UNIVERSIDADE DO VALE DO RIO DOS SINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FABIO MÜLLER VARISCO

REINFORCEMENT LEARNING-BASED TRAFFIC LIGHTS CONTROLLER WITH ADAPTIVE REWARD FUNCTION

São Leopoldo
2019

FABIO MÜLLER VARISCO

REINFORCEMENT LEARNING-BASED TRAFFIC LIGHTS CONTROLLER WITH ADAPTIVE REWARD FUNCTION

Article presented as partial requisite for obtaining a Bachelor of Computer Science degree, for the Course of Computer Science from Universidade do Vale do Rio dos Sinos (UNISINOS)

Advisor: Prof. Dr. Gabriel de Oliveira Ramos

São Leopoldo
2019

REINFORCEMENT LEARNING-BASED TRAFFIC LIGHTS CONTROLLER WITH ADAPTIVE REWARD FUNCTION

Fábio Müller Varisco[1]
Gabriel de Oliveira Ramos[2]

Abstract: Traffic Congestion affects the economy, the sustainability of the urban environment and the citizens' well-being. This problem can be alleviated by promoting more efficient usage of the road network through intelligent Traffic Control strategies. Reinforcement Learning-based Traffic Signal controllers offer many benefits in relation to other techniques, and one of them is the ability to tune the actions of the controller by customising its reward function. In this paper, we evaluate how different reward functions behave under different demand conditions and propose an Adaptive Reward Function that dynamically adapts its goal according to the road's saturation levels.

Keywords: Traffic Signal Controller. Q-learning. Adaptive Reward Function. Saturation Levels.

Resumo: O congestionamento no tráfego tem impactos na economia, na sustentabilidade das cidades e no bem-estar dos cidadãos. Este problema pode ser reduzido usando estratégias de controle de tráfego inteligentes para promover um uso mais eficiente da rede rodoviária. Controladores de semáforo baseados na técnica de Aprendizado por Reforço oferecem muitos benefícios em relação à outras técnicas, e um deles é ser capaz de ajustar as ações do controlador customizando a função de recompensa utilizada. Neste artigo, funções de recompensa são avaliadas sob diferentes condições de demanda e propomos uma função de recompensa adaptativa, que adapta seu objetivo dinamicamente de acordo com os níveis de saturação das vias.

Palavras-chave: Controlador de semáforos. Q-learning. Função de Recompensa Adaptativa. Níveis de saturação.

1 INTRODUCTION

Traffic Congestion heavily affects the economy, sustainability of the urban environment and the citizens' well-being (BORSCHETTE, 2018). McKinsey's report *Mobility's second great inflection point* mentions that traffic congestion costs countries 2 to 5 percent of national Gross Domestic Product (GDP), according to measures on lost time, wasted fuel and increased cost of doing business (DHAWAN et al., 2019). Despite being possible to alleviate the problem by providing additional capacity to the road network, many authors point that this approach can not indefinitely solve the problem and it may be a complete waste of resources at some portions of the day when the network is not heavily used. Therefore, traffic control strategies that lead to more efficient usage of the existing resources is a suitable alternative (PAPAGEORGIOU et al.,

[1]Undergraduate student in Computer Science from Unisinos. Email: fmvarisco@gmail.com
[2]Assistant Professor of the Graduate Program in Applied Computing of Unisinos. Email: gdoramos@unisinos.br

2003; VITI, 2006; REED; KIDD, 2019). According to Viti (2006), these strategies can improve the road network conditions by adapting dynamically the road capacity to demand or by better distributing the traffic demand throughout the network.

Traffic Control can be understood as a control loop consisting of a traffic network, that receive inputs such as traffic demand, surveillance mechanisms, and a control strategy, which can be manipulated by human operators or determined by an algorithm (PAPAGEORGIOU et al., 2003). It is defined from a control theory point-of-view by Gazis (1964), who says that traffic control has the goal of optimizing an objective function, which is to be achieved by controlling the parameters of the traffic system, respecting its constraints. Papageorgiou et al. (2003) mentions traffic lights at intersections as being one of the major control mechanisms in urban road networks. Methods for optimizing the traffic light schedules have been the topic of the study for a very long time. Initially, systems to optimize the traffic light schedules offline based on historical data were proposed and implemented. These are known as *Pre-timed controllers* and some examples are TRANSYT-7F and Synchro (GORDON et al., 2005). Then, *Responsive Control Systems* were proposed, which adapt the traffic signal schedules online in real-time to respond to changes in the current traffic situation, such as SCOOT and SCATS (GORDON et al., 2005). Finally, these evolved to *Adaptive Control Systems*, which also adapt the traffic signal timings in real-time and in an online fashion, but that work based on more advanced surveillance capabilities that are able to provide a detailed profile of traffic approaching an intersection. Some examples of this type of controller are RHODES and OPAC (GORDON et al., 2005).

In recent years, most of the research around *Adaptive Control Systems* attempt to solve the traffic signal control problem by using Reinforcement Learning (RL) techniques. This is mainly due to the benefits offered by this type of approach: RL-based controllers can quickly adapt to changes and continuously improve their performance by learning online; RL agents exhibit many autonomic and intelligent characteristics; and they are goal oriented, allowing to choose which system parameters we wish to optimise by customising the reward functions (MANNION; DUGGAN; HOWLEY, 2016). However, existing RL approaches fail to meet the different characteristics of under-saturated and over-saturated traffic conditions because they use the same reward function regardless of traffic saturation levels.

In this paper, we focus our research on how customising the reward function can help to tune the actions that the controller will take and how this could be useful to adapt the controller to different road conditions. First, we evaluate how different reward functions behave under different demand conditions. Then, based on the initial evaluation, we propose an Adaptive Reward Function that dynamically changes the reward function being used according to the road's saturation levels. To evaluate the reward functions and the proposed method, we execute simulations using SUMO, a microscopic traffic simulator. To execute the simulations, we developed a code architecture which was designed with flexibility and extensibility in mind. By applying object-oriented design patterns, we enable re-use, combination and replacement of its components, allowing rapid testing of new hypotheses and evaluation of new methods.

The main contributions of this work can be enumerated as follows: different Reward Functions taken from the literature (along with a few adaptations) are evaluated under different levels of demand; a method for dynamically changing the reward function based on road saturation level is proposed; and, finally, a simulation architecture that fosters modularization and re-use and allows rapid testing of new methods and hypotheses is proposed.

In Section 2, we start by introducing Traffic Signal Control methods and then discuss the advancements made by academic research since 2010 in Section 3. Next, in Section 4, we present the Q-learning method used for evaluation in terms of the typical RL structure (action definition, state representation, reward function and state selection), detailing the evaluated reward functions and the proposed adaptive reward function. The reward functions are then evaluated through a series of simulations executed using SUMO. The design of the simulation and the results of the experiments are presented in Section 5. Finally, we provide our conclusion and opportunities of future work in Section 6.

2 TRAFFIC SIGNAL CONTROL METHODS

Gordon et al. (2005) defines two main types of traffic signal controllers. The first is *Pre-timed Controller*, in which traffic signal plans are generally computed offline based on historical traffic data to determine the optimized cycle length, green time split and number and sequence of the phases. And the second is *Traffic-Actuated Control*, in which the traffic signal controllers adjust the traffic signal timings dynamically in response to traffic. This second classification may be further subdivided into *Traffic Responsive Control Systems* and *Traffic Adaptive Control Systems*.

Regarding Pre-timed Controllers, Gordon et al. (2005) explains that pre-timed signal plans can be computed manually by experts in the domain or by offline computer techniques, which generally optimize the timing plan based on some optimization criteria, such as a combination of number of stops and delay. TRANSYT-7F and Synchro are two widely used systems that implement this function. TRANSYT-7F is able to both evaluate an existing timing plan and also to create new optimized plans in order to minimize either a linear combination of delays, stops and queue spillback, or the total operating cost. Its newer version uses either hill-climbing or genetic algorithms as the optimization technique. TRANSYT-7F also has the possibility of optimizing the offsets, in which case the cycle times are fixed for all intersections. It has been extensively used in the U.S.. Synchro optimizes the cycle length and split ratios using Highway Capacity Manual techniques. Gordon et al. (2005) also cites the PASSER and aaSIDRA programs for this category.

Traffic Responsive Control Systems is the first type of *Traffic-Actuated Control*, which have basic traffic detection capabilities and optimize the traffic signal parameters based on the current traffic conditions. Gordon et al. (2005) explains that these systems respond to changes in traffic quite rapidly, usually at the next phase. In this category, they mention SCOOT, which optimizes the split, cycle and offsets based on detector measurements upstream of the intersection. Every four seconds, it uses the measurements to project traffic profiles to the downstream intersection using the TRANSYT dispersion model and optimizes the parameters based on this estimation. SCOOT has also been deployed in several locations in the U.S. and in some others around the world. Gordon et al. (2005) also mention SCATS (Sydney Coordinated Traffic Control System) which uses vehicle detectors placed immediately before the stop-line to adjust signal timing in response to variations in traffic demand and system capacity. It also updates cycle time, phase splits and offsets, and has been widely deployed. It has also been extensively evaluated and compared to the TRANSYT method, showing significant improvements regarding reduction in travel time, stopped vehicle delay and also number of stops.

Traffic Adaptive Control Systems are similar to responsive strategies in that the traffic signal parameters are also adapted dynamically in response to traffic changes, but the major difference pointed by Gordon et al. (2005) is that this type of systems "feature sufficient surveillance capability to provide a detailed profile of traffic approaching an intersection" (GORDON et al., 2005). Usually, this type of system uses more advanced algorithms, which are able to take advantage of the more detailed measurements of traffic (GORDON et al., 2005). In this type, it is also common that the traffic lights exchange information with their neighbors to consider the impact of incoming traffic and their neighbors' actions. In this category, they cite RHODES and OPAC. RHODES decomposes the problem in three levels: it adjusts signal phase and duration at the intersection level based on observed and predicted vehicle flows, and coordination and operational constraints; at the network level, predictions of platoon flows are used to establish coordination constraints for each intersection; and at the network loading level, the travel demand is predicted over longer periods of time to be used by the other levels. OPAC (Optimized Policies for Adaptive Control) calculates signal timings to minimize a performance function of stops and delays. It implements a "rolling horizon" strategy, over which traffic patterns are projected and optimum phase change information is calculated.

3 RELATED WORK

During the first part of this work, 119 articles published since 2010 about intersection management techniques were analysed. These articles were gathered from the main artificial intelligence and transportation research conferences (ATT, ALA, IJCAI, AAAI, AAMAS, ICML and ITSC) and also using the popular search engine, Google Scholar (which indexes academic literature from various source). The articles were filtered by the search terms "traffic signal", "traffic light", "traffic control", "intersection", "semaphore" and "junction" and then manually filtered by their relevance to the theme. After this analysis, we could see that research in recent years has been devoted mainly to advancing the Adaptive type of traffic control systems. Out of the 119 reviewed articles, 81 were concerned with optimizing the traffic signal control plans, 37 with autonomous intersection management techniques and 1 of them proposed a method for dynamic lane grouping. The research about traffic signal control plans will generally define the problem and attempt to solve it as one of the following: based on rules generated from observation and/or experience; using mathematical approximation models; as an optimization or scheduling problem; or as a Reinforcement Learning (RL) problem. See Figure B.1 for a taxonomy of Intersection Management techniques generated from this analysis. Another interesting result is regarding the used Simulation Tools. Most of Traffic Signal Control works evaluate the proposed methods using microscopic traffic simulators. The most used ones are SUMO, VISSIM and Paramics, with SUMO ranking first in the list with 14 papers using it (see Figure B.2). Next, we discuss in more detail some works that pose the problem as a RL problem and the optimization functions used (both by RL and optimization-based methods). In this section, we only discuss the main findings of the reviewed methods, focusing on their design choices and the impacts that they have in the results, without going into detail about all aspects of each method. Similarities and differences in how they approach the problems are also highlighted. To help with the understanding of this discussion, we provide an overview of the reviewed methods in Chart B.1.

The choice of problem definition was perceived as one of the most important design choices as it greatly influences the results. Designing the solution around a proper problem definition is decisive for the success of the proposed methods and can help prevent problems that incur from lack of flexibility from the model. For example, Teo, Kow and Chin (2010) propose a Genetic Algorithm (GA) based method where the green times for all phases are determined at the beginning of the cycle based on the current queue lengths. The tests have shown that this choice of action leads to benefiting phase 1, but does not generate adequate values for the other phases. The authors try to workaround this by incorporating forecasts of arriving demand for the remaining phases. This does improve the situation a bit, but the experiments showed that this was not so effective, as phases 2 and 3 accumulate many more cars than phase 1 during the simulation. This could be due to: (1) the incoming traffic flows estimation is not accurate; (2) it's not adequate to define the green time of all phases at the beginning of an cycle; (3) the input variables are not adequate; or (4) a combination of the three. Another potential problem that was found is over-discretizing the inputs and the outputs of the model. Prabuchandran, Kumar and Bhatnagar (2014) propose a Q-learning based approach, but they discretize both the inputs (queue lengths) and the output of the algorithm (duration of green time), which limits the flexibility of the model and results in limited improvements.

Around the topic of state representation, El-Tantawy and Abdulhai (2010) proposed a Q-learning based signal control method with a variable phasing sequence and they compared 3 different state representations. S1 considers the number of vehicles arriving for the green phase and the queue length for each of the red phases; S2 uses the queue length of each phase; and S3 uses the summation of the cumulative delay of all vehicles in the queue for each phase. The ones that yield better results are S2 and S3 and the authors conclude that the queue length provides a better representation of the current state of the network, with S3 also performing well as the cumulative delay is directly related to the queue length. For high traffic scenario, the one based on cumulative delay (S3) slightly outperformed the one based on queue length (S2).

Another interesting approach is to incorporate rules in the design of the algorithm or of the objective function to solve problems which the system designers are already aware of. Bi et al. (2014) proposes a type-2 Fuzzy-Logic Control (FLC) method optimized by Differential Evolution (DE), where a second layer takes over if they detect congestion in downstream links to prevent sending more vehicles to already congested areas. Results show that this reduced the formation of large queues and the number of blocked intersections, which shows the benefits of enforcing rules that are known a-priori and how it can improve the system's actions. They used a direct coordination approach, but this type of rule could also be incorporated in an objective or cost function to improve the flexibility of the system. Incorporating rules in the design of the algorithm can also help agents to learn their policies. Mannion, Duggan and Howley (2015) extend a Q-learning based method to control the green signal duration by applying Potential-Based Advice, which is a technique to provide guidance to learner agents using knowledge specific to the problem environment. They use the same state, action and reward definitions as El-Tantawy and Abdulhai (2012), where, at each time step t, the agent can decide whether to keep the current phase or choose the next phase to be active. The Look-Ahead Advice potential function is defined as the relation between the queue length of phase corresponding to the proposed action and the sum of all queue lengths. This is similar to a longest queue first rule, as the potential reward is higher when the proportion of queuing vehicles for that phase is higher, which means that the agent will be encouraged to give more green time to phases with the longest queues. Tests have shown that the RL approach that employs Potential-based Advice has an increased rate of learning and better performance in terms of Average Waiting Times (AWT) and Average Queue Lengths (AQL) at the end of the training period.

The Reward/Objective Function is also an important choice when designing optimization or RL-based controllers and can be used to tune the kind of solution at which the model is expected to arrive. Most of the proposed methods consider more immediate and local indicators as the reward function. Abdoos, Mozayani and Bazzan (2011), for example, use a reward inversely proportional to the average length of the queues in the approaching links, normalized to remain between 0 and 1. Other papers use the reward as a function of vehicle cumulative delay. For example, trying to minimize the average delay of vehicles (PRABUCHANDRAN; KUMAR; BHATNAGAR, 2014) or defining the reward as the difference (savings) between the total cumulative delay of two successive decision points (EL-TANTAWY; ABDULHAI, 2010; EL-TANTAWY; ABDULHAI; ABDELGAWAD, 2013; MANNION; DUGGAN; HOWLEY, 2015). This type of reward function is well suited for under-saturated conditions, where the controller can focus on the driver's convenience. Li et al. (2013), on the other hand, propose an objective function aimed at improving the traffic conditions of over-saturated intersections. The proposed function attempts to maximize the throughput, which makes the algorithm tend to extend the stage lengths in order to avoid the lost time due to phase transition process, while minimizing the queue ratio, which prevents the stage length being so long that the queue of the conflicting approaches grows too large, which could potentially lead to queue "spillback" and block intersections in the network. The fact that under-saturated and over-saturated conditions have difference characteristics (GORDON et al., 2005) and that there are different objective functions that meet those different characteristics suggests that an approach that dynamically adjusts its objectives according to the traffic conditions could be beneficial. A possible alternative is a type of adaptive multi-model function, as proposed by Dujardin et al. (2011) (discussed next), which adapts its weights of the multi-modal function depending on the traffic conditions.

Conversely, Christofa and Skabardonis (2011) and Dujardin et al. (2011) use the objective function to give priority to transit vehicles and make public transportation a more attractive choice. Christofa and Skabardonis (2011) proposes to minimize the total person delay at intersections, considering the number of passengers that each type of vehicle is able to carry. By minimizing the total person delay as the objective function, they are able to reduce the overall passenger delay, giving priority to buses, while limiting the impacts to auto passenger delay. With this method, total person delay was reduced by 9.5% and bus passenger delay was reduced by 35.5% compared to the results of vehicle-based optimization. The increase of auto passenger delay was only about 2.8%. Dujardin et al. (2011), on the other hand, proposes an adaptive multi-modal objective function, which considers the cumulative waiting time, the number of stops and the total delay of buses. In order to avoid having to define weights for each objective a-priori, they propose to adapt the objective function according to the traffic situation. For example, when the controller detects that there are buses waiting, it considers the total delay of buses more in the objective function.

When it comes to considering road networks where several intersections have Traffic Lights, it becomes important to consider the impacts that the signal timing from one intersection have on their neighbors and on the whole network. However, it's not possible to have global knowledge of the network when calculating the signal timing plans of each intersection, as this would lead to exponential growth in the state and action representations. To overcome this problem, coordination schemes have been proposed, which can be categorized as Indirect Coordination and Direct Coordination. In Indirect Coordination, non-local information from neighbor intersections is incorporated as input to the algorithm in each intersection. In Direct Coordination, on the other hand, neighboring intersections exchange messages to coordinate their actions. In this review, it has been noticed that Indirect Coordination is a popular approach, as it simplifies the problem and helps to achieve good results. In Vázquez et al. (2010), queue lengths from upstream intersections are incorporated as inputs, and the experiments show that the intersections were able to synchronize the green period with the outgoing platoons from the upstream intersections, in order to form green waves. Araghi, Khosravi and Creighton (2015) also use implicit coordination by incorporating the number of vehicles coming from the neighbor intersections as inputs to the algorithm. In Xie et al. (2011), they propose a self-scheduling approach to traffic signal controls, where traffic flow is aggregated into groups of vehicles (platoons) and the intersections treat each platoon as a job in a scheduling problem, applying a simple set of rules on whether to extend or change a phase. Next, Xie, Smith and Barlow (2012) extend this approach by aggregating the expected output flows from upstream intersections into the cluster sequences that are the inputs to the scheduling problem. This helps them achieve indirect coordination and favor the progression of platoons through a road. Then, Hu and Smith (2018) further extend this approach by incorporating congestion feedback from the intersection's downstream neighbours, as sending vehicles into already over-satured intersections will lead to queue "spillback" and block the intersection. This favors phases that send vehicles into non-congested areas of the network. The tests show that each of the extensions provide some level of improvement over the previous model and to a better performance of the network as a whole.

El-Tantawy and Abdulhai (2012) extend their previous work (EL-TANTAWY; ABDULHAI, 2010) with a coordination strategy, which incorporates the state and actions from its neighbors into the decision. The authors point that this improves the learning conversion, as coordination allows the agents to consider the impacts of policies from their neighbors. Without any coordination in multi-agent learning problems, agents are faced with a moving-target learning problem. In this work, they compare direct (through message exchange) and indirect (through estimations) coordination. The levels of performance are similar, but indirect coordination converges faster and requires less computation time. However, this result is quite specific to Q-learning and to their formulation, and may not apply to other techniques. Regarding explicit Direct Coordination, Shirai et al. (2012) proposes a two layer framework using direct and indirect coordination. Their proposed Direct Coordination protocol takes control when it detects congestion and then coordinates with its neighbors to adjust their offset and achieve green wave formation. Neighbors then engage their own neighbors and so on. Their main contribution is a framework where green-waves can emerge from anywhere in the network, as intersections participating in the green-wave formation do not need to be predefined. This allows an increased level of adaptability for the network and the authors show it can adapt quickly to road blockages and unexpected events.

As discussed above, there are a variety of methods and important aspects that should be considered when designing a method for Traffic Signal Control. And, even though there is much research on the subject, Traffic Signal Control is a very complex domain and there are still open possibilities for research. For example, current approaches focus either on improving the traffic signal timings for under-saturated or over-saturated traffic conditions, which results in limited improvements under the conditions for which the method was not designed. To take full advantage of adaptive traffic signal control, a method that is able to dynamically adapt and is suitable for both conditions is highly desirable. In this work, we tackle this by introducing an adaptive reward function that should be able to work well for both under-saturated and over-saturated conditions, adapting itself automatically without the need of human interference.

4 PROPOSED METHOD

4.1 Overview and motivation

The method proposed in this paper has been motivated by the different characteristics of under-saturated and over-saturated traffic demands and the different optimization objectives that traffic lights should take into account under each situation. Using the total travel time (POHLMANN; FRIEDRICH, 2010) or waiting time (MANNION; DUGGAN; HOWLEY, 2015) as optimization function, for example, primarily targets the driver's convenience of being able to pass through the intersection faster. And while this is suited for under-saturated conditions, it could have negative effects under over-saturated scenarios. For over-saturated conditions, the optimization objective should be primarily concerned with the overall health of the network and returning the traffic situation back to normal as soon as possible. In Li et al. (2013), for example, they proposed a multi-objective optimization function targeted for over-saturated conditions based on maximizing throughput, while maintaining the queue ratio. Maximizing the throughput makes the algorithm tend to extend the cycle length to avoid the lost time due to phase transition process. However, extending it too much could cause the queue of the conicting approaches grow too long and potentially lead to queue *spillback*. Optimizing for the queue ratio tries to prevent this and maintain the queue length at acceptable levels for all incoming approaches, thus preventing or minimizing the number of blocked intersections.

Taking into account that the traffic situation usually ranges from under-saturated to over-saturated conditions throughout the day and one of the goals of traffic control is to be able to adapt to such changes, it is desirable to have an optimization function that either (1) is able to naturally adapt to such variations or (2) that adapts to such variations by changing the optimization criteria according to the traffic conditions. In this paper, we investigate the second option by proposing an Adaptive Reward Function, which receives one reward function to be used for under-saturated and one for over-saturated conditions and is able to adapt to using one or other according to some dynamic weight, which in this case should be a measure of traffic saturation.

4.2 Algorithm Definition

The design elements of the proposed Q-learning algorithm are discussed next in terms of the typical Reinforcement Learning Structure.

4.2.1 Action

In the proposed method, we adopted a fixed-phasing sequence, where the action is the effective green time of the next stage (similarly to what was used by Prabuchandran, Kumar and Bhatnagar (2014)), discretized by intervals of 3 seconds. As such, an action is taken at the beginning of each stage and the action can be defined as:

$$a = \{\, i \in N;\ 2 \leq i \leq 40 \,\} \tag{1}$$

The next stage green time (in seconds) is defined as $g_i = a * 3$.

4.2.2 Action Selection

At each decision point, the next action is chosen according to an ϵ-greedy policy, in which the ϵ-greedy learner selects the greedy action (that is, the action with highest Q-value) most of the time, except for ϵ amount of the time, when it selects an action uniformly at random (SUTTON; BARTO, 1998). In all of the conducted experiments, the ϵ value starts at 0.7 in the first simulation run and decreases by 30% at the end of each simulation, until it reaches a minimum of 0.1. Decreasing the value of ϵ is a common practice in Q-learning models as it enables the Q-Learning agent to explore the state-action space at the beginning of the learning process and then favors exploitation as the agent converges to an optimal policy (EL-TANTAWY; ABDULHAI, 2012).

4.2.3 State Representation

Also similar to what was used by Prabuchandran, Kumar and Bhatnagar (2014), the state is represented by an array containing the max queue lengths of each stage and an additional element containing the index of the current stage. Formally, we have:

$$S = \left\{ \left(\max_{l \in L(i)} q \right) \ \forall i \in \{1, 2, .., N\} \right\} \cup \{\, c \,\} \tag{2}$$

Where $L(i)$ is the set of all lanes for stage i, q is the queue length, N is the number of stages and c is the current stage index. As in El-Tantawy and Abdulhai (2010), a vehicle is considered at a queue if its speed is below 7 km/h. The queue length was chosen as the main part of the state representation as, in a comparison by El-Tantawy and Abdulhai (2010), it yielded better results (along with a cumulative delay representation). They also point that it is the most widely used state representation in RL-based literature. Then, it was decided to also include the current stage index because of the problem definition of selecting the green time of the next stage as the action, so that the model can correlate it to the queue length information and make a proper decision.

4.2.4 Reward Function

In this section, we first present the proposed Adaptive Reward Function and then the Reward Functions which were taken from the literature and evaluated by this work.

4.2.4.1 Adaptive Reward Functions

As explained previously, the proposed Adaptive Reward Function receives one reward function to be used under-saturated and one for over-saturated conditions and it adapts to using one or the other according to some dynamic weight, which in this case should be a measure of traffic saturation. It does so by performing an weighted sum of the reward of each reward function, where the weights are based on the aforementioned dynamic weight. In this paper, we've used a measure of lane occupancy (in percentage) provided by the traffic simulator. We also examined the possibility of using some measure based on arrival rate, but this would not account for situations where the lane is already fully occupied and no more vehicles can arrive and, thus, the arrival rate would be zero. The formula based on the adaptive weighted sum is the following:

$$r = ((1 - \omega) * r_u) + (\omega * r_o) \tag{3}$$

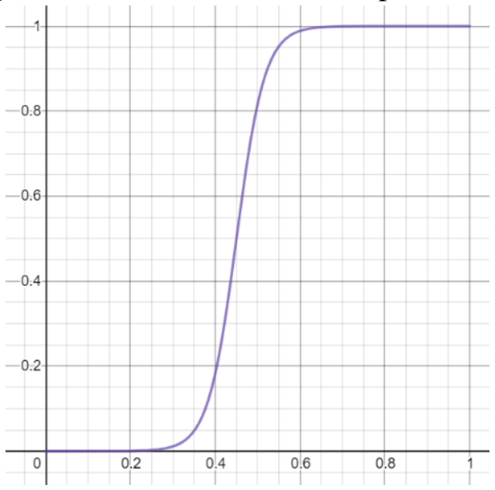
Where: ω is the dynamic weight after applying the activation function; r_u is the reward function for under-saturated conditions; r_o is the reward function for over-saturated conditions. The dynamic weight for r_u is $1-\omega$ because the dynamic weight being used increases as the saturation levels increase and the under-saturated reward function should be used when the saturation levels are the lowest.

An activation function is applied to the dynamic weight so that the reward function uses either most of one reward function or most of the other, preventing that the result of the reward function is a mix of the two which would be hard for the model to understand. For it, we used an adapted logistic function, which assumes an dynamic weight input in the range $[0, 1]$ and outputs the activated dynamic weight also in the range $[0, 1]$. The adapted logistic function used for activation is the following:

$$\omega = \left(\frac{1}{1 + e^{-\kappa * (\upsilon_a - \iota)}}\right) \tag{4}$$

Where κ and ι are parameters of the function: κ represents how steep the curve of the function will be and we chose an default value of 15 for this parameter; and ι is the inflection point of the curve, which represents where the curve will be centered. υ_a is an adjusted dynamic weight where $\upsilon_a = (\upsilon * 2) - 1$. This adjustment was applied so that the output of the activation lies in the range $[0, 1]$. The parameter ι has been tuned by running simulations with different values for this parameter and verifying which worked best, as explained in Section 4.2.4.2. The chosen value was -0.1. Figure 1 shows the activation function behavior with the chosen parameters as a function of the dynamic weight.

Figure 1: Adapted logistic activation function with parameters: $\kappa = 15$; $\iota = -0.1$.



Source: Own author (2019).

4.2.4.2 Evaluated Reward Functions

The intuition behind the adaptive reward function assumes that there is some reward function that works better for under-saturated conditions and some reward function that works better for over-saturated conditions. To verify if such premise is true, we evaluated several reward functions taken from the literature and a few adaptations. The experiments and results are discussed in detail in Section 5.2.2. Based on the results, we decided to apply the Average Vehicle Number and the Cumulative Vehicle Delay with penalty for Wasted Time as the reward functions targeted at under-saturated conditions and Throughput as the reward function for over-saturated conditions. The evaluated reward functions are described below. It is important to highlight that all reward functions and penalties are normalized to have a value below one, but that they are not bound to be higher than zero.

4.2.4.2.1 Average Queue Length

Defined as $r = 1 - (q_{avg}/max_q)$, where q_{avg} is the average queue length across all lanes and the max_q is the maximum acceptable queue length, defined as 70% of the maximum number of cars that a lane can hold. This reward function is similar to the one used by Abdoos, Mozayani and Bazzan (2011).

4.2.4.2.2 Average Vehicle Number

It was observed that, right after the green time ends and the signal advances to the next phase, the vehicles in the lanes of the last active stage are still moving and none (or very few) will have a speed below 7 km/h and, thus, be considered for the queue length measurement. As a consequence, the reward value will be the same whether or not the green time was enough to clear the queue of that approach. To resolve this issue, we introduced this reward function, which is based on the average of the number of vehicles in all incoming lanes. It can be defined as $r = 1 - (v_{avg}/max_q)$, where v_{avg} is the average of the vehicle number from all incoming lanes.

4.2.4.2.3 Cumulative Vehicle Delay

The vehicle cumulative delay D_v is the total time spent by this vehicle v in a queue. The cumulative delay for stage i, $D(i)$ is the summation of the cumulative delay of all the vehicles that are travelling on the lanes that form that stage (EL-TANTAWY; ABDULHAI, 2010). This reward can be defined as $r = 1 - (D(i)/D_{max})$, where i is the last active stage and D_{max} has been empirically adjusted to be the sum of the maximum possible queue lengths times the maximum possible stage length. Reward Functions based on the cumulative delay are one of the most common in RL Traffic Light Controller literature (EL-TANTAWY; ABDULHAI, 2010, 2012; MANNION; DUGGAN; HOWLEY, 2015).

4.2.4.2.4 Cumulative Vehicle Delay with penalty for Residual Queue

After observing that the model would choose green times that were not long enough to clear the stage's queue, a reward function that introduced a penalty for residual queue was implemented. The residual queue is defined as the number of vehicles that were already in the queue at the beginning of the stage and were not discharged until the end of that stage. The intuition was to guide the model to learn to fully dispatch a queue every time. The penalty can be defined as

$$p = \sum_{l \in L(i)} \frac{rq(l)}{q_{max}} \tag{5}$$

where $rq(l)$ is the residual queue for lane l and i is the last active stage index. The reward can be defined as $r = r_{bf} - p$, where r_{bf} is the reward of the base function (Cumulative Vehicle Delay, in this case).

4.2.4.2.5 Cumulative Vehicle Delay with penalty for Wasted Time

This reward function has been implemented because of a reason inverse to the previous one. It was observed that the model would sometimes choose reward functions that were too long and would waste effective green time long after the queue had already been cleared, while the queue of the conflicting approaches was increasing. To prevent this from happening, a penalty for wasted time was introduced. Wasted time is the amount of effective green time after the queue at the beginning of the stage has already been cleared. It can be defined as $wt = g_t - c_t$, where g_t is the effective green time and c_t is the time need to clear the original queue. The penalty is the proportion of wasted time in relation to the maximum possible stage time: $p = wt \,/\, s_{max}$.

It was decided to apply the penalties only to the cumulative delay reward function because it is one of the most widely used reward functions in the related literature.

4.2.4.2.6 Throughput

This reward function is an adaptation of the objective function from Li et al. (2013). It is the average of two reward functions: maximizing for throughput and minimizing the average queue ratio. As explained previously, the intuition behind this objective function is that maximizing for throughput will make the model extend the stage time, with the objective of minimizing the lost time due to phase transition process (yellow and all red times), while minimizing the queue ratio will prevent it from extending the stage too long in a way that would cause the queue of the conflicting approaches to grow too large. The reward function for maximizing throughput is defined as

$$r_{th} = (sr * st)/(sr * st_{max}) \tag{6}$$

where sr is the saturation rate the arrival in veh/s under a situation of saturation flow, st is the last stage time and st_{max} is the maximum possible stage time. The reward function for minimizing the queue ratio is based on the average of maximum queue ratios of each stage. It can be formalized as

$$r_{qr} = 1 - \frac{\sum_{i=1}^{N} \left\{ \max_{l \in L(i)} \frac{ql(l)}{ql_{acc}} \right\}}{N} \tag{7}$$

where $ql(l)$ is the queue length of the corresponding lane and ql_{acc} is the maximum acceptable queue length. The functions are then combined by taking the average.

4.2.4.2.7 Actual Throughput and Maximum Queue Ratio

This is an adaptation of the Throughput reward function. In this, the throughput is no longer considered to be a function of the saturation rate and stage time, but rather considers the actual number of vehicles that were able to pass the intersection during the green time by simply counting them. In a real-world scenario, this could be achieved with an induction loop placed at the very start of the conflict zone. Regarding the queue ratio, instead of taking the average from the maximum queue ratios, this function considers the maximum queue ratio across all incoming lanes. The intuition behind this decision is that taking the average could be misleading in situations where some lanes have a really small queue, but a few lanes are completely saturated. Therefore, taking the maximum queue ratio instead of the average could help in giving a more accurate sense of whether the queue is growing too long in one of the conflicting approaches.

5 EXPERIMENTS

In this section, we conduct experiments using a microscopic traffic simulator to (1) evaluate how reward functions behave under different levels of demand and (2) to evaluate the performance of our proposed adaptive reward function and compare it against related work. We begin by describing our methodology (simulation tool, intersection topology, demand generation, development tools and code architecture) in Section 5.1, and then present our numerical results and discussion in Section 5.2.

5.1 Simulation Setup

5.1.1 Simulation Tool

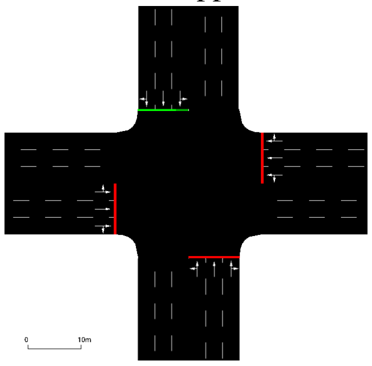
The simulations are executed using SUMO, an open source, microscopic and continuous traffic simulation package that help to prepare and perform simulation of traffic (KRAJZEWICZ et al., 2012). It started being developed by the German Aerospace Center (DLR) in 2001 and had its first open source version released in 2002. It is considered to be time-discrete with step length of 1 second and space-continuous. It features an API, called TraCI (Traffic Control Interface), which uses a TCP based client/server architecture to provide access to SUMO. SUMO was chosen as the simulation tool for our experiments because it is open source and the TraCI API allows to interact and customize the simulation while it is running, making it possible to implement custom Traffic Light Control.

5.1.2 Intersection Topology and Demand Generation

For all of the experiments, we use an isolated intersection network, which consists of four approaches and three lanes each. For each approach, all lanes allow vehicles to move straight forward. Besides, the leftmost lane also allows vehicles to turn left and the rightmost lane also allows vehicles to turn right. It was decided that there would be four stages, one for each approach, such that all the movements from one approach are allowed during the same stage.

Demand for the simulations has been generated using SUMO's utility script *randomTrips.py* and routes for the output trips have been generated using SUMO's utility tool *DUAROUTER*. Demand has been generated based on given levels of demand in relation to intersection capacity. The capacity of a lane group can be calculated as $c_i = s_i * (g/C)_i$, where: s_i is the saturation flow rate for lane group i (veh/h); $(g/C)_i$ is the ratio of green time (g) to cycle time (C) for lane group i (STOKES; BANKS, 2004). The saturation flow for each lane group can be estimated as $s_i = \sum_{l \in L(i)}(525 * w_l)$ where w_l is the width of lane l (WEBSTER, 1958). Given $w_l = 3.2$ for all lanes in this intersection and 3 lanes forming each group, we have $s_i = 5.040$. Assuming that the green time is equally divided among the four groups, we can approximate the capacity equation to $c_i = 5.040 * 0.25 = 1260$. Considering the four lane groups, we have a total capacity $c = 5.040$ (veh/h). This capacity value serves as a basis to generate demand for the simulations.

Figure 2: Intersection with four approaches and three lanes each.



Source: Own author (2019).[3]

5.1.3 Development Tools and Resources

As mentioned, SUMO provides a TraCI library to connect to the simulation and interact with it while it's running. This library is available for a variety of languages: Python, C++, .NET, MatLab and Java. We have decided to use Python to build our experiments because of the author's familiarity with the language. As supporting tools, we use *Astah UML* to create UML diagrams and plan the implementation, *Git SCM* as the source code's and simulation results' version management system, and *github.com* as our central code repository. Regarding the development and execution environments, we use *Docker* to containerize the simulation environment and make it portable, and *AWS EC2* servers with Intel Xeon Platinum 3,0 GHz processor to run the simulations. Finally, to manage the project, we use a Kanban Board in *trello.com* to keep track of the project's tasks and their progress, and *hackmd.io* to keep a journal of the implementation and experiments.

5.1.4 Simulation Architecture

Vilarinho, Tavares and Rossetti (VILARINHO; TAVARES; ROSSETTI, 2016) propose the design of a Multiagent System for Real-Time Traffic Control, which served as inspiration for the implementation of our system, in a much simplified manner. Given that the main intention of this system is to allow experimenting and the testing of hypothesis, it's important that the architecture of the system provides the flexibility of allowing to easily replace one implementation by another and also allow the combination and re-use of its components. With that in mind, the Simulation system has been developed with a strong focus on using Object-Oriented design patterns to allow for modularization, flexibility and re-usability. The component diagram (Figure 3) shows that the developed simulation system has been organized into 5 modules: Simulation

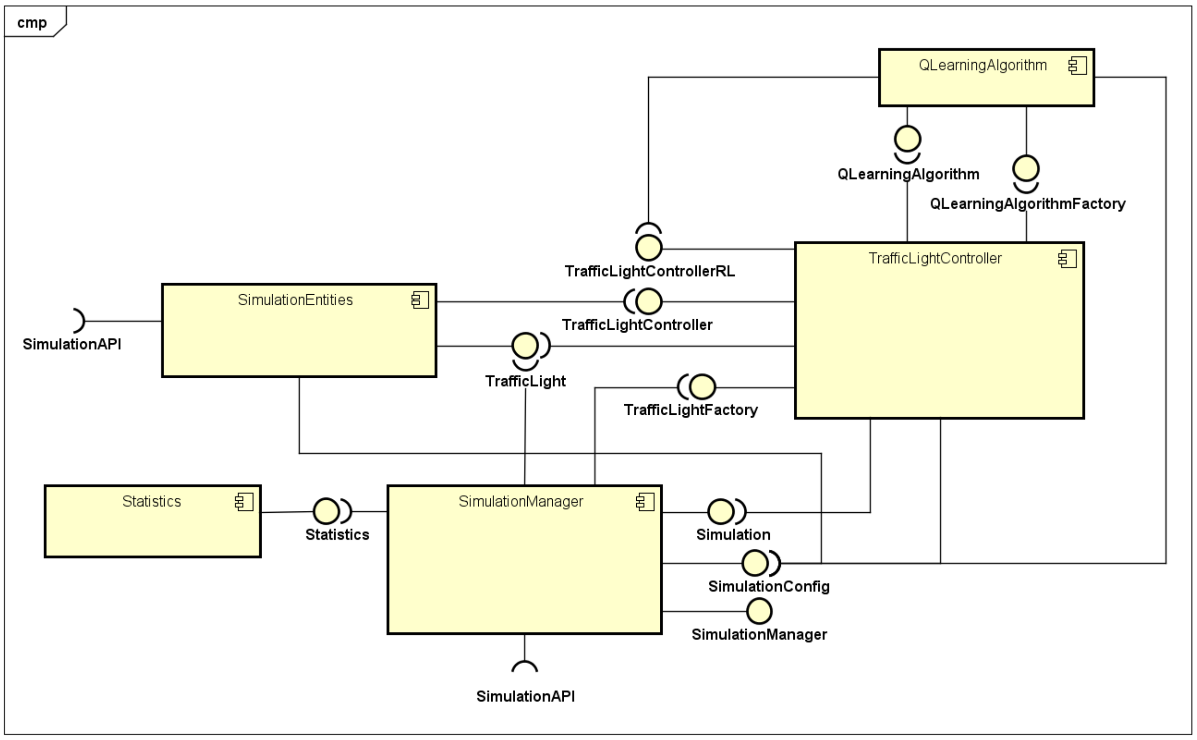
[3]Snapshot generated by SUMO from the isolated intersection network.

Runner, Simulation Entities, Traffic Light Controller, Q-Learning Algorithm and Statistics. Details about the 5 modules, their implementation and intention behind the design are explained next.

5.1.4.1 Simulation Runner

The Simulation Manager module is responsible for coordinating all of the parts in the simulation. The interface *SimulationManager* allows the user to submit an experiment, which has the goal of running simulations with different parameters and collecting statistics about the simulation runs for later analysis. Figure C.1 shows this process in detail. Basically, for each experiment, an configuration file is loaded to the *SimulationConfig* class and a new Simulation object is created. The *SimulationManager* then subscribes the relevant *Statistics* objects to the current simulation and starts the simulation. The simulation object will first start SUMO through the *TraCI* API and get a list of Traffic Light IDs. Based on the IDs and on the *TrafficLightController* type obtained from the configuration file, *TrafficLight* objects are created using the *TrafficLightFactory*. After that, the Simulation will call *TraCI* to execute one simulation step and then call all its *TrafficLight* objects to perform one step as well, until there are no more vehicles in the network. After the simulation finishes, the Statistics objects are requested to save their data and control goes back to the *SimulationManager* to submit another simulation execution.

Figure 3: Component Diagram of the simulation system developed for the experiments.



Source: Own author (2019).[4]

5.1.4.2 Simulation Entities

The goal of the Simulation Entities module is both to provide an easy point of access to all traffic information required by the other modules as well as to decouple the simulation system from the TraCI API. This was done by implementing a class for each traffic control entity (refer to Figure C.2), such that each provides an interface to obtain information and take actions, implementing the requests using TraCI API. In this sense, the Simulation Entities act as an adapter, converting the TraCI interface into another interface that is suited for our simulation system (GAMMA et al., 1994) and making it easier for the other modules to access the required information. This encapsulation would also make it easier to change the simulation tool being used, as it would only require a new implementation of the Simulation Entity classes.

5.1.4.3 Traffic Light Controller

The Traffic Light Controller module implements all the logic for controlling the Traffic Lights. In this type of experiment, it is common that one would like to implement the controller logic under study (in our case, a Q-Learning controller) and also one (or more) controllers that will serve as a baseline (in our case, a fixed time controller). To allow to vary the controller algorithm independently from clients that use it (in our case, the Traffic Lights), we have used a Strategy pattern, which "defines a family of algorithms, encapsulates each one, and makes them interchangeable" (GAMMA et al., 1994). This is implemented through the *TrafficLightController* abstract class (see Figure 4), which has an *step()* abstract method, This method is called by the *TrafficLight*'s own *step()* method and the concrete classes should implement it to define their behavior.

As concrete strategies, we have a fixed time controller, *TrafficLightControllerFXM*, and a Q-learning controller, *TrafficLightControllerQLearning* (which is actually an abstract class). For the Q-learning controller, it is possible to use one of many problem definitions. In our case, we have decided to use a fixed-phasing controller that chooses the effective green time of the next stage at the beginning of the stage. But, it is also possible to opt for other definitions, such as a variable-phasing controller that chooses whether to extend or switch to a new phase at every time step. Three things can change according to the chosen problem definition: the list of possible actions, when decision points happen (i.e. when next action needs to be taken) and how to execute the action (i.e. what the action actually means). To allow for several implementations of the Q-learning controller, we have applied a Template Method pattern, where an abstract class defines the skeleton of an algorithm in operation and defers some steps to subclasses, which redefine certain steps of the algorithm, without changing its structure (GAMMA et al., 1994). In our case, the abstract class *TrafficLightControllerQLearning* defines the skeleton of the algorithm and the concrete class *TrafficLightControllerQLearningFPVCL* (the FPVCL stands for

[4]Diagram created using Astah UML.

Fixed-Phasing Variable Cycle Length) implements the above three methods. In this concrete implementation: the list of possible actions will be an array of possible stage lengths, respecting a range of minimum and maximum length; the next action will be taken at the beginning of each stage; and taking the action means setting the next stage length in the controller.

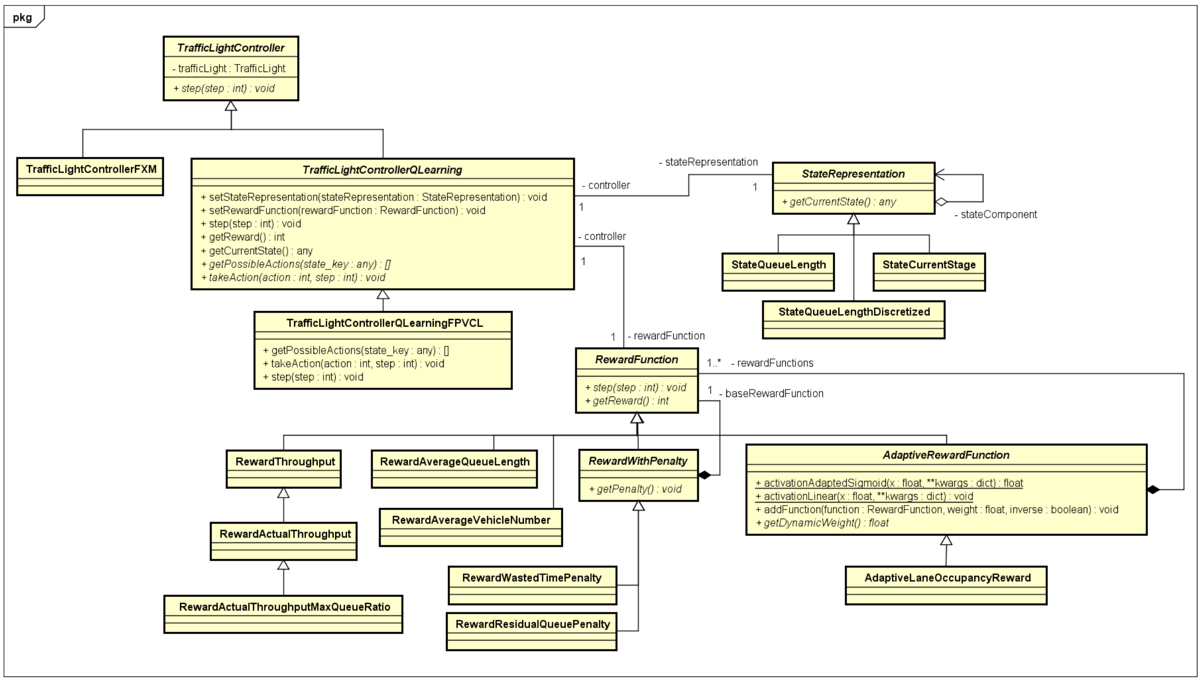
Another important aspect of the Q-learning controller is how to implement the State Representation and the Reward Functions. Since the State Representation can be an aggregation of several indicators and the Reward Function is the main aspect under investigation in this paper, it was a requirement to easily allow new implementations to be tested and even combined among them. For example, the State Representation can be a combination of the Queue Lengths of all approaches and the current stage of the Traffic Light, however, it could also be a combination of the Cumulative Waiting Time of the vehicles and the current stage of the Traffic Light. A similar scenario applies to Reward Functions and being able to combine them in Multi-Objective Reward Functions, for example. Therefore, it would be impractical to extend by subclassing, as there is a large number of different combinations and it would result in an explosion of subclasses to support all of them. An alternative to subclassing for extending functionality is the Decorator pattern, which allows to attach additional features to an object dynamically (GAMMA et al., 1994). This pattern has been applied to the implementation of both State Representation and Reward Functions.

The case for *StateRepresentation* is relatively straightforward and only involves combining different State Representations. The result should be either a concatenated string or a concatenated array. The Decorator pattern is implemented by passing one *StateRepresentation* component to the new *StateRepresentation*, such that it creates a chain-like structure. When the *getCurrentState()* method is called for the *StateRepresentation* object, it calls its state component *getCurrentState()* method and concatenates it in the result. This is done recursively until the end of the chain.

As for the *RewardFunction*, this pattern has been used for both the *RewardWithPenalty* and for the *AdaptiveRewardFunction*. For the *RewardWithPenalty* type, the idea is that it can receive any concrete *RewardFunction* to obtain a base reward and it will apply a penalty on top of this base reward. The *RewardWithPenalty*, itself, applies Template Method pattern, where it defines the skeleton and concrete implementations only need to implement the *getPenalty()* method to define which indicator will be used to generate the penalty. As for the *AdaptiveRewardFunction*, the idea is that it can receive reward functions for under-saturated and over-saturated conditions and it will combine them according to some dynamic weight. It also applies the Template Method pattern and its concrete classes must implement the *getDynamicWeight()* method to define what indicator will be used for the dynamic weight.

With this architecture, we achieve the requirement of allowing to easily exchange and also to combine State Representations and Reward Functions, allowing to create a new type of controller simply by changing the simulation configuration, as opposed to having to create a new concrete implementation of some class.

Figure 4: Class diagram of the Traffic Light Controller module.


pkg
TrafficLightController
- trafficLight : TrafficLight
+ step(step : int) : void
TrafficLightControllerFXM
TrafficLightControllerQLearning
+ setStateRepresentation(stateRepresentation : StateRepresentation) : void
+ setRewardFunction(rewardFunction : RewardFunction) : void
+ step(step : int) : void
+ getReward() : int
+ getCurrentState() : any
+ getPossibleActions(state_key : any) : []
+ takeAction(action : int, step : int) : void
TrafficLightControllerQLearningFPVCL
+ getPossibleActions(state_key : any) : []
+ takeAction(action : int, step : int) : void
+ step(step : int) : void
- controller
1
- stateRepresentation
1
StateRepresentation
+ getCurrentState() : any
- stateComponent
StateQueueLength
StateCurrentStage
StateQueueLengthDiscretized
- controller
1
1
- rewardFunction
RewardFunction
+ step(step : int) : void
+ getReward() : int
1..*
- rewardFunctions
1
- baseRewardFunction
RewardThroughput
RewardAverageQueueLength
RewardWithPenalty
+ getPenalty() : void
AdaptiveRewardFunction
+ activationAdaptedSigmoid(x : float, **kwargs : dict) : float
+ activationLinear(x : float, **kwargs : dict) : void
+ addFunction(function : RewardFunction, weight : float, inverse : boolean) : void
+ getDynamicWeight() : float
RewardActualThroughput
RewardAverageVehicleNumber
RewardActualThroughputMaxQueueRatio
RewardWastedTimePenalty
RewardResidualQueuePenalty
AdaptiveLaneOccupancyReward

Source: Own author (2019).[5]

5.1.4.4 Q-Learning Algorithms

The Q-learning Algorithm module's single responsibility is to provide the Q-learning algorithms that are used by the Q-learning Traffic Light Controller. The idea is to allow the Traffic Light Controllers to use different Q-learning algorithm implementations, such as Greedy Q-learning or Deep Q-learning. For this, the Strategy pattern has been applied and the algorithms must conform to the required interface, providing an *step()* method which implements the learning process (see class diagram in Figure C.3). As of now, only a Greedy Q-Learning Algorithm has been implemented. It uses *pyqlearning* (ACCELBRAIN, 2019) for the underlying Greedy Q-Learning, which is a Python library that implements Reinforcement Learning and Deep Reinforcement Learning.

5.1.4.5 Statistics

To gather statistics about the simulation, we have applied the Template Method pattern, where an *Statistics* abstract class implements the behavior of saving a *CSV* file with the information and it leaves up to the concrete sub-classes to implement how they will insert new records through the *update()* method (see Figure C.4). The Statistics implementation work in an Observer fashion, where Statistics objects are subscribed to some event in the running simulation and the various entities notify the *Simulation* object when some event happens, which in turn executes the *update()* method of the subscribed *Statistics*. For example, the Q-learning

[5]Diagram created using Astah UML.

Algorithm notifies the Simulation at every decision point, sending the state representation, the action that was taken and the corresponding reward. The Simulation, in its turn, forwards this information to the subscribed Statistics objects.

5.2 Experiments and Discussion of results

In this section, we discuss the experiments we conducted and their results. First, we discuss how the parameters of the Q-learning model were tuned. Next, we discuss the experiment where the Reward Functions were evaluated. Based on the analysis of the Reward Functions, we define the Reward Functions to be used for both under-saturated and over-saturated conditions in the Adaptive Reward Function. With this definition, we proceed to tune the inflection point parameter of the adaptive reward function. Finally, we evaluate the proposed Adaptive Reward Function and compare it to the base Reward Functions.

5.2.1 Experiment #1 - Hyper-parameter tuning

There are three parameters that need to be tuned for the ϵ-Greedy Q-Learning algorithm we will use for the Traffic Light Controller: learning rate (α); gamma rate (γ), which is the discounting rate for future rewards; and the ϵ-Greedy Rate (ϵ), which is the probability of the model choosing a random action (exploration) versus it choosing the action with the maximum Q-value (exploitation). The ranges that we evaluate are described in Table 1. We wanted to try broader ranges, mainly for (γ) values, but it would result in too many combinations and it was not possible to do so due to time limitation. Only for the ranges in Table 1, 60 experiments would be needed to try every possible combination, which was not feasible to executed. Therefore, we decided to tune the hyper-parameters in two parts: first, the learning rate was tuned and, then, the gamma rate and ϵ-greedy rate.

The simulation for the tuning consisted of 18.000 steps and the demand varied from around 5% to 240% of road capacity. First, the α was tuned using fixed values of $\gamma = 0.7$ and $\epsilon = 0.6$, during 4 episodes (each episode is a complete simulation execution) where the ϵ decreased by 30% at the end of each. We understand that running the simulation for more episodes would be beneficial for the simulation, however, this was not possible due to time restrictions. Results were analysed regarding the mean waiting time, time loss, departure delay and travel time of all vehicles for both the last experiment and an aggregation across all experiments. The $\alpha = 1E - 03$ had the best results regarding all of the indicators, except for the last experiment regarding waiting time, where $\alpha = 0.1$ was 0.24 second better, which is not a significant difference (see Table D.1 for results for total travel time). Therefore, $\alpha = 1E - 03$ was chosen for all of the following experiments.

Next, we evaluated the different combinations of γ and ϵ. The same demand was used, but this time 5 episodes were executed for each combination of parameters, also decreasing ϵ by

30% at the end of each. Results were analysed regarding the same indicators, but, this time, the statistics did not provide such homogeneous results as for the α tuning experiment (see Table D.1 for results for total travel time). There was no single best combination of parameters, however, the results presented some trends. The higher γ had best results in the first simulation runs, when the model was still learning, which also helped them achieve a better aggregated score. However, the aggregated score could be misleading, as for the last simulation run, the higher γ performed consistently lower when compared across the same ϵ runs for all indicators, except departure delay. This observation suggests that using high gamma rates in the beginning could help the model learn, but later on this value needs to be decreased. As for the ϵ, higher values also had better results in the first 2 experiment runs, but, after the model had already learned a bit, lower rates performed better. This suggests that starting with a high ϵ helps the model explore the action space and decreasing it as the model learns favors exploitation (EL-TANTAWY; ABDULHAI, 2010), resulting in better performance. Since, decreasing ϵ as the simulation progresses is already a common practice, we will use it. However, the idea of a decreasing γ is not so widely discussed, and would, therefore, require further investigation before being applied, so it will be out of scope for this paper. Based on the above observations, it was decided to use $\gamma = 0.6$ and $\epsilon = 0.7$, where the ϵ decreases by 30% after each simulation run.

Table 1: Hyper-parameter tuning ranges.

Parameter	**Min**	**Max**	**Increment Factor**	**Increment Value**
Learning Rate (α)	1e-05	0.1	10	
Discounting Rate (γ)	0.6	0.9		0.1
ϵ-Greedy Rate (ϵ)	0.6	0.8		0.1

Source: Own author (2019).

5.2.2 Experiment #2 - Reward Function Comparison

After tuning the hyper-parameters for the Q-learning controller, we proceed to verify if the premise that there is some reward function that works better in under-saturated conditions and, likewise, in over-saturated conditions. To verify this premise, we evaluate several reward functions taken from the literature and a few adaptations (all reward functions listed in Section 4.2.4.2) with different levels of demand. The Q-learning algorithm used (action definition, action selection and state representation) have been described in Section 4. The Q-learning parameters used are: $\alpha = 1E-03$; $\gamma = 0.6$; $\epsilon = 0.7$. Simulations are executed with 20%, 40%, 60% and 80% of demand in relation to the intersection capacity, each consisting of 7.200 steps. The simulations are executed for 10 episodes for each Reward Function and demand level, and the ϵ-greedy rate decreases from 0.7 to 0.1 during the first 5 episodes and then remains at 0.1 for the latter 5.

As it happened for the latter hyper-parameter tuning experiment, there's no clear winner for

Table 2: Mean and Standard Deviation for Travel Time (in seconds) aggregated across the latter 5 simulation runs for Experiment #2 - Reward Function Comparison.

	20%		**40%**		**60%**		**80%**	
Reward Function	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
Queue Length	134.20	73.10	150.87	71.10	207.68	104.24	273.98	112.11
Vehicle Number	129.15	**67.11**	**149.07**	**68.66**	207.15	93.34	**270.96**	108.00
Vehicle Delay	131.91	70.55	155.23	76.14	202.50	**92.06**	273.87	116.89
Throughput	133.84	73.19	153.87	75.33	213.88	118.08	270.99	110.50
Delay_PRQ	127.83	69.18	154.23	78.25	212.86	123.57	281.85	127.58
Delay_PWT	130.08	68.17	152.78	79.54	**200.77**	92.47	275.85	113.33
ATMQR	**127.36**	67.57	158.71	80.16	206.85	120.24	273.66	**107.63**

Source: Own author (2019).

the under-saturated (20 - 60% of capacity) and saturated (80% of capacity) conditions. However, the results provide some interesting insights. Table 2 shows the results of this experiment regarding Travel Time aggregated across the latter 5 simulation runs. In the table: Delay with penalty residual queue has been abbreviated as Delay_PRQ; Delay with penalty wasted time as Delay_PWT; and Actual Throughput and Max Queue Ratio as ATMQR. As we can see in the table, Vehicle Number reward function works generally well, including for saturated conditions and ranks first for 40% demand and 80% demand. The reward function based on cumulative vehicle delay works generally well too and, when it is combined with the penalty for wasted time, it does stand out for under-saturated conditions and ranks first for 60% demand. Throughput based reward function does not work so well for under-saturated scenarios, but it does work well for the 80% demand scenario, ranking 2nd with the mean only 0.3 seconds after Vehicle Number. It was believed by the authors that Throughput had some issues regarding how the throughput was calculated and that ATMQR would solve these issues. However, it performed around average for most of the demand levels, only ranking 1st for the 20% demand scenario.

After analysing the results, it was decided to test the Adaptive Reward Function with both the Average Number of Vehicles and the Delay with Penalty for Wasted Time as the under-saturated Reward Functions. The Throughput will be used as the over-saturated Reward Function.

5.2.3 Experiment #3 - Adaptive Reward Function - Inflection Point tuning

After defining which Reward Functions would be used for testing the adaptive Reward Function, we experimented with tuning the inflection point of the activation function that is applied to the Dynamic Weight. For this, we used the parameters defined so far for the Q-learning algorithm and the Adaptive Reward Function with Average Vehicle Number for under-saturated and Throughput for over-saturated conditions. The activation function being tuned is the adapted sigmoid described in Section 4, using a fixed steepness parameter of 15 and varying the inflection point parameter from -0.2 to 0.5 with increments of 0.1. Since this experiment aims at

Table 3: Aggregated Travel Time (in seconds) across simulation runs 5 - 10 for Inflection Point Tuning.

Inflection Point	**Mean**	**Std**	**Max**
-0.2	190.03	103.50	1069
-0.1	**185.93**	**95.63**	**845**
0	191.87	103.38	1188
0.1	187.68	99.49	1215
0.2	193.91	105.50	1984
0.3	188.62	97.81	1006
0.4	192.91	108.03	1150
0.5	191.53	97.63	922

Source: Own author (2019).

testing when the adaptive reward function should change from using one or the other reward functions (i.e. the inflection point), we defined a simulation with demand going from 10% to 80% of road capacity, with increments of 10% every hour, and resulting in a total of 28.800 steps. For each value of the inflection point, 10 episodes were executed, with the ϵ-greedy rate starting at 0.7 and reaching 0.1 in the fifth simulation, and stabilizing at 0.1 until the end. Therefore, only the statistics from episodes 5 - 10 were analysed. Results are presented in Table 3. The mean, standard deviation and maximum travel time are considerably smaller for inflection point -0.1, which indicates it as the optimal inflection point and the parameter that was chosen for the final simulation. With this inflection point parameter, the dynamic weight will start rising when the dynamic weight is around 0.3, pass through 0.5 at dynamic weight 0.45 and reach 1 when the dynamic weight is around 0.6 (see Figure 1). In practice, it means that the over-saturated reward function will start being predominant when the dynamic weight is higher than 0.45.

5.2.4 Experiment #4 - Adaptive Reward Function

Finally, we evaluate the proposed Adaptive Reward Function. For this, we execute two experiments: one using the Average Vehicle Number as the Reward Function for under-saturated conditions and one using the Cumulative Vehicle Delay with penalty for wasted time. For both experiments, we simulate traffic during the morning of a weekday, from 4 am to 1 pm, with a peak hour reaching 100% of road capacity. The demand used for both experiments is detailed in Table 4. For both experiments, we compare the model using the Adaptive Reward Function to models using the base under-saturated Reward Function, the Throughput Reward Function and a fixed-time controller, which acts as a baseline. The fixed-time controller is optimized for 40% of demand, with a fixed stage time of 39 seconds. For all methods, 15 simulation runs were executed and the results from the last 10 were analysed.

Table 4: Levels of demand per hour (as % of capacity) that were used in the simulation to test the Adaptive Reward Function.

From Hour	**To Hour**	**Demand (% of capacity)**
4:00	5:00	10
5:00	6:00	20
6:00	7:00	40
7:00	7:30	60
7:30	8:00	80
8:00	8:30	100
8:30	9:00	80
9:00	10:00	60
10:00	11:00	40
11:00	12:00	20
12:00	13:00	20

Source: Own author (2019).

5.2.4.1 Adaptive Reward Function based on Average Vehicle Number

In this experiment, we evaluate the Adaptive Reward Function using the Average Vehicle Number (*AdapVehNo*) as the reward function for under-saturated conditions and Throughput for over-saturated conditions. We compare this reward function with models that use only the Average Vehicle Number (*VehNo*) and Throughput (*Thp*), and also to a fixed-time controller (*FXM*), as described in the overview of this experiment. The results show that the *VehNo* outperformed *AdapVehNo* regarding both average travel time and average number of stops, as we can see in Table 5. Unexpectedly, the FXM controller performed better than the Q-learning controllers during most of the simulation, except during the high demand conditions. While unexpected, this is likely due to the fact that the FXM controller was optimized for 40% of demand and most of the simulation happened under that condition. As we can see in Figure 5, the three Q-learning methods show similar levels of performance. Compared to the *FXM* controller, they show significantly better performance after demand is above 70% of road capacity. This can be especially noted in the Number of Stops plot (Figure 5), when demand is at its highest, the *FXM* controller makes car stops 3.5 times in average to pass the intersection, while cars need to stop only 2 times for the Q-learning controllers.

5.2.4.2 Adaptive Reward Function based on Delay with penalty for wasted time

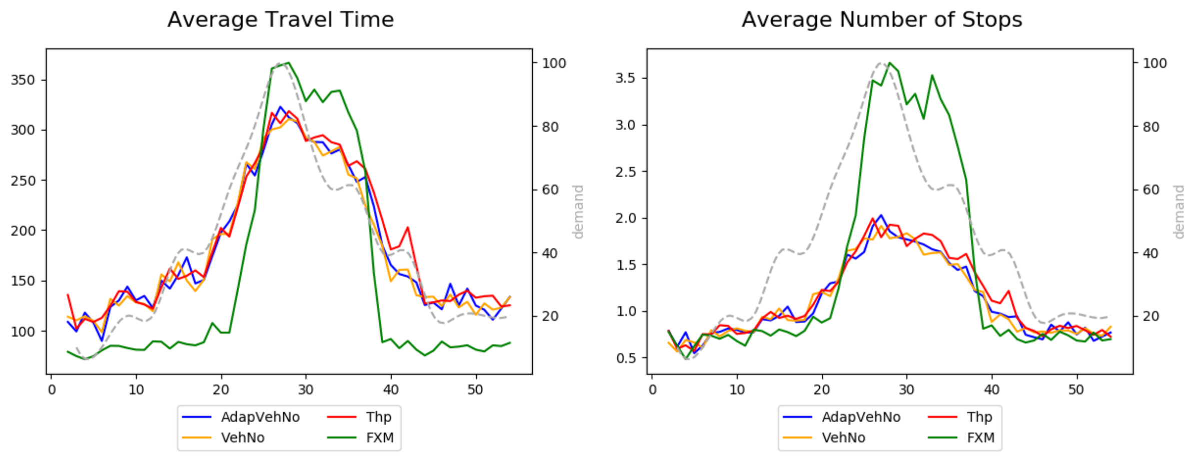
Next, we evaluate the Adaptive Reward Function using the Delay with penalty for wasted time (*AdapDWTP*) as the reward function for under-saturated conditions and Throughput for over-saturated conditions. Similar to the previous experiment, we compare this adaptive reward function with models that use only the Delay with penalty for wasted time (*DWTP*) and Throughput (*Thp*), and also to a fixed-time controller (*FXM*). The Adaptive Reward Function is analysed regarding the average travel time and the number of stops and produces similar results.

Table 5: Travel Time (in seconds) and Number of Stops results for Experiment #4 evaluating the Adaptive Reward Functions.

	Travel Time			**Number of Stops**		
Method	**Mean**	**Std**	**Max**	**Mean**	**Std**	**Max**
AdapVehNo	190.80	91.89	1381.00	1.16	0.55	7.00
VehNo	**188.79**	**88.54**	**702.00**	**1.15**	**0.54**	**4.29**
Thp	194.99	92.68	731.00	1.19	0.57	5.33
AdapDWTP	191.30	90.39	777.00	1.16	0.54	5.67
DWTP	189.21	91.66	768.33	1.16	0.56	11.00
FXM	194.85	152.57	1540.00	1.84	1.69	22.00

Source: Own author (2019).

Figure 5: Average Travel Time (seconds) (left) and Average Number of Stops (right) plots for the experiment comparing *AdapVehNo*, *VehNo*, *Thp* and *FXM*. Indicators have been aggregated at intervals of 600 time steps. Demand levels are plotted in gray.



Source: Own author (2019).

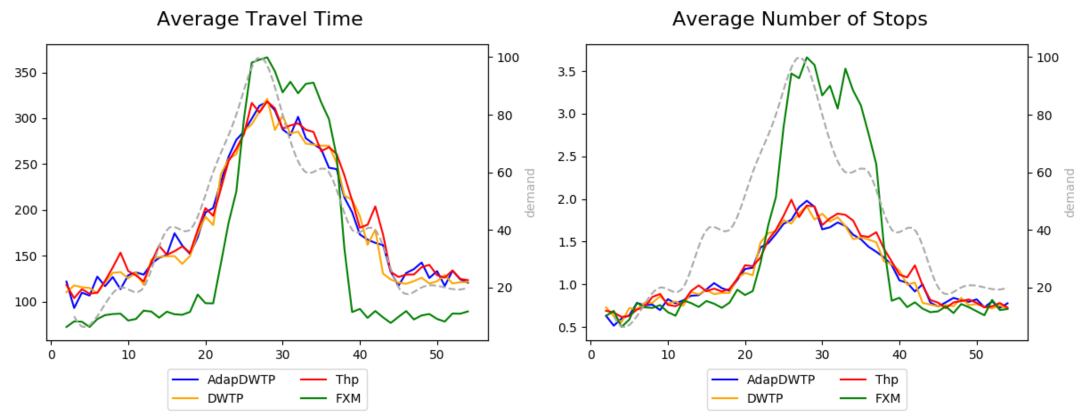
As Table Table 5 shows, using only the base reward function *DWTP* outperforms the adaptive reward *AdapDWTP* function by a small margin, then followed by the *Thp* reward function.

As in the previous experiment, the Q-learning methods outperform the *FXM* controller for high-traffic scenarios (when demand is above 70% of capacity) both regarding travel time and specially regarding number of stops (Figure 6). However, the Q-learning models do not show significant differences among themselves. Even though the *AdapDWTP* outperforms *Thp*, it still performs worse than using only *DWTP*. This suggests that, even though an adaptive reward function might work if its base reward functions work better in under-saturated or over-saturated conditions, this case does not present the expected results because the *DWTP* function seems to work better than *Thp* even for over-satured scenarios.

6 CONCLUSION

In the present paper, we review the use of Reinforcement Learning in the Traffic Signal Control domain, focusing on the possibility to orient the goals of the control system by customising

Figure 6: Average Travel Time (seconds) (left) and Average Number of Stops (right) plots for the experiment comparing *AdapDWTP*, *DWTP*, *Thp* and *FXM*. Indicators have been aggregated at intervals of 600 time steps. Demand levels are plotted in gray.



Source: Own author (2019).

the Reward Function. For this, we first introduced the subject by presenting the main advances in academic research for the past decade around the subject, focusing on optimization-based and RL-based methods and on the objective and reward functions that were used. Based on this review, we propose to evaluate some Reward Functions - taken from the literature along with a few adapted versions - with regards to how they perform under different saturation levels. Based on the notion that under-saturated and over-saturated traffic conditions have different characteristics (GORDON et al., 2005) and, therefore, different optimization objectives can be more or less suited under different conditions, we propose an adaptive reward function, which dynamically changes the reward function being used according to the saturation level of the road network.

To evaluate the reward functions, we conducted simulations using microscopic traffic simulator SUMO under different levels of demand. One of the main contributions of this paper is the code architecture used to run the simulation and control the Traffic Lights, as it was designed to be flexible and easily extensible. It allows to rapidly set up experiments using different implementations of the Q-learning elements (i.e. action definition, state representation, reward function, Q-learning algorithm) and also to combine different implementations to create a new method. In the future, we plan to extend our implementation to incorporate what is considered the state-of-the-art today in Q-learning-based Traffic Signal Control and release it as an open-source library, with the intention of allowing researchers to start experimenting new methods more quickly and help advance the research on the domain.

The results of our evaluation of the reward functions support the notion that there are some objectives that are more suited for under-saturated road conditions and, likewise, for over-saturated conditions. The experiments have shown that reward functions based on Average Vehicle Number and on Cumulative Delay are strong candidates for under-saturated conditions, while the reward function based on throughput (that tend to extend the stage time and minimize the lost time due to phase transition process) is a good candidate for over-saturated conditions.

On the other hand, the results of the evaluation of the Adaptive Reward Function have been inconclusive. We evaluated using both the Average Vehicle Number and the Cumulative Delay with penalty for wasted time as the reward function for under-saturated conditions and both experiments showed very similar levels of performance for the under-saturated, the adaptive and the over-saturated reward functions.

This could be due to one or a combination of the following factors: (1) a more complex reward function (such as the adaptive) could require more training or more advanced learning techniques; (2) the correlations between the state representation and the reward introduced by the dynamicity of the adaptive reward function may be too complex for a simple Q-learning model to learn; (3) this type of more complex reward function may require more information to be included in the state representation, so the model can learn the more complex correlations. We would like to try to solve the above issues as part of future research. Regarding point 2, we believe that a Deep Q-learning model would be able to capture correlations between state and reward that a simple Q-learning model is not able to. It would also allow to introduce more information in the State Representation without causing the problem of the state space becoming too large, as a Deep Q-learning model is able to approximate the Q-values function. In addition, we would like to conduct this evaluation with a more realistic road network and traffic demand definitions, then also applying indirect coordination techniques.

References

ABDOOS, M.; MOZAYANI, N.; BAZZAN, A. L. C. Traffic light control in non-stationary environments based on multi agent q-learning. In: **2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)**. [S.l.: s.n.], 2011. p. 1580–1585. ISSN 2153-0017.

ACCELBRAIN. **Reinforcement Learning library: pyqlearning**. 2019. Available in: <https://github.com/chimera0/accel-brain-code/tree/master/Reinforcement-Learning>. Access in: 5 nov. 2019.

ARAGHI, S.; KHOSRAVI, A.; CREIGHTON, D. Intelligent cuckoo search optimized traffic signal controllers for multi-intersection network. **Expert Systems with Applications**, v. 42, n. 9, p. 4422 – 4431, 2015. ISSN 0957-4174.

BI, Y. et al. Type-2 fuzzy multi-intersection traffic signal control with differential evolution optimization. **Expert Systems with Applications**, v. 41, n. 16, p. 7338 – 7349, 2014. ISSN 0957-4174.

BORSCHETTE, A. Report on the H2020RTR conference. n. November, 2018. Available in: <https://egvi.eu/wp-content/uploads/2019/02/Report-on-H2020RTR18-final.pdf>. Access in: 10 may 2019.

CHRISTOFA, E.; SKABARDONIS, A. Traffic signal optimization with application of transit signal priority to an isolated intersection. **Transportation Research Record**, v. 2259, n. 1, p. 192–201, 2011.

DHAWAN, R. et al. Mobility’s second great inflection point. n. February, p. 1–11, 2019. Available in: <https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/mobilitys-second-great-inflection-point>. Access in: 10 may 2019.

DUJARDIN, Y. et al. Multiobjective and multimodal adaptive traffic light control on single junctions. In: **2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)**. [S.l.: s.n.], 2011. p. 1361–1368. ISSN 2153-0017.

EL-TANTAWY, S.; ABDULHAI, B. An agent-based learning towards decentralized and coordinated traffic signal control. In: **13th International IEEE Conference on Intelligent Transportation Systems**. [S.l.: s.n.], 2010. p. 665–670. ISSN 2153-0017.

EL-TANTAWY, S.; ABDULHAI, B. Multi-agent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc). In: **2012 15th International IEEE Conference on Intelligent Transportation Systems**. [S.l.: s.n.], 2012. p. 319–326. ISSN 2153-0017.

EL-TANTAWY, S.; ABDULHAI, B.; ABDELGAWAD, H. 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**, IEEE, v. 14, n. 3, p. 1140–1150, 2013.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1. ed. [S.l.]: Addison-Wesley Professional, 1994. ISBN 0201633612.

GAZIS, D. C. Optimum control of a system of oversaturated intersections. **Oper. Res.**, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 12, n. 6, p. 815–831, dez. 1964. ISSN 0030-364X.

GORDON, R. L. et al. **Traffic control systems handbook**. [S.l.], 2005.

HU, H.-C.; SMITH, S. F. Bi-directional information exchange in decentralized schedule-driven traffic control. In: **Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems**. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018. (AAMAS ’18), p. 1962–1964.

KRAJZEWICZ, D. et al. Recent development and applications of SUMO - Simulation of Urban MObility. **International Journal On Advances in Systems and Measurements**, v. 5, n. 3&4, p. 128–138, December 2012.

LI, Y. et al. Multi-objective optimization of traffic signal timing for oversaturated intersection. **Mathematical Problems in Engineering**, v. 2013, p. 9, 2013.

MANNION, P.; DUGGAN, J.; HOWLEY, E. Learning traffic signal control with advice. In: **Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2015)**. [S.l.: s.n.], 2015.

MANNION, P.; DUGGAN, J.; HOWLEY, E. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In: ____. **Autonomic Road Transport Support Systems**. Cham: Springer International Publishing, 2016. p. 47–66. ISBN 978-3-319-25808-9.

PAPAGEORGIOU, M. et al. Review of road traffic control strategies. **Proceedings of the IEEE**, v. 91, n. 12, p. 2043–2067, Dec 2003. ISSN 0018-9219.

POHLMANN, T.; FRIEDRICH, B. Online control of signalized networks using the cell transmission model. In: **13th International IEEE Conference on Intelligent Transportation Systems**. [S.l.: s.n.], 2010. p. 1–6. ISSN 2153-0017.

PRABUCHANDRAN, K.; KUMAR, A.; BHATNAGAR, S. Multi-agent reinforcement learning for traffic signal control. In: **17th International IEEE Conference on Intelligent Transportation Systems (ITSC)**. [S.l.: s.n.], 2014. p. 2529–2534. ISSN 2153-0009.

REED, T.; KIDD, J. **INRIX 2018 Global Traffic Scorecard**. [S.l.], 2019. Available in: <http://inrix.com/wp-content/uploads/2019/02/Traffic-Scorecard-Infographic-2018_US-FINAL-v5.pdf>. Access in: 10 may 2019.

SHIRAI, T. et al. Multi-agent traffic light control framework based on direct and indirect coordination. In: **Proceedings of 7th International Workshop on Agents in Traffic and Transportation**. [S.l.: s.n.], 2012. p. 9–17.

STOKES, R.; BANKS, J. **Civil Engineering: Transportation Engineering Review**. [S.l.]: Kaplan Education, 2004. (Kaplan Aec Education Series). ISBN 9780793195626.

SUTTON, R. S.; BARTO, A. G. **Introduction to Reinforcement Learning**. 1st. ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981.

TEO, K. T. K.; KOW, W. Y.; CHIN, Y. K. Optimization of traffic flow within an urban traffic light intersection with genetic algorithm. In: **2010 Second International Conference on Computational Intelligence, Modelling and Simulation**. [S.l.: s.n.], 2010. p. 172–177. ISSN 2166-8523.

VÁZQUEZ, C. R. et al. Hybrid petri net model of a traffic intersection in an urban network. In: **2010 IEEE International Conference on Control Applications**. [S.l.: s.n.], 2010. p. 658–664. ISSN 1085-1992.

VILARINHO, C.; TAVARES, J. P.; ROSSETTI, R. J. F. Design of a multiagent system for real-time traffic control. **IEEE Intelligent Systems**, v. 31, n. 4, p. 68–80, July 2016.

VITI, F. **The Dynamics and the Uncertainty of Delays at Signals**. [S.l.]: TRAIL Research School, 2006. ISBN 9055840785.

WEBSTER, F. V. Traffic signal settings. **Road Research Technical Paper**, HMSO, 1958.

XIE, X. et al. Platoon-based self-scheduling for real-time traffic signal control. In: **2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)**. [S.l.: s.n.], 2011. p. 879–884. ISSN 2153-0017.

XIE, X.-F.; SMITH, S. F.; BARLOW, G. J. Coordinated look-ahead scheduling for real-time traffic signal control. In: **Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3**. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2012. (AAMAS '12), p. 1271–1272. ISBN 0-9817381-3-3, 978-0-9817381-3-0.

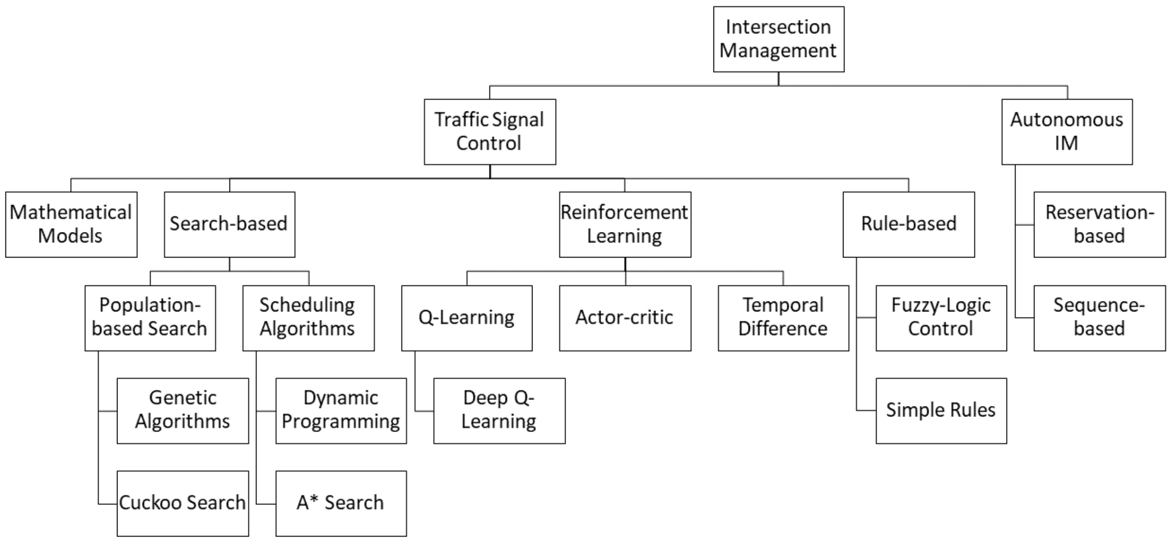
APPENDIX A – SIMULATION RESOURCES

- Github Repository: https://github.com/fabiovarisco/TCC-II
- Sumo Docker Image[6]: https://cloud.docker.com/repository/docker/fabiovarisco/sumo

[6]Docker Image which contains the simulation environment, including Python, SUMO and all the required libraries.

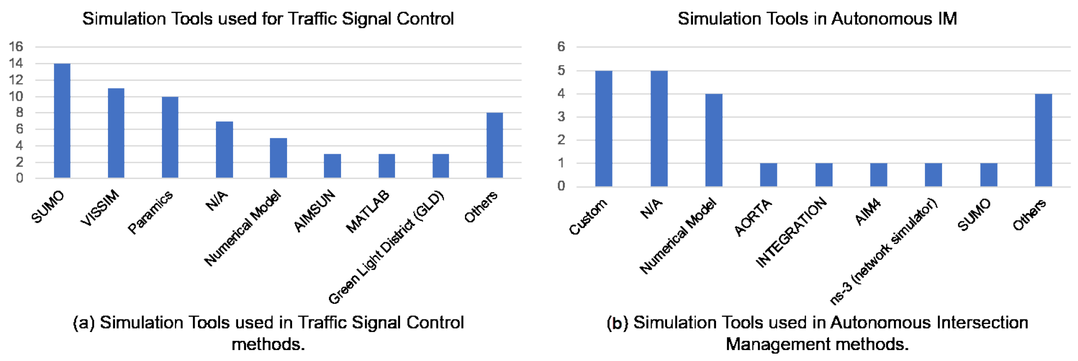
APPENDIX B – RELATED WORK RESEARCH

Figure B.1: Proposed taxonomy of Intersection Control methods.



Source: Own author (2019).

Figure B.2: Simulation Tools used by (a) Traffic Signal Control methods and for (b) Autonomous Intersection Management.



Source: Own author (2019).

Chart B.1: Overview of discussed papers describing the method used, input, output and optimization objective.

Paper	Method	Output	Inputs	Objective
Vázquez et al. (2010)	Mathematical Model	Green times for all phases at beginning a time horizon of 1200 seconds.	Queue Lengths	Minimize the queue lengths

Christofa and Skabardonis (2011)	Mathematical Model	Green times for all phases at beginning of cycle	Number of passengers (from both cars and buses)	Minimize passenger delay and residual queue
Shirai et al. (2012)	Mathematical Model	1st layer - Cycle length (Webster formula) and green time splits for each phase (spring model) 2nd layer - Offset (through a direct coordination approach when it detects congestion)	Current traffic flow	Clear queues
Dujardin et al. (2011)	Mixed Integer Linear Programming Optimization	Traffic Lights Switch-over dates		Cumulative Waiting Time, Number of Stops and Total Delay of Buses
Bi et al. (2014)	Fuzzy-Logic Control (FLC) optimized by Differential Evolution (DE) + Second layer to prevent sending vehicles to congested areas	Next phase to be effective and its green time	Maximum queue length and second maximum queue length	Minimize Vehicle Delay
Araghi, Khosravi and Creighton (2015)	(1) Adaptive Neuro-Fuzzy Inference System(2) Neural Network Controller(both optimized by using Cuckoo Search)	Green time for each phase at the beginning of the phase	Queue Lengths and Number of Vehicles coming from neighbouring intersections	Minimize Vehicle Delay

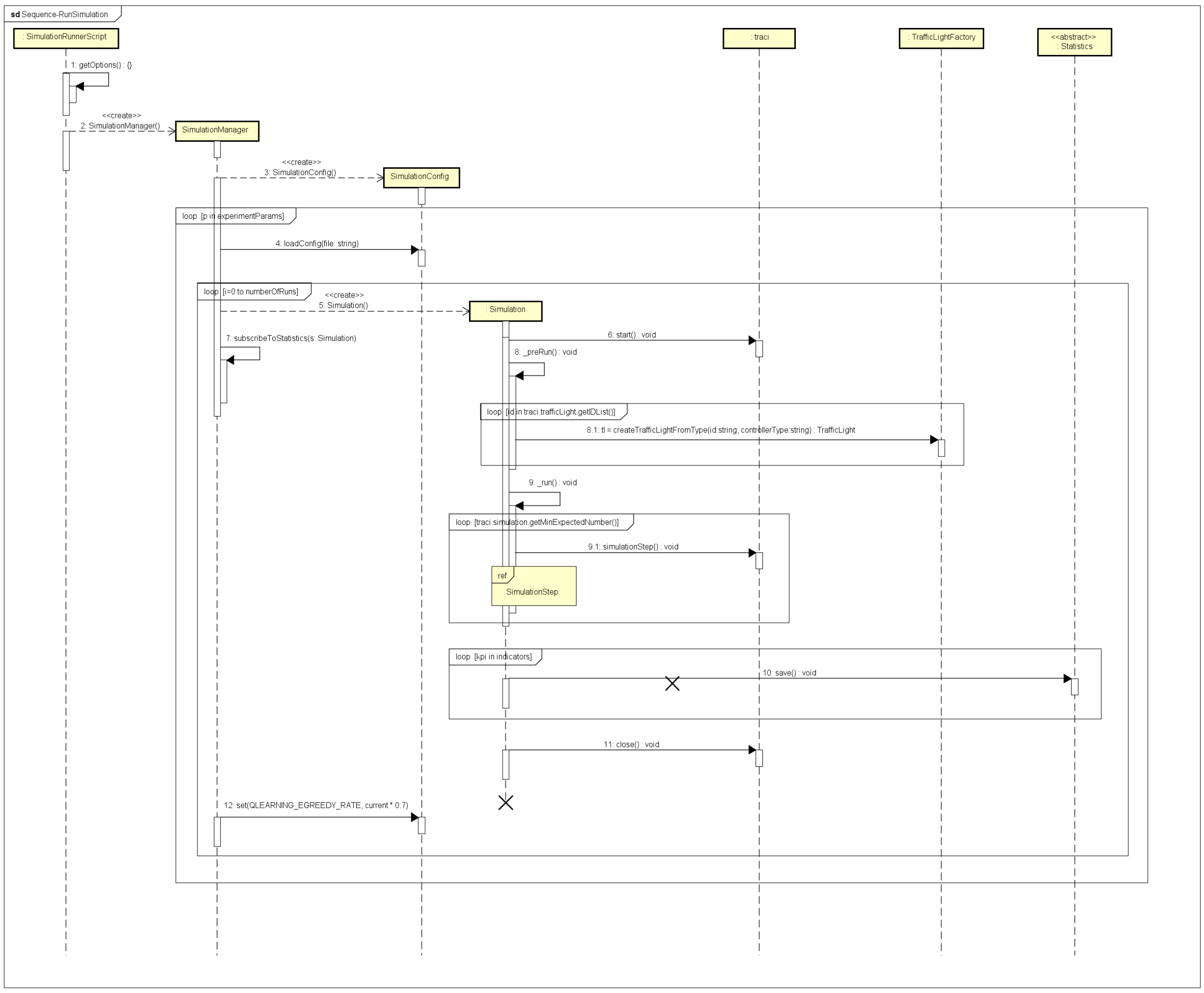
Teo, Kow and Chin (2010)	Genetic Algorithm	Green times for all phases at beginning of cycle	(1) Queue Lengths(2) Queue Lengths + Forecast for phases 2 and 3	Minimize queue length
Li et al. (2013)	Genetic Algorithm	Green times for all phases at beginning of cycle	Queue Lengths	Maximize throughput while minimizing queue ratio
Xie et al. (2011)	Scheduling Algorithm	At each decision point, determine the extension time for the current phase or advance to next by choosing 0 (zero)	Aggregate incoming traffic into platoons (which are treated as jobs)	Minimize delay
Xie, Smith and Barlow (2012)	Scheduling Algorithm	Same as Xie et al. (2011)	Same as Xie et al. (2011), also considering the expected outflows from upstream intersections	Same as Xie et al. (2011)
Hu and Smith (2018)	Scheduling Algorithm	Same as Xie, Smith and Barlow (2012)	Same as Xie, Smith and Barlow (2012), also considering congestion levels from downstream intersections	Same as Xie, Smith and Barlow (2012)
Abdoos, Mozayani and Bazzan (2011)	Q-Learning	Cycle time is fixed, output is the split of green time among phases	Approaching links indexes ordered by queue length descending	Minimize the average queue length

Prabuchandran, Kumar and Bhatnagar (2014)	Q-Learning (ϵ-greedy and Upper-Confidence Bounds (UCB))	Green time for each phase at the beginning of the phase	Queue Lengths discretized in low, medium and high + Index of the next green phas	Minimize Long Term Average Delay
El-Tantawy and Abdulhai (2010)	Q-Learning (ϵ-greedy)	Next phase (if phase is the same as current, extend it by 1 sec)	(1) Number of arriving vehicles for current phase + queue length for red phases(2) queue length (3) summation of cummulative delay	Maximize the savings in total cumulative delay between two decision points
El-Tantawy and Abdulhai (2012), El-Tantawy, Abdulhai and Abdelgawad (2013)	Q-Learning (ϵ-greedy)	Next phase (if phase is the same as current, extend it by 1 sec)	Queue Lengths	Maximize the savings in total cumulative delay between two decision points
Mannion, Duggan and Howley (2015)	Q-Learning with Potential-based advice (PBA)	Next phase (if phase is the same as current, extend it by 1 sec)	Queue Lengths	Maximize the savings in total cumulative delay between two decision points

Source: Own author (2019).

APPENDIX C – DIAGRAMS

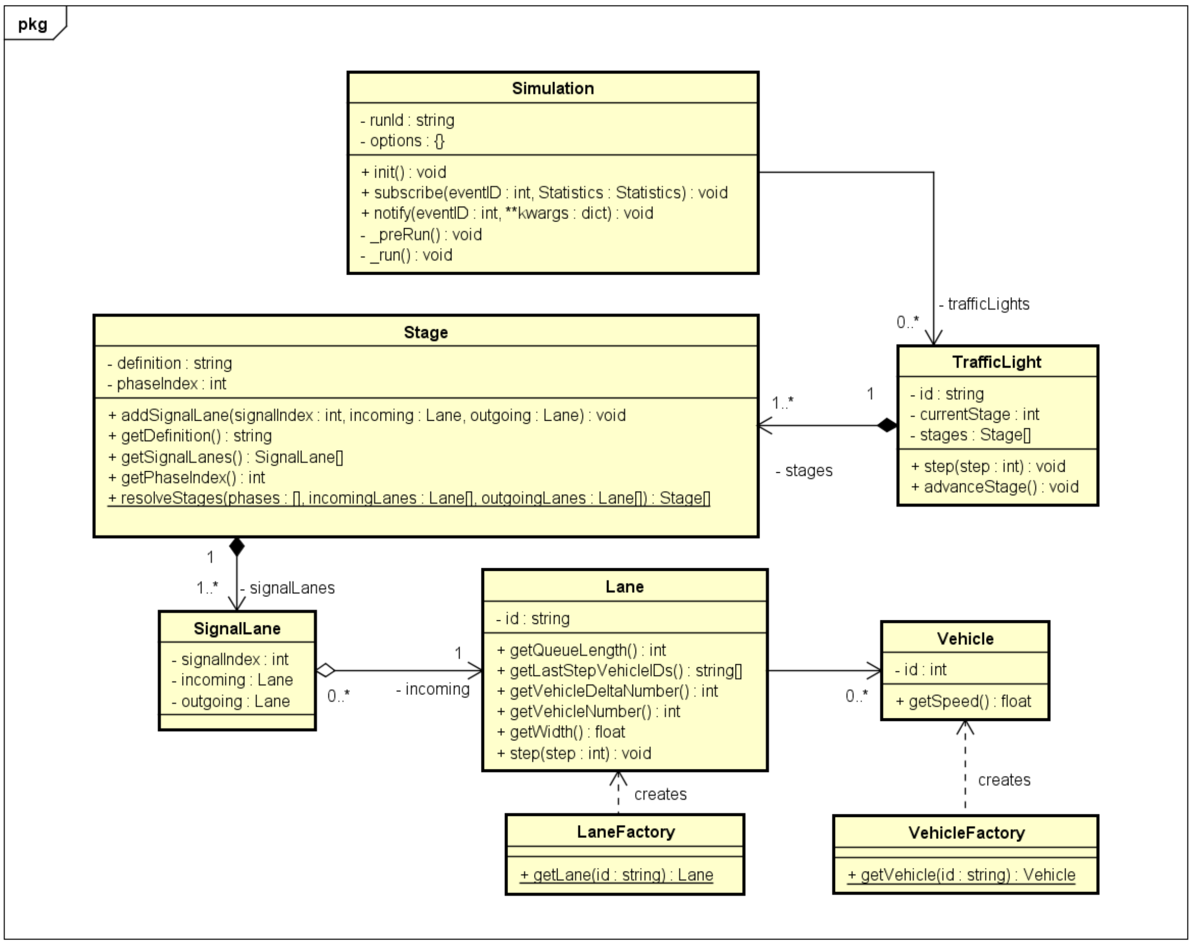
Figure C.1: Sequence diagram showing the process of an experiment execution.


sd Sequence-RunSimulation
: SimulationRunnerScript
: traci
: TrafficLightFactory
<<abstract>> : Statistics
1: getOptions() : {}
<<create>> 2: SimulationManager()
SimulationManager
<<create>> 3: SimulationConfig()
SimulationConfig
loop [p in experimentParams]
4: loadConfig(file: string)
loop [i=0 to numberOfRuns]
<<create>> 5: Simulation()
: Simulation
6: start() : void
7: subscribeToStatistics(s: Simulation)
8: _preRun() : void
loop [id in traci.trafficLight.getIDList()]
8.1: tl = createTrafficLightFromType(id:string, controllerType:string) : TrafficLight
9: _run() : void
loop [traci.simulation.getMinExpectedNumber()]
9.1: simulationStep() : void
ref
SimulationStep
loop [kpi in indicators]
10: save() : void
11: close() : void
12: set(QLEARNING_EGREEDY_RATE, current * 0.7)

Source: Own author (2019).[7]

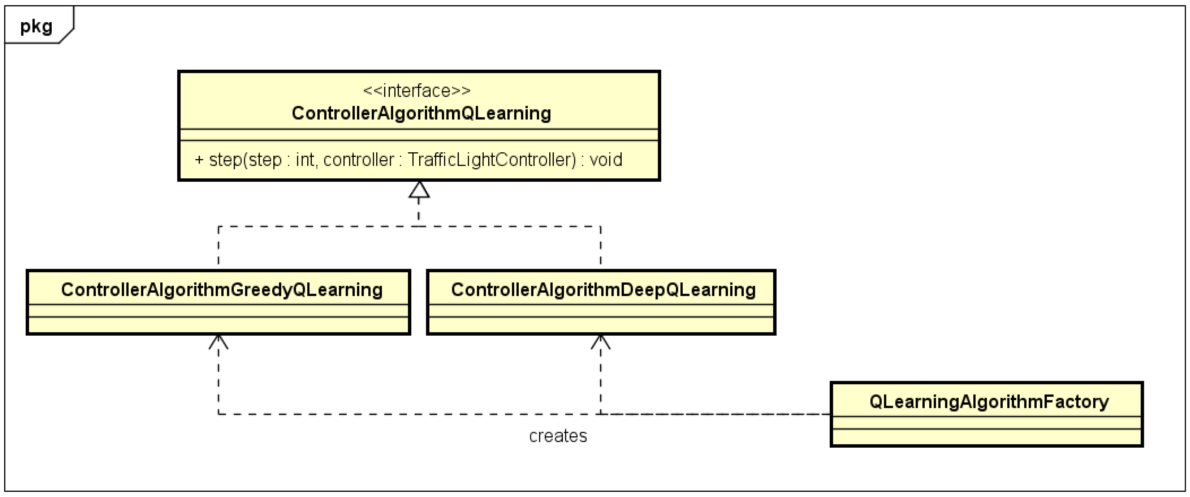
[7]Diagram created using Astah UML.

Figure C.2: Class diagram of the Simulation Entities module.


pkg
Simulation
- runId : string
- options : {}
+ init() : void
+ subscribe(eventID : int, Statistics : Statistics) : void
+ notify(eventID : int, **kwargs : dict) : void
- _preRun() : void
- _run() : void
- trafficLights
0..*
Stage
- definition : string
- phaseIndex : int
+ addSignalLane(signalIndex : int, incoming : Lane, outgoing : Lane) : void
+ getDefinition() : string
+ getSignalLanes() : SignalLane[]
+ getPhaseIndex() : int
+ resolveStages(phases : [], incomingLanes : Lane[], outgoingLanes : Lane[]) : Stage[]
1..*
1
- stages
TrafficLight
- id : string
- currentStage : int
- stages : Stage[]
+ step(step : int) : void
+ advanceStage() : void
1
1..*
- signalLanes
SignalLane
- signalIndex : int
- incoming : Lane
- outgoing : Lane
1
0..*
- incoming
Lane
- id : string
+ getQueueLength() : int
+ getLastStepVehicleIDs() : string[]
+ getVehicleDeltaNumber() : int
+ getVehicleNumber() : int
+ getWidth() : float
+ step(step : int) : void
0..*
Vehicle
- id : int
+ getSpeed() : float
creates
creates
LaneFactory
+ getLane(id : string) : Lane
VehicleFactory
+ getVehicle(id : string) : Vehicle

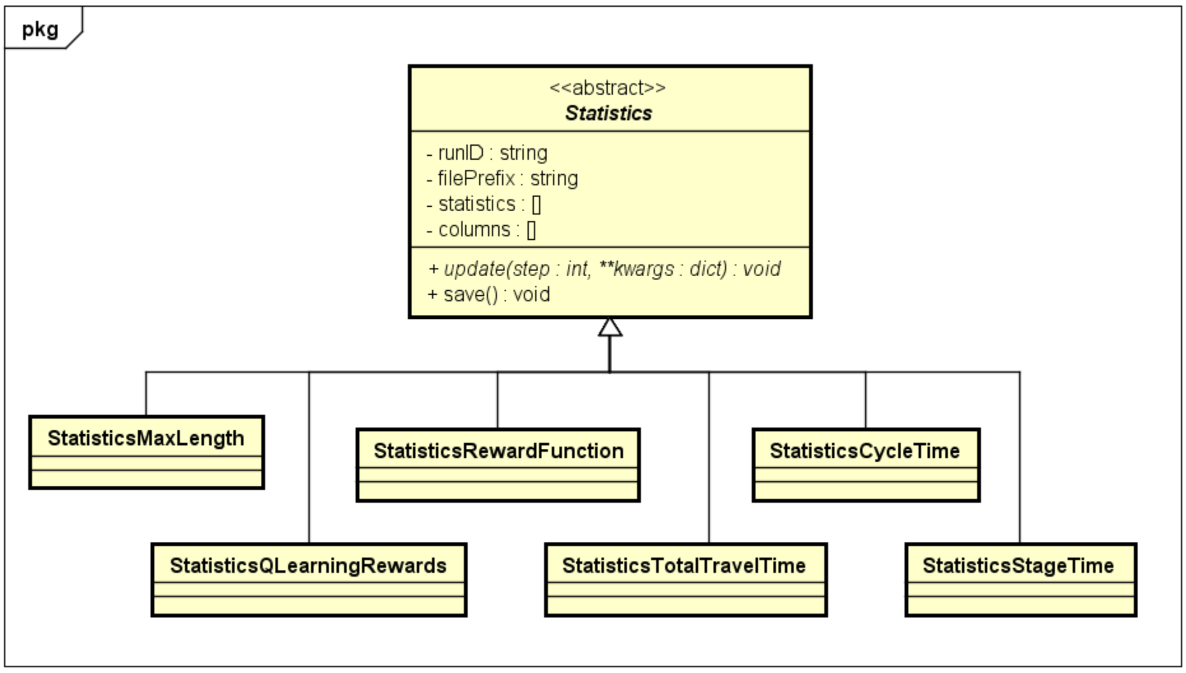
Source: Own author (2019).[7]

Figure C.3: Class diagram of the Q-learning Algorithms module.


pkg
<<interface>>
ControllerAlgorithmQLearning
+ step(step : int, controller : TrafficLightController) : void
ControllerAlgorithmGreedyQLearning
ControllerAlgorithmDeepQLearning
QLearningAlgorithmFactory
creates

Source: Own author (2019).[7]

Figure C.4: Class diagram of the Statistics module.



Source: Own author (2019).[7]

APPENDIX D – ADDITIONAL EXPERIMENT RESULTS

Table D.1: Hyper-parameters tuning - Learning Rate - Travel Time results

	Episode 5			**Aggregated Results**		
Learning Rate (α)	**Mean**	**Std**	**Max**	**Mean**	**Std**	**Max**
1e-05	261.61	167.17	1294.00	254.53	136.83	1341
1e-04	271.06	174.43	1164.00	250.00	120.41	1164
1e-03	**257.49**	**153.42**	**981.00**	**247.05**	**118.54**	**962**
1e-02	272.39	180.70	1081.00	253.32	123.56	916
1e-01	257.51	160.60	1072.00	249.09	121.19	981.5

Source: Own author (2019).

Table D.2: Hyper-parameters tuning - Gamma Rate and ϵ-greedy rate - Travel Time results

		Episode Run 5			**Aggregated Results**		
Gamma Rate (γ)	**ϵ-Greedy Rate (ϵ)**	**mean**	**std**	**max**	**mean**	**std**	**max**
0.6	**0.6**	**256.74**	179.32	1372.00	247.82	117.63	1123.00
0.7	**0.6**	261.12	174.02	1184.00	248.35	113.91	1093.00
0.8	**0.6**	257.08	174.64	1543.00	247.96	117.08	1075.00
0.9	**0.6**	262.28	173.64	1294.00	**246.58**	115.25	1061.00
0.6	**0.7**	257.44	171.36	1550.00	249.46	117.58	1082.00
0.7	**0.7**	267.46	169.33	1205.00	251.78	120.47	1009.00
0.8	**0.7**	263.56	**163.28**	**1007.00**	252.04	121.91	1091.50
0.9	**0.7**	273.07	172.24	1139.00	251.51	**111.12**	**859.00**
0.6	**0.8**	266.79	181.24	1375.00	249.19	120.56	1445.00
0.7	**0.8**	259.93	167.69	1189.00	246.80	112.48	1144.00
0.8	**0.8**	266.25	174.25	1234.00	249.67	114.36	1056.00
0.9	**0.8**	279.02	171.34	1355.00	251.98	116.64	1187.00

Source: Own author (2019).