achieved during this thesis work. However, such a pursuit would be worthwhile, in our opinion. Another short-term research question of interest would be the use of modern actor-critic methods using deep reinforcement learning for traffic signal control. We mentioned in section 4.2.4.2 that deep Q-networks and deep actor-critic methods both were popular options for advanced RL-TSC controllers. While the results of section 8.2.4 led us to study the former over the latter, recent multi-agent actor-critic approaches featuring multi-agent policy training (Lowe et al., 2017) or cooperative exploration of the state-action space by agent (Christianos et al., 2020) should be investigated.

Finally, we believe that the communication protocol learning process of the DEC-DQN algorithm opens fascinating research questions from a machine learning perspective. More specifically, the common communication protocol reached by DEC-DQN agents should be investigated. For instance, we wonder which features of the state space are leveraged by intersections for communication. Do they communicate their signal cycle properties, congestion data, or other features? This study could be conducted on alternative state spaces featuring more features of the true traffic environment states since such an analysis could teach us which features of the

[1] For instance, the the open traffic collection, lists various open data sets of traffic demand data.

environment are essential to communicate to neighboring intersections. Furthermore, we wonder whether the common communication protocol of agents changes depending on the network topology and traffic demand levels. Do agents settle on a similar protocol each time, or is it network or simulation-dependent? Additionally, we wonder whether intersections settle on human-understandable communication settings, corresponding to clear indications such as "traffic is saturated on my lanes" or whether these messages are simply designed to maximize neighboring intersections' expected rewards. Overall, we believe that the DEC-DQN algorithm opens a new set of research questions that we want to investigate further.

Bibliography

1. Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. "Traffic light control in non-stationary environments based on multi agent Q-learning". In: *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. IEEE. 2011, pp. 1580–1585.
2. Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. "Reinforcement learning for true adaptive traffic signal control". *Journal of Transportation Engineering* 129:3, 2003, pp. 278–285.
3. James S Albus. "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)". *Journal of Dynamic Systems, Measurement, and Control* 97:3, 1975, pp. 220–227.
4. Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.
5. Sahar Araghi et al. "Q-learning method for controlling traffic signal phase time in a single intersection". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE. 2013, pp. 1261–1265.
6. Román Aragon-Gómez and Julio B Clempner. "Traffic-signal control reinforcement learning approach for continuous-time markov games". *Engineering Applications of Artificial Intelligence* 89, 2020, p. 103415.
7. Itamar Arel et al. "Reinforcement learning-based multi-agent system for network traffic signal control". *IET Intelligent Transport Systems* 4:2, 2010, pp. 128–135.
8. Kai Arulkumaran et al. "A brief survey of deep reinforcement learning". *arXiv preprint arXiv:1708.05866*, 2017.
9. Mohammad Aslani, Mohammad Saadi Mesgari, and Marco 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, 2017, pp. 732–752.
10. Bram Bakker et al. "Traffic light control by multiagent reinforcement learning systems". In: *Interactive Collaborative Information Systems*. Springer, 2010, pp. 475–510.
11. Andrew G Barto. "Reinforcement learning and dynamic programming". In: *Analysis, Design and Evaluation of Man–Machine Systems 1995*. Elsevier, 1995, pp. 407–412.
12. Ana LC Bazzan. "Opportunities for multiagent systems and multiagent reinforcement learning in traffic control". *Autonomous Agents and Multi-Agent Systems* 18:3, 2009, p. 342.
13. Ana LC Bazzan, Denise De Oliveira, and Bruno C da Silva. "Learning in groups of traffic signals". *Engineering Applications of Artificial Intelligence* 23:4, 2010, pp. 560–568.

14. Richard Bellman. "A Markovian decision process". *Journal of mathematics and mechanics*, 1957, pp. 679–684.
15. Claude Berge. "Graphs and hypergraphs", 1973.
16. Daniel Bienstock, Gonzalo Muñoz, and Sebastian Pokutta. "Principled deep neural network training through linear programming". *arXiv preprint arXiv:1810.03218*, 2018.
17. Tim Brys, Tong T Pham, and Matthew E Taylor. "Distributed learning and multi-objectivity in traffic light control". *Connection Science* 26:1, 2014, pp. 65–83.
18. Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A comprehensive survey of multiagent reinforcement learning". *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38:2, 2008, pp. 156–172.
19. Vinny Cahill et al. "Soilse: A decentralized approach to optimization of fluctuating urban traffic using reinforcement learning". In: *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*. IEEE. 2010, pp. 531–538.
20. Jeancarlo Arguello Calvo and Ivana Dusparic. "Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control." In: *AICS*. 2018, pp. 2–13.
21. Yanan J Cao et al. "Design of a traffic junction controller using classifier system and fuzzy logic". In: *International Conference on Computational Intelligence*. Springer. 1999, pp. 342–353.
22. Edmond CP Chang et al. "MAXBAND-86: Program for optimizing left-turn phase sequence in multiarterial closed networks". *Transportation Research Record* 1181, 1988.
23. Chacha Chen et al. "Toward A thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3414–3421.
24. Yit Kwong Chin et al. "Q-learning based traffic optimization in management of signal timing plan". *International Journal of Simulation, Systems, Science & Technology* 12:3, 2011, pp. 29–35.
25. Debashish Chowdhury, Ludger Santen, and Andreas Schadschneider. "Statistical physics of vehicular traffic and some related systems". *Physics Reports* 329:4-6, 2000, pp. 199–329.
26. Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. "Shared experience actor-critic for multi-agent reinforcement learning". *Advances in Neural Information Processing Systems* 33, 2020, pp. 10707–10717.
27. Tianshu Chu, Sandeep Chinchali, and Sachin Katti. "Multi-agent Reinforcement Learning for Networked System Control". In: *International Conference on Learning Representations*. 2020. URL: `https://openreview.net/forum?id=Syx7A3NFvH`.
28. Tianshu Chu et al. "Multi-agent deep reinforcement learning for large-scale traffic signal control". *IEEE Transactions on Intelligent Transportation Systems*, 2019.
29. Li Chun-Gui et al. "Urban traffic signal learning control using fuzzy actor-critic methods". In: *2009 Fifth International Conference on Natural Computation*. Vol. 1. IEEE. 2009, pp. 368–372.

30. Robert H Crites and Andrew G Barto. "An actor/critic algorithm that is equivalent to Q-learning". In: *Advances in Neural Information Processing Systems*. 1995, pp. 401–408.

31. Stacy Davis and Robert Gary Boundy. *Transportation Energy Data Book: Edition 39*. Tech. rep. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2021.

32. Samah El-Tantawy and Baher Abdulhai. "An agent-based learning towards decentralized and coordinated traffic signal control". In: *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*. IEEE. 2010, pp. 665–670.

33. Samah El-Tantawy and Baher Abdulhai. "Multi-agent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC)". In: *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*. IEEE. 2012, pp. 319–326.

34. Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC): methodology and large-scale application on downtown Toronto". *IEEE Transactions on Intelligent Transportation Systems* 14:3, 2013, pp. 1140–1150.

35. Jakob Erdmann. "SUMO's lane-changing model". In: *Modeling Mobility with Open Data*. Springer, 2015, pp. 105–123.

36. Jakob Erdmann and Daniel Krajzewicz. "SUMO's road intersection model". In: *Simulation of Urban MObility User Conference*. Springer. 2013, pp. 3–17.

37. Jakob Foerster et al. "Learning to communicate with deep multi-agent reinforcement learning". *Advances in neural information processing systems* 29, 2016.

38. Juntao Gao et al. "Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network". *arXiv preprint arXiv:1705.02755*, 2017.

39. Nathan H Gartner. *OPAC: A demand-responsive strategy for traffic signal control*. 906. 1983.

40. Nathan H Gartner, Farhad J Pooran, and Christina M Andrews. "Implementation of the OPAC adaptive control strategy in a traffic signal network". In: *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No. 01TH8585)*. IEEE. 2001, pp. 195–200.

41. Nathan H Gartner, Chronis Stamatiadis, and Philip J Tarnoff. "Development of advanced traffic signal control strategies for intelligent transportation systems: Multilevel design". *Transportation Research Record* 1494, 1995.

42. Nathan H Gartner et al. "A multi-band approach to arterial traffic signal optimization". *Transportation Research Part B: Methodological* 25:1, 1991, pp. 55–74.

43. Wade Genders. "Deep reinforcement learning adaptive traffic signal control". PhD thesis. 2018.

44. Wade Genders and Saiedeh Razavi. "Evaluating reinforcement learning state representations for adaptive traffic signal control". *Procedia computer science* 130, 2018, pp. 26–33.

45. Wade Genders and Saiedeh Razavi. "Using a deep reinforcement learning agent for traffic signal control". *arXiv preprint arXiv:1611.01142*, 2016.

46. Martin Gregurić et al. "Application of deep reinforcement learning in traffic signal control: An overview and impact of open traffic data". *Applied Sciences* 10:11, 2020, p. 4011.

47. Ivo Grondman et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42:6, 2012, pp. 1291–1307.

48. Hado Hasselt. "Double Q-learning". *Advances in neural information processing systems* 23, 2010, pp. 2613–2621.

49. Ammar Haydari and Yasin Yilmaz. "Deep reinforcement learning for intelligent transportation systems: A survey". *IEEE Transactions on Intelligent Transportation Systems*, 2020.

50. Jean-Jacques Henry, Jean Loup Farges, and J Tuffal. "The PRODYN real time traffic algorithm". In: *Control in Transportation Systems*. Elsevier, 1984, pp. 305–310.

51. Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

52. Duan Houli, Li Zhiheng, and Zhang Yi. "Multiobjective reinforcement learning for traffic signal control using vehicular ad hoc network". *EURASIP journal on advances in signal processing* 2010, 2010, p. 7.

53. Ronald A Howard. "Dynamic programming and markov processes.", 1960.

54. PB Hunt et al. *SCOOT-a traffic responsive method of coordinating signals*. Tech. rep. 1981.

55. Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

56. Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". *Journal of artificial intelligence research* 4, 1996, pp. 237–285.

57. Mohamed A Khamis and Walid Gomaa. "Adaptive multi-objective reinforcement learning with hybrid exploration for traffic signal control based on cooperative multi-agent framework". *Engineering Applications of Artificial Intelligence* 29, 2014, pp. 134–151.

58. Mohamed A Khamis, Walid Gomaa, and Hisham El-Shishiny. "Multi-objective traffic light control system based on Bayesian probability interpretation". In: *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*. IEEE. 2012, pp. 995–1000.

59. Mohamed A Khamis et al. "Adaptive traffic control system based on Bayesian probability interpretation". In: *Electronics, Communications and Computers (JEC-ECC), 2012 Japan-Egypt Conference on*. IEEE. 2012, pp. 151–156.

60. Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980*, 2014.

61. Jelle R Kok and Nikos Vlassis. "Using the max-plus algorithm for multiagent decision making in coordination graphs". In: *Robot Soccer World Cup*. Springer. 2005, pp. 1–12.

62. Peter Koonce and Lee Rodegerdts. *Traffic signal timing manual*. Tech. rep. United States. Federal Highway Administration, 2008.

63. Stefan Krauß. "Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics", 1998.

64. Lior Kuyer et al. "Multiagent reinforcement learning for urban traffic control using coordination graphs". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2008, pp. 656–671.

65. M Lambert. *Traffic Management In Urban Areas - The French Expertise*. 2017. URL: `https://www.tresor.economie.gouv.fr/Articles/ed92a1e7-6eb5-4518-8ac3-a54f1fe2a5fb/files/f15999c0-02d0-4c52-9bd3-7088c775462b` (visited on 01/03/2022).

66. Joel Z Leibo et al. "Multi-agent reinforcement learning in sequential social dilemmas". *arXiv preprint arXiv:1702.03037*, 2017.

67. Xiang Li et al. "Understanding the disharmony between dropout and batch normalization by variance shift". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2682–2690.

68. Xiaoyuan Liang et al. "A deep reinforcement learning network for traffic light cycle control". *IEEE Transactions on Vehicular Technology* 68:2, 2019, pp. 1243–1253.

69. John DC Little. "A proof for the queuing formula: L= λ W". *Operations research* 9:3, 1961, pp. 383–387.

70. John DC Little. "The synchronization of traffic signals by mixed-integer linear programming". *Operations Research* 14:4, 1966, pp. 568–594.

71. John DC Little, Mark D Kelson, and Nathan H Gartner. "MAXBAND: A versatile program for setting signals on arteries and triangular networks", 1981.

72. Junjia Liu et al. "Learning scalable multi-agent coordination by spatial differentiation for traffic signal control". *Engineering Applications of Artificial Intelligence* 100, 2021, p. 104165.

73. Pablo Alvarez Lopez et al. "Microscopic traffic simulation using sumo". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 2575–2582.

74. Ryan Lowe et al. "Multi-agent actor-critic for mixed cooperative-competitive environments". *Advances in neural information processing systems* 30, 2017.

75. Jinming Ma and Feng Wu. "Feudal multi-agent deep reinforcement learning for traffic signal control". In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2020, pp. 816–824.

76. Patrick Mannion, Jim Duggan, and Enda Howley. "An experimental review of reinforcement learning algorithms for adaptive traffic signal control". In: *Autonomic Road Transport Support Systems*. Springer, 2016, pp. 47–66.

77. Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. "Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning". In: *International Conference on Artificial Neural Networks*. Springer. 2006, pp. 840–849.

78. Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

79. Clay McShane. "The origins and globalization of traffic control signals". *Journal of Urban History* 25:3, 1999, pp. 379–404.

80. Juan C Medina and Rahim F Benekohal. "Traffic signal control using reinforcement learning and the max-plus algorithm as a coordinating strategy". In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*. IEEE. 2012, pp. 596–601.

81. Sadayoshi Mikami and Yukinori Kakazu. "Genetic reinforcement learning for cooperative traffic signal control". In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. IEEE. 1994, pp. 223–228.

82. Alan J. Miller. *A Computer control system for traffic networks*. University of Birmingham, Graduate School in Highway & Traffic Engineering Birmingham, England, 1963.

83. Pitu Mirchandani and Larry Head. "RHODES: A real-time traffic signal control system: architecture, algorithms, and analysis". *Transportation Research Part C: Emerging Technologies* 9:6, 2001, pp. 415–432.

84. Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". *nature* 518:7540, 2015, pp. 529–533.

85. Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. "Traffic light control using deep policy-gradient and value-function-based reinforcement learning". *IET Intelligent Transport Systems* 11:7, 2017, pp. 417–423.

86. Kai Nagel and Michael Schreckenberg. "A cellular automaton model for freeway traffic". *Journal de physique I* 2:12, 1992, pp. 2221–2229.

87. Ranjit Nair et al. "Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs". In: *AAAI*. Vol. 5. 2005, pp. 133–139.

88. Kumpati S Narendra and Mandayam AL Thathachar. "Learning automata-a survey". *IEEE Transactions on systems, man, and cybernetics* 4, 1974, pp. 323–334.

89. Tomoki Nishi et al. "Traffic signal control based on reinforcement learning with graph convolutional neural nets". In: *2018 21st International conference on intelligent transportation systems (ITSC)*. IEEE. 2018, pp. 877–883.

90. Mohammad Noaeen et al. "Reinforcement Learning in Urban Network Traffic Signal Control: A Systematic Literature Review", 2021.

91. Ann Nowé, Katja Verbeeck, and Maarten Peeters. "Learning automata as a basis for multi agent reinforcement learning". In: *International Workshop on Learning and Adaption in Multi-Agent Systems*. Springer. 2005, pp. 71–85.

92. Ann Nowé, Peter Vrancx, and Yann-Michaël De Hauwere. "Game theory and multi-agent reinforcement learning". In: *Reinforcement Learning*. Springer, 2012, pp. 441–470.

93. Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. "Optimal and approximate Q-value functions for decentralized POMDPs". *Journal of Artificial Intelligence Research* 32, 2008, pp. 289–353.

94. Norihiko Ono and Kenji Fukumoto. "A modular approach to multi-agent reinforcement learning". In: *Distributed Artificial Intelligence Meets Machine Learning Learning in Multi-Agent Environments*. Springer, 1997, pp. 25–39.

95. Liviu Panait and Sean Luke. "Cooperative multi-agent learning: The state of the art". *Autonomous agents and multi-agent systems* 11:3, 2005, pp. 387–434.

96. Markos Papageorgiou et al. "Review of road traffic control strategies". 91, 2004, pp. 2043–2067.

97. Tong Thanh Pham et al. "Learning coordinated traffic light control". In: *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS-13)*. Vol. 10. IEEE. 2013, pp. 1196–1201.

98. LA Prashanth and Shalabh Bhatnagar. "Reinforcement learning with function approximation for traffic signal control". *IEEE Transactions on Intelligent Transportation Systems* 12:2, 2011, pp. 412–421.

99. Hongsheng Qi et al. "Coordinated intersection signal design for mixed traffic flow of human-driven and connected and autonomous vehicles". *IEEE Access* 8, 2020, pp. 26067–26084.

100. Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions". *arXiv preprint arXiv:1710.05941*, 2017.

101. Silvia Richter, Douglas Aberdeen, and Jin Yu. "Natural actor-critic for road traffic optimisation". In: *Advances in neural information processing systems*. 2007, pp. 1169–1176.

102. Martin Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European conference on machine learning*. Springer. 2005, pp. 317–328.

103. Dennis I Robertson. "Research on the TRANSYT and SCOOT Methods of Signal Coordination". *ITE journal* 56:1, 1986, pp. 36–40.

104. Dennis I Robertson. "TRANSYT: a traffic network study tool", 1969.

105. Nagui Rouphail, Andrzej Tarko, and Jing Li. *Traffic flows at signalized intersections*. Transportation Research Board, 1998.

106. Gérard Scemama and Olivier Carles. "Claire-siti, public and road transport network management control: A unified approach", 2004.

107. Tom Schaul et al. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952*, 2015.

108. Soheil Mohamad Alizadeh Shabestary and Baher Abdulhai. "Deep learning vs. discrete reinforcement learning for adaptive traffic signal control". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 286–293.

109. Michael Shenoda, Randy Machemehl, et al. *Development of a phase-by-phase, arrival-based, delay-optimized adaptive traffic signal control methodology with metaheuristic search*. Tech. rep. Texas Transportation Institute, 2006.

110. Saijiang Shi and Feng Chen. "Deep Recurrent Q-learning Method for Area Traffic Coordination Control". *Journal of Advances in Mathematics and Computer Science*, 2018, pp. 1–11.

111. Chronis Stamatiadis and Nathan Gartner. "MULTIBAND-96: a program for variable-bandwidth progression optimization of multiarterial traffic networks". *Transportation Research Record: Journal of the Transportation Research Board* 1554, 1996, pp. 9–17.

112. Merlijn Steingrover et al. "Reinforcement Learning of Traffic Light Controllers Adapting to Traffic Congestion." In: *BNAIC*. Citeseer. 2005, pp. 216–223.

113. Andrew Sullivan et al. *Traffic Signal Design Guide & Timing Manual*. The University Transportation Center of Alabama. Alabama Department of Transportation, 2015.

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

115. Ardi Tampuu et al. "Multiagent cooperation and competition with deep reinforcement learning". *PloS one* 12:4, 2017, e0172395.

116. Ming Tan. "Multi-agent reinforcement learning: Independent vs. cooperative agents". In: *Proceedings of the tenth international conference on machine learning*. 1993, pp. 330–337.

117. Saad Touhbi et al. "Adaptive traffic signal control: Exploring reward definition for reinforcement learning". *Procedia Computer Science* 109, 2017, pp. 513–520.

118. Maxime Tréca, Julian Garbiso, and Dominique Barth. "Guidelines for Action Space Definition in Reinforcement Learning-Based Traffic Signal Control". In: *ICAPS Conference, Special Track on Planning and Learning*. ICAPS. AAAI, 2020.

119. Maxime Tréca et al. "Fast Bootstrapping for Reinforcement Learning-Based Traffic Signal Control Systems Using Queueing Theory". In: *2020 IEEE 92nd Vehicular Technology Conference: VTC2020*. VTC. IEEE, 2020.

120. Thomas Urbanik et al. *Signal timing manual*. Transportation Research Board, 2015.

121. Luis F Urquiza-Aguiar et al. "Comparison of traffic demand generation tools in SUMO: Case study: Access highways to quito". In: *Proceedings of the 16th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*. 2019, pp. 15–22.

122. US Department of Transportation USDT. *Every Day Counts brochure*. 2011. URL: `https://www.fhwa.dot.gov/innovation/everydaycounts/edc-1/pdf/asct_brochure.pdf` (visited on 11/30/2011).

123. Elise Van der Pol. "Deep reinforcement learning for coordination in traffic light control". *Master's thesis, University of Amsterdam*, 2016.

124. Elise Van der Pol and Frans A Oliehoek. "Coordinated deep reinforcement learners for traffic light control". *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.

125. Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

126. Simon Vanneste et al. "Learning to Communicate with Reinforcement Learning for an Adaptive Traffic Control System". In: *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. Springer. 2021, pp. 207–216.

127. Vladimir Vapnik and Rauf Izmailov. "Rethinking statistical learning theory: learning using statistical invariants". *Machine Learning* 108:3, 2019, pp. 381–423.

128. RA Vincent and JR Peirce. *'MOVA': Traffic Responsive, Self-optimising Signal Control for Isolated Intersections*. Tech. rep. 1988.

129. Peter Wagner et al. "Remarks on traffic signal coordination", 2019.

130. Song Wang et al. "Deep reinforcement learning-based traffic signal control using high-resolution event-based data". *Entropy* 21:8, 2019, p. 744.

131. Ziyu Wang et al. "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1995–2003.

132. Christopher JCH Watkins and Peter Dayan. "Q-learning". *Machine learning* 8:3-4, 1992, pp. 279–292.

133. Fo Vo Webster. *Traffic signal settings*. Tech. rep. 1958.

134. Hua Wei et al. "A survey on traffic signal control methods". *arXiv preprint arXiv:1904.08117*, 2019.

135. Hua Wei et al. "Colight: Learning network-level cooperation for traffic signal control". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 1913–1922.

136. Christopher W Wells. *Car country: An environmental history*. University of Washington Press, 2013.

137. Kaige Wen, Shiru Qu, and Yumei Zhang. "A stochastic adaptive control model for isolated intersections". In: *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2007, pp. 2256–2260.

138. MA Wiering. "Multi-agent reinforcement learning for traffic light control". In: *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*. 2000, pp. 1151–1158.

139. Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". *Machine learning* 8:3-4, 1992, pp. 229–256.

140. Yuanhao Xiong et al. "Learning traffic signal control from demonstrations". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 2289–2292.

141. Kok-Lim Alvin Yau et al. "A survey on reinforcement learning models and algorithms for traffic signal control". *ACM Computing Surveys (CSUR)* 50:3, 2017, p. 34.

142. Bao-Lin Ye et al. "A survey of model predictive control methods for traffic signal control". *IEEE/CAA Journal of Automatica Sinica* 6:3, 2019, pp. 623–640.

143. Jinghong Zeng, Jianming Hu, and Yi Zhang. "Adaptive traffic signal control with deep recurrent Q-learning". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2018, pp. 1215–1220.

144. Fangyu Zou et al. "A sufficient condition for convergences of adam and rmsprop". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11127–11135.

List of Figures

List of Tables

Acronyms

2DQN	Dueling Deep Q-network
3DQN	Dueling Double Deep Q-network
ATSC	Adaptive Traffic Signal Control
CMAC	Cerebellar Model Articulation Controller
DDQN	Double Deep Q-network
DEC-DQN	Direct-Evaluation Communication DQN
DIAL	Differentiable Inter-Agent Learning
DP	Dynamic Programming Methods
DQN	Deep Q-network
DRL	Deep Reinforcement Learning
DTSE	Discrete Traffic State Encoding
I2DQN	Independent Dueling Deep Q-network
LA	Learning Automata
LRI	Linear Reward-Inaction
LRP	Linear Reward-Penalty
MARL	Multi-Agent Reinforcement Learning
MC	Monte-Carlo Methods
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
RIAL	Reinforced Inter-Agent Learning
RL	Reinforcement Learning
RL-TSC	Reinforcement Learning-Based Traffic Signal Control
SARL	Single-Agent Reinforcement Learning
TD	Temporal-Difference Learning
TLS	Traffic Light System
TSC	Traffic Signal Control

List of Symbols

$\mathcal{F}$	Global objective function of a MDP
$\mathcal{S}$	State space of a MDP
$\mathcal{A}$	Action space of a MDP
$\mathcal{T}$	Transition function of a MDP
$\mathcal{R}$	Reward function of a MDP
t	Discrete time step
s, s_t	Environment state at step t
r, r_t	Agent reward at step t
a, a_t	Agent action at step t
π	Agent policy on a MDP
V	Value function of a state
Q	Quality function of a state-action pair
γ	Discount factor
α	Learning Rate
ε	Random action selection parameter
Ω	Observation space of a POMDP
O	Observation function of a POMDP
θ	Function approximation weights
$\mathcal{L}$	Loss function
G	Network Graph
V	Set of Vertices of G
A	Set of Arcs of G
u, v, w	Vertices of G
uv, vw	Arcs of G
p	Path made of arcs on G
$\Gamma(u), \Gamma^+(u), \Gamma^-(u)$	Neighborhood, successors and predecessors of vertex u
$\overline{uw}$	Traffic trajectory from uv to vw
ϕ_v	A green phase over vertex v
ϕ'_v	The amber phase associated with green phase ϕ_v
ϕ^0_v	The red phase on vertex v
Φ_v	A signal cycle on vertex v
$\phi_t(v)$	The active phase on intersection v at step t
$d_t(v)$	Amount of steps for which phase $\phi_t(v)$ has been active at step t
C_v	Total signal cycle duration of intersection v
$d_{\min}, d_{\max}$	Minimum/maximum duration of a traffic phase
T	Traffic transition function

$\langle p, e \rangle$	A vehicle following path p entering the network at step e
$\omega_t(c)$	Cumulated waiting time of vehicle c at time step t
$c_t(uv)$	Congestion of lane uv at step t
$P_t(uv)$	Position function of vehicles on arc uv at step t
λ	Exponential distribution parameter used for arrival rate generation
L_v	Lost time in the signal cycle of intersection v
f_l	Flow to saturation flow ratio of lane l
g_i	Allocated green time to phase i computed by Webster's formula
k	Strategy depth of the approximation method
h	Strategy application horizon of the approximation method
σ	Reward parameter of the linear reward-inaction algorithm
τ	Penatly parameter of the linear reward-inaction algorithm
$\delta, \hat{\delta}$	TD-error and TD-error estimate values
n	Moving window for the stopping criteria in simulations
κ	Performance delta for the stopping criteria in simulations
$\mathcal{D}$	Experience replay buffer
B	Minibatch size for deep learning methods
A	Advantage function on a given MDP
K	Target network update interval in steps
m	Message action of RIAL controller
Δ_t	Message reception delay of RIAL controller
r	Shortest route deviation factor

A Appendix - Optimal Method

This Appendix details the mode of operation of the optimal strategy approximation algorithm presented in section 8.1.2.

Algorithm 10: Pseudocode representation of the approximation algorithm on isolated intersections. Saving and loading simulation states is omitted for readability.

Function `Search`(t, S, ω)**:**
    if *Last element of* S *is* 0 **then**
        $t \leftarrow t + 1$;
        `SUMONextStep`();
        $\omega \leftarrow \omega +$ `SUMOGetWait`();
    else
        `SUMONextPhase`();
        for $i = 1$ **to** $min(d_{\min}, k - t) + 1$ **do**
            $t \leftarrow t + 1$;
            `SUMONextStep`();
            $\omega \leftarrow \omega +$ `SUMOGetWait`();
        `SUMONextPhase`();
        for $i = 1$ **to** $min(d_{\min}, k - t) + 1$ **do**
            $t \leftarrow t + 1$;
            `SUMONextStep`();
            $\omega \leftarrow \omega +$ `SUMOGetWait`();
    if $t \geq k$ **then**
        return S, ω;
    `Search`(t, $S + 0$, ω);
    `Search`(t, $S + 1$, ω);

The algorithm begins by verifying which action was taken last. If it was a phase extension, the simulation step and waiting time are updated, and the simulation moves one step forward in SUMO. If the last action was a phase switch, the algorithm simulates two successive phase switches (transition and beginning of the following green phase) of $d_{\min}$ steps each and update the waiting times accordingly. These phase switches can be cut short if the resulting simulation time is greater than the desired search depth k. If the resulting strategy is long enough, we return it alongside its associated waiting time. If not, we perform a new search split by recursively calling the function `Search` with both possible actions. Finally, once the algorithm has exhausted all possible strategies for the agent, it returns the strategy associated with the minimum vehicle waiting time.

B Appendix - Q-learning pre-estimation method

This appendix presents the function approximation method using state-action pair pre-estimation with phase-based actions as used in section 8.3.1. The elements relating to queueing theory and how the average service time per vehicle is computed can be found in our paper on the topic (Tréca et al., 2020b).

Algorithm 11: Pseudocode representation of the Q-learning pre-estimation algorithm.

for *each state* $s \in \mathcal{S}$ **do**
  for *each action* $a \in \mathcal{A}$ **do**
    $\tilde{Q}(s,a) \leftarrow 0$;
    for *each lane* $l \in L$ **do**
      $o_l \leftarrow c_l$ from $\mathcal{S}$;
      $n_l \leftarrow \lfloor \lambda_l \times a \rfloor$;
      if l *has a green light* **then**
        $o_l^- \leftarrow \min(\lfloor a/2 \rfloor, o_l)$;
        $n_l^- \leftarrow \min(\lfloor a/2 \rfloor - o_l^-, n_l)$;
      else
        $o_l^- \leftarrow 0$;
        $n_l^- \leftarrow 0$;
      $o_l^+ = o_l - o_l^-$;
      $n_l^+ = n_l - n_l^-$;
    $\tilde{Q}(s,a) \leftarrow \tilde{Q}(s,a) + T_l \times (n_l^- + o_l^-) - a \times (o_l^+ + n_l^+)$;

This algorithm enumerates the entire state-action couples around an isolated intersection. For each state and action, the algorithm computes, for each lane of the intersection, the number of vehicles already present in the lane (o_l, directly taken from state s), and an estimate of the number of new vehicles in the lane (n_l, estimated from the arrival rate on lane l, λ_l). Using these values, the algorithm estimates the number of vehicles to exit or stay on the lane for both of these categories, depending on whether the current lane l has a green signal in the current phase. Finally, the quality of action a in state s is estimated for the current lane by using Equation 8.6.

C Appendix - Complexity Analysis

This appendix estimates the average amount of operations needed to perform a single learning step for different coordinated RL-TSC methods. This metric is used instead of the traditional big-$\mathcal{O}$ complexity calculations since all algorithms are likely to appear in the same class ($\mathcal{O}(n)$), and we want to underline their computational requirements more precisely. This analysis does not take into account simple operations such as message passing, observation selection from the replay buffer or next action selection through a policy since they are negligible compared to the two main operations are required for agent learning: forward and backpropagation on a neural network. This estimation is simplified by the fact that the three tested algorithms all rely on the same neural network architecture showcased on Figure 8.8. If the number of intersections present on the road network, n, obviously plays a role in the average amount of learning operations per step, it should be noted that learning only occurs on time steps at which an agent picks a traffic-related action, meaning that the agent does not learn when it is in the amber, red, or minimal green phase. While the exact proportion of agents learning at each time step is dependent on the network type, traffic flows, the intersection's position within the network, and learning trajectory, we have observed from experimental data that an average of 50% of intersections choosing an action at each time step was a good general approximation for these calculations. It is hence possible, on this basis, to estimate the computational costs associated with the three main RL-TSC methods tested in section 9.5

I2DQN The simplest method to compute, I2DQN, only features a single learning step per agent each time it takes a traffic action. Since each neural network associated with an agent features an input layer of size $|\mathcal{S}|$, four fully connected hidden layers of size 128, and an output layer of size $|\mathcal{A}|$, each single forward and backpropagation on such a neural network each requires approximately $128 \times (|\mathcal{S}| + 128 + 128 + |\mathcal{A}|)$ operations, which are mostly due to matrix multiplications (Bienstock et al., 2018). Supposing that an average of half of the intersections present on the network perform these two operations at each time step, the associated average number of operations is equal to

$$\mathcal{C}_{I2DQN} = n/2 \times 2 \times 128 \times (2 \times 128^2 + |\mathcal{S}| + |\mathcal{A}|)$$

Since other operations associated with learning are negligible in comparison, this formula is a good approximation of the complexity of a single learning step on a single intersection using the base I2DQN method.

Deep MARLIN In comparison, the deep MARLIN method features the same exact local learning process for each network intersection but performs additional learning on joint state-action between each intersection of the network and its neighbors.

For g the average number of neighbors per intersection on a given network, each intersection performs a joint state-action learning task with an average of $g/2$ neighbors each step. Furthermore, the input and output sizes of these neural networks are doubled since they take into account local and neighboring states and actions, yielding a total amount of operations per step of

$$\begin{aligned}\mathcal{C}_{MARLIN} = n/2 \times 2 \times 128 \times (2 \times 128^2 + |\mathcal{S}| + |\mathcal{A}|) \\ + n/2 \times (g/2 \times 2 \times 128 \times (2 \times 128^2 + 2|\mathcal{S}| + 2|\mathcal{A}|))\end{aligned}$$

DEC-DQN Finally, the average number of operations per step for the DEC-DQN method can be split between traffic and communication-related actions. The traffic-related learning process is identical to the I2DQN case, with the exception that the state space is slightly increased since each neighbor also receives a communication action from its average g neighbors. Each agent chooses a communication at each time step regarding communication actions, regardless of their traffic action. However, backpropagation only occurs once a neighboring agent, having received an earlier communication action, receives it. Hence, each time an intersection chooses an action, it triggers g backpropagations (i.e., for each neighbor that sent a message to that intersection) on the shared communication neural network. Furthermore, each of these backpropagations requires to compute an associated reward, which is itself computed using forward propagation on the neural networks of each neighbor that sent the original message, as specified in algorithm 9.

$$\begin{aligned}\mathcal{C}_{DEC-DQN} = n/2 \times 2 \times 128 \times (2 \times 128^2 + |\mathcal{S}| + 4 + |\mathcal{A}|) \\ + n \times 128 \times (2 \times 128^2 + |\mathcal{S}| + |\mathcal{A}_m|) \\ + n/2 \times g \times 128 \times (2 \times 128^2 + |\mathcal{S}| + |\mathcal{A}_m|) \\ + n/2 \times g^2 \times 128 \times (2 \times 128^2 + |\mathcal{S}| + 4 + |\mathcal{A}|)\end{aligned}$$