UNIVERSITAT POLITÈCNICA DE CATALUNYA

Master in Artificial Intelligence

Final Project

Experiments with reinforcement learning on traffic conditions changes

Paul Gazel-Anthoine paul.gazel-anthoine@estudiantat.upc.edu

April 20, 2022

Abstract

Speed limits are an important instrument for urban planners to regulate traffic in dense urban areas. This master thesis attempts to study strategies for such tool through the training and study of Reinforcement Learning models on simulation programs. This project builds a framework to implement a Reinforcement Learning method, Deep Q-Learning, on the SUMO simulation tool and considers two networks, one built on the model of the Eixample neighborhood in Barcelona, on which to apply such tools. Different design choices in the Reinforcement Learning paradigm, in particular for the state representation and reward function, are discussed, implemented and evaluated. Computational cost, realism of the simulation and other important concerns are discussed at length.

Contents

1 Introduction **3**
1.1 Project presentation 4
1.2 Project structure and requirements 5
1.2.1 Traffic simulator 5
1.2.2 Reinforcement Learning 6
1.2.3 Connection 7
1.3 Related work 7

2 Simulation **8**

3 Reinforcement Learning **12**
3.1 Deep Q-Learning 12
3.2 State representation 13
3.2.1 Image Observation 13
3.2.2 Edge information 14
3.2.3 Edge analytics 14
3.2.4 Edge vehicles 15
3.3 Reward function 15

4 Implementation and Execution **16**

5 Experiments **17**
5.1 Small/custom network 18
5.2 Big/Real network 19

6 Conclusion **22**

1 Introduction

Traffic in modern cities is a key challenge for urban planners. Car commutes represent an important share of daily commutes for most European cities, including cities that have efficient public transport networks. The transit capacity of a city road network is limited and usually very hard to increase, as space is scarce in cities. The growth of cities, and creation of suburban areas with reduced public transport, and the standardization of car ownership has, in the past decades, put a lot of tension on urban road networks. Most modern European cities frequently experience traffic congestion at peak hours, and small localized incidents can often lead to drastic consequences on important parts of the network. The reliance on cars for mobility is likely to be a long-term characteristic of modern cities in the next decades[1], despite efforts to increase public transport. City infrastructures are not always adapted to the traffic they experience, especially in city centers. Such problems have an important impact on life quality in urban areas[2]. Not only does it impact the life of the commuters stuck in traffic on a regular basis, it also affects the general life quality of city inhabitants. The environmental cost (air quality, noise polution...) of cars in urban areas is substantial, and is significantly increased when the network is unable to avoid global perturbations such as traffic jams[1].

Urbanism requires studying and carefully considering traffic dynamics and available option to mitigate potential problems. Both the design of the city network and the handling of this network rely on a comprehensive understanding of transit in the city. Building efficient system for traffic is a complex task with multiple constraints and needs. Structural changes to the network are limited by time, space and budget. Traffic handling must be coherent, readable and answer the needs of multiple populations and individuals. Its efficiency is very dependent on specificities of the city and its inhabitants. While urbanism is an old discipline, the scale of modern cities have brought the problem to a level of complexities at which traditional solutions struggle to offer satisfying results.

Technological advances are offering new tools and analysis to help build efficient systems for regulating traffic in urban areas. The collection of data on road transit has helped document the issue. Many cities collaborate with companies to collect data on their road network in the objective of having a better understanding of its dynamics and accordingly improve their regulation strategies. Additionally, models have been build to reproduce and study the phenomenon. Scientific research in different fields, from public policy to statistics and machine learning, using this data and models, have provided advice and conclusions to help inform such choices.

Artificial Intelligence offers multiple advantages in studying such problem, as it is able to tackle problems with high complexity without requiring too much simplification or modelling assumptions. The Reinforcement Learning framework is particularly adapted as it only requires the design of a suited reward function and can be trained with modeling tools. AI can adapt to city and population specificities, create a flexible system that can adapt to different situations and even, if required, use live data collection to improve its decisions. While, due to the multiple constraints and rigidities the issue includes, it is unlikely that unsupervised AI models will dictate traffic regulations, their behavior and choices can bring important value to urban planners.

[1] https://www.eae.es/actualidad/noticias/barcelona-incrementa-el-uso-del-coche-hasta-el-47-desde-el-inicio-de-la-pandemia

[2] https://www.irtba.org/QualityofLife

Figure 1: Traffic in Avinguda Paral.lel in Barcelona

1.1 Project presentation

This project aims at implementing a Reinforcement Learning approach for regulating global speed limit in urban space under network perturbation. While accidents are not accounted for in this framework, events leading to the cut-off of a roads such as a planned protest are. Here the RL model should react to such events and adapt accordingly, taking into account the state of the network.

This study is centered around the neighborhood Eixample in Barcelona, considering portions of its network, or similarly dense and connected networks. While, due to computational limits of the simulation model, the study could not cover the entire neighborhood, the chosen areas are meant to be representative of it. The tension put on the network is meant to reflect usual traffic situation in Barcelona, i.e. a big European city with consequent urban infrastructure. Therefore networks considered are characterized by a mix of big avenues and small roads with many crossings and traffic lights, often with some type of synchronization.

The main goal of the approach is to create a model able to prevent or reduce traffic jams. Urban planning requires taking into account multiple sometimes conflicting objectives, and while this objective seems sensible, it does require additional definition and design choices. Building a usable reward function for RL implementation implies choosing an adequate metric that reflects the objective. Multiple options are possible and may result in slightly different strategies.

The model only sets global speed limits, a simple traffic regulation instrument. Perturbation of the network are not known before perturbation by the model but are known to it when they happen. The

entire studied network is available to the RL model, although such information may be preprocessed. The state representation thus potentially includes all information on the roads and crossings in the networks, along with cars' position and speed.

The aim of the project is to study how the speed limit tool may be used to prevent traffic jams. The RL model, built and trained in a controlled simplified environment, should have explored multiple strategies and selected an optimal strategy. While real-life situation and implementation have to answer and adapt to other constraints, this experiment aims at exploring new ways in which an important tool of traffic regulation can be utilized.

1.2 Project structure and requirements

Fulfilling the objective fixed required a model able to simulate a city traffic scenario and a tool to implement and train a reinforcement learning agent using these simulations.

The project can thus be divided into tree equally important part: first, the traffic simulator that needs to simulate as close as possible to reality the traffic within a network that replicates the desired city, but should also have the necessary tools for the reinforcement learning part to interact with the simulation (both gathering information for the agent as well as applying the action decided by it). The second part consist of the reinforcement learning side, this section must implement a chosen reinforcement learning algorithm that, with the information provided by the traffic simulator, is able to learn an optimal or nearly optimal policy that solves the problem, i.e. consistently obtains high reward. Finally, the last part represents the connection between the two, although seemingly less important, this part is essential for the agent to learn an optimal policy. Since learning is on-policy, the agent has to be able to correctly manipulate the traffic simulator.

1.2.1 Traffic simulator

There are two main traffic simulators that seemed to fit the project requirements: Aimsum and SUMO. Aimsum is a private company (created by UPC students) that, as indicated in their website, is used by a very large number of universities, public transport entities, as well as companies, to simulate urban traffic. It is cited as the state-of-the-art model for such uses. It is however not open-source, nor free. Therefore it did not represent a fit for this project.

The second option is SUMO, *Simulation of Urban MObility*, a modeling tool that just as Aimsum, fulfills the requirement for this project. Additionally, it is open source. SUMO is developed by the Institute of Transportation Systems at the German Aerospace Center and its purpose is to support the traffic research community. It is designed as a multi-modal traffic simulation package that aims to be fast and portable. It comes with a large range of tools from network editing tools to traffic generation scripts. Such tools come in very handy for this project as described below.

While SUMO is not specifically made for Artificial Intelligence and particularly Reinforcement Learning, it is used in this area of research. There thus exists several projects and libraries that are designed to implement and adapt SUMO to an Reinforcement Learning environment. Most Reinforcement Learning projects utilizing SUMO are however related to traffic simulation in either traffic lights control or autonomous driving.

Autonomous driving project are of no help in this work as they tackle the simulation from a local perspective around the controlled car. Traffic lights control projects, on the other hand, have a very

similar approach to ours. Libraries such as *SUMO-RL* [2] implement interfaces to create a Reinforcement Learning environment for traffic signal control. They however do not provide any tools for speed limit controlling or event generation, which make their use impossible. Nevertheless, these libraries, usually open source, provide a great help when implementing a solution for this project as their approach and code can be replicated and adapted to the speed limit scenario.

Additional tools, such as *Flow*[3], may have been useful too. Flow was created by a team from *UC Berkeley* and aims at creating a Deep Reinforcement Learning framework for mixed autonomy traffic. It works with SUMO (As well as Aimsum) and provided tutorials as well as a entire framework to use SUMO as a Reinforcement Learning environment.

Flow brings considerable benefits to the table. Including it in this work could have simplified some of the steps of the project. However when moving from small RL experiments, meant to familiarize myself with the tool, to the real scenario of this project some serious problems appeared. The main reason behind it is that Flow uses an integrated older version of SUMO (1.1.0) while SUMO itself is now at version 1.12.0. This meant that a large amount of SUMO new content and fixes weren't in Flow and thus the creation of special events as required by our project as well as tools to access the simulation data were not up to date and didn't match the content of SUMO documentation. Therefore, after having learned how to use the tool and trying an implementation using Flow, I decided to abandon the idea of using Flow and to implement myself the connection between Reinforcement Learning and SUMO.

1.2.2 Reinforcement Learning

Reinforcement Learning is a sub-field of unsupervised Machine Learning. The Reinforcement Learning paradigm considers an agent following a trial and error strategy to explore possible policies. A reward mechanism is implemented in the framework upon which the agent bases its evaluation of states and actions. The agent balances out current and future rewards which requires building an understanding of the inherent value of state configurations. Reinforcement Learning builds a paradigm that combines learning values of the a priori unknown state space and optimization of expected reward, i.e. explore and exploit. Research has designed two related tools to capture such information, the value function and the Q-functions. The Q-function represent value of an action for a certain state. Standard Reinforcement Learning strategies are designed around building a correct and efficient way to estimate such function, in particular in promising situations, and recover an optimal policy from it. The challenge often centers around fine-tuning this trade-off and avoid unhelpful computations.

Deep Q-Learning is a state-of-the-art method for Reinforcement Learning, built upon an important RL literature ([4], [5]). The Q-function is estimated by a neural networks, of which the method attempts to learn the parameters. Parameters are learned through minimization of a loss function consisting of the distance between the learned value and a target built with the received reward and the previous estimate for an batch of observed transitions. Optimal policy can then be easily retrieved from the Q-function. This method was designed for Q-learning in real-life RL problems, with potentially complex Q-functions. While there have been modifications proposed to tackle some of the pitfalls of the method ([6]), it is commonly used in RL projects, as it offers good performance in most RL scenarios while being scalable. It is implemented in open-source, well-documented and easily-usable machine learning libraries such as Tensorflow in Python. DQN is a state-of-the-art usable and flexible algorithm for Q-Learning that fits the needs of this project.

1.2.3 Connection

The two parts previously described (in 1.2) are independent. However in order for the project to work, the agent from the Reinforcement Learning part needs to learn from the simulator and therefore the connection between the two is important. The agent will learn the correct policy by following a trial and error strategy, keystone of Reinforcement Learning. This means that the learning agent has to be able to manipulate the simulator: launching and restarting it when necessary, for exploration, but also send information from one side to another. The simulator needs to know the actions taken by the agent in order to apply them, and the agent needs to know what happens in the simulator in order to decide what action to take.

The necessary connection between the two brings two important question and problem of any Reinforcement Learning project: What information does the agent needs to understand fully the simulation for it to be able to learn an efficient policy, the state representation, and how to define if the action taken has had a positive or negative impact on the environment, which is described in RL by the reward function. Those two design problems are key aspects of any Reinforcement Learning project as they have a direct impact on performance. If the state representation does not contain all the necessary information to understand the simulation, the agent will be unable to learn an efficient policy for all states. If the reward function does not correctly evaluate the state following the agent action, the agent cannot learn if taking this action lead to an improvement or not, and therefore won't be able to deduce the optimal policy.

State representation and reward function in RL are design choices. They are subject to multiple trade-offs. Their design greatly influences learning capacity and speed, along with greatly influencing which strategy might be chosen. There may not be any reward function allowing proper learning that perfectly reflecting the intended objective. Similarly state representation should include all useful information, to ensure the agent has access to it, but avoid including useless information, as it may lead to reduced learning speed and could cause noise overfitting. Multiple design choices are tried out in this project. The different designs considered for state representation and reward function are discussed later on.

The state representation and the reward function are the two main factor of the communication from the environment (Here the traffic simulator) to the agent. However, there is also a need for communication in the other direction. The simulator needs to be able to modify the simulation according to the agent actions. The possible actions of the agent have to be defined in advance. As explained before, in this scenario, the agent only has control over the speed limit of the simulation.

1.3 Related work

There are several projects using Reinforcement Learning combined with SUMO. The libraries mentioned before all have examples as well as entire projects built using them. As mentioned before, they mainly fall into two categories: autonomous driving and traffic lights control. Only the ones from the second category are helpful for the structuring of this project.

A project tackling a similar problem is the thesis *Learning Control Policies in Smart Cities from Physical Data* by Mykhaylo Marfeychuk [7]. This work uses real world data of traffic and traffic-related emissions in the city of Trondheim in Norway. It uses it to create a Reinforcement Learning based solution that is able to stop the traffic in some regions of the city in order to reduce the traffic-related emissions. In his environment, blocking the access to some streets and forcing the use of rerouting algorithm for the vehicles that had in their path the newly blocked roads as in SUMO the routes are precomputed, are implemented features, close to what is implemented in this project.

2 Simulation

The objective of the simulation is to be as faithful as possible to the reality so that results and policy obtained could be used in real life scenarios. In order to achieve that some assumptions must be made even though these may decrease the performance of the model.

Firstly I decided to separate the simulation step and the Reinforcement Learning step. The simulation step represents the SUMO reinforcement step and is by default equivalent to 1 seconds. The Reinforcement Learning step represents the application by the agent of an action into the environment. In this scenario, the agent's actions are the change of the speed limit and it makes little sense to change the speed limit every second. I chose to allow changes in speed limit every 10 minutes, which seems possible to implement in the real world, as it gives the infrastructure and the drivers time to adapt. Therefore, a Reinforcement Learning step is equivalent to 600 environment steps which brings additional complexity on how to define both the state representation as well as the reward function. Nevertheless while in the main road network (Representing part of the Eixample neighborhood) we decided to use this ratio for the smaller network we decided to go for a lower one of 1 action every minute (1 Reinforcement learning step equals 60 environment steps). In this scenario vehicles trips tend to last less than 10 minutes which would make the configuration entirely new (no car in common with the previous step) at each Reinforcement Learning step under a 10 minute window for an action. In such scenario with too high of a window, it is unlikely that the speed limit changes would hold any relation to the current state of the network.

Another self-imposed restriction is that the speed limit is changed globally, meaning that the whole road network has the same speed limit. This significantly reduces our agent's room to maneuver, however it also considerably reduces the complexity of the project as it greatly reduces the action space size, i.e. the combinations of possible action the agent can perform at each time step. There is also a restriction on which values the speed limit may take, in order to be as realistic as possible the range of available values goes from 10km/h to 50km/h with a 10km/h step. The upper limit was chosen because its the current speed limit and there would be no point in going above it, and the lower limit is set as there is no point in giving the drivers a lower one which is impossible to follow by car. The step is chosen because intermediate values beyond multiples of 10 are very uncommon. The action space is thus discrete and not continuous which further reduces the complexity. All these values, upper bound, lower bound and step can be easily modified in the XML configuration file as explained in section 4.

Originally the simulations were to be done on two networks. First in a small network made manually in order to try out the simulations and confirm that everything works smoothly. In order to create custom networks, SUMO comes with Netedit a tool that allow the creation and customization of networks. It was used to create our fist network, shown in figure 2.

Figure 2: First small network created with Netedit

This network serves a sandbox purpose, to test and ensure proper functioning on a small network requiring much less training time and computational capacities. It is also easier to evaluate on this first environment the correctness of the computation of the different state representations or reward functions as well as the overall functioning of the program. However it is very uncertain that there something to learn for the agent in such a small environment. Speed limit changes are meant to have a global impact which may not have great importance for small networks.

The second network is the target network of the study, the Eixample neighborhood, shown in figure 3. SUMO provides a tool to import a network directly from OpenStreetMap called osmWebWizard. This tool allows to create a SUMO network, editable later on with Netedit from anywhere on earth. I ended up not importing the entire of the Eixample neighborhood as I considered it too big and only took the part on the left of the street *Passeig de Gràcia*, which cuts it in half. After importing the network from OpenStreetMap a post processing is required as a lot of networks parts need readjusting to make it work properly. Adjacent junctions need to be merged in some scenarios, tunnels needs to be removed. This process is discussed in further length in section 5.

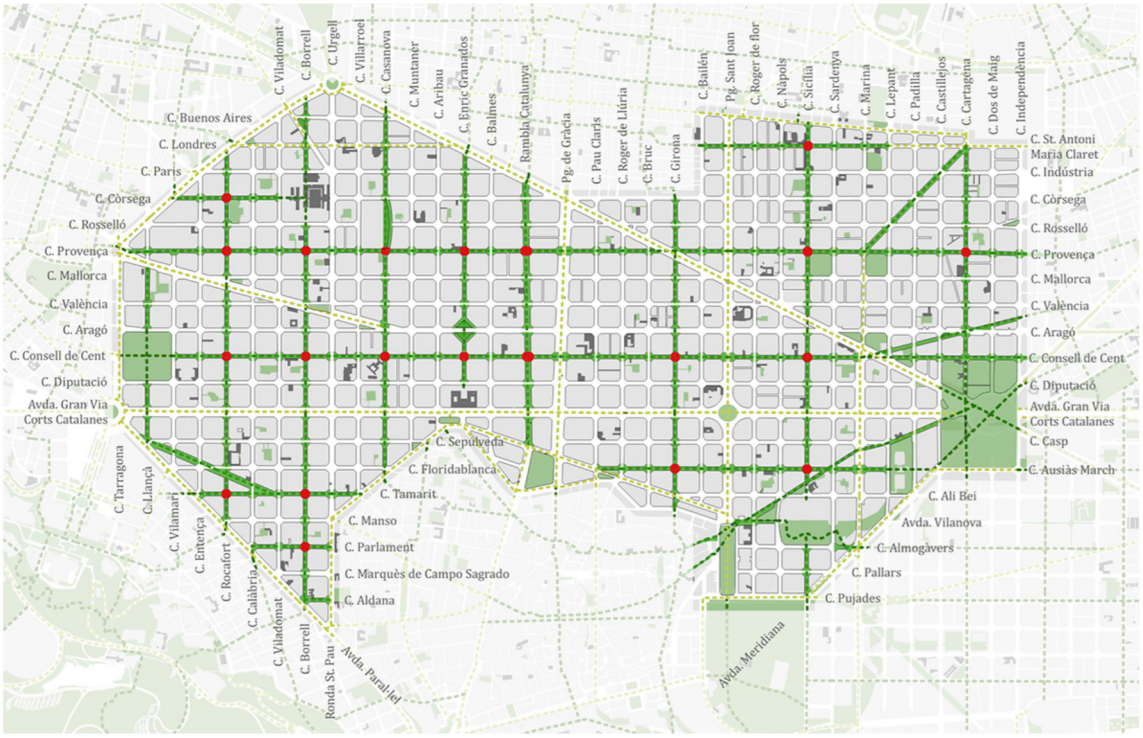


Figure 3: Map of the Eixample neighborhood

Another important aspect that needs to be defined within the simulation are the cars and their trips. The goal of our project is to reach a state where, without perturbation, the traffic is dense but not saturated and only when the perturbation occurs traffic jams may form. SUMO provides several ways to generate these, from creating flows within the simulation (Large number of cars going from one zone of the network to another one) to simply random trips. In order to stay as realistic as possible the choice was made to use the randomTrips tool. The trips used in this project are however not completely random and some parameters have been used to ensure more realistic trips:

1. The *min-distance* parameter is used to ensure that the trips have a decent length. The default value is set to 500m in the large network and 10m for the small one.

2. The *fringe-factor* parameter increases the probability that trips start or end at the extremities of the network, this makes a lot of sense as our network is situated within a city and is not a closed environment so in the real world the majority of cars within Eixample come from further away. The default value was set to 10 meaning there is 10 times more chances that the edges selected is one of the ends of the network.

3. The *period* parameter is used to influence the density of cars in the network, through changing the numbers of cars to be added to the network. It represents the arrival rate. The lower it is the more cars will be added to the simulation at each simulation step. From experiments a value around 5 creates quite dense traffic but still acceptable. The default value was thus set to 5.

All these value can easily be changed in the XML configuration file.

The vehicle type of cars in the network also has to be defined. SUMO lets the user define its own vehicle type allowing to change parameters such as the acceleration, the car behavioral model and other

key attributes. The default car used for SUMO is, according to them, a quite accurate standard car and behaves in a realistic way. One important aspect in the definition is the behavior of the car with respect to speed limits. SUMO's default car type follows a normal distribution when facing the speed limit, meaning 95% of the vehicles drive between 80% and 120% of the legal speed limit. In this project the default SUMO vehicle type is used.

Another important aspect of the simulation is how long does the simulation last and when does the special event, i.e. road block, is triggered. For the real-life big scale scenario, the decision was made to go for simulations lasting 2 hours, meaning cars will keep appearing within the simulation in the first 2 hours. For the small network a 30 minutes window seemed more accurate and was thus selected. Finally an end step is required in the scenario where a simulation doesn't end (There are still cars in the simulation), mostly for safety measure. 3 hours was chosen for the big network and 45 minutes for the smaller one. These values can also be modified in the XML configuration file.

There are also some parameters related to the event, the perturbation, in the configuration file. Firstly where and when it occurs within the simulation. In the experiments on the small network we selected both directions of an edge, as shown in figure 4 in orange. The decision was also taken to, by default, make the perturbation happen prior to the first Reinforcement Learning step so the agent has to react to it and can't anticipate it. The occurrence of events creates problems within SUMO as cars trips are precomputed and if one of them has in its route the blocked edge it will cause an error. To counter that, SUMO offers a rerouting module that can, during the simulation, readjust the route of any car taking into considerations blocked edges as well as traffic. The downside of this is that during the simulation some additional computations are required (to find the new routes) and the simulation is thus longer.

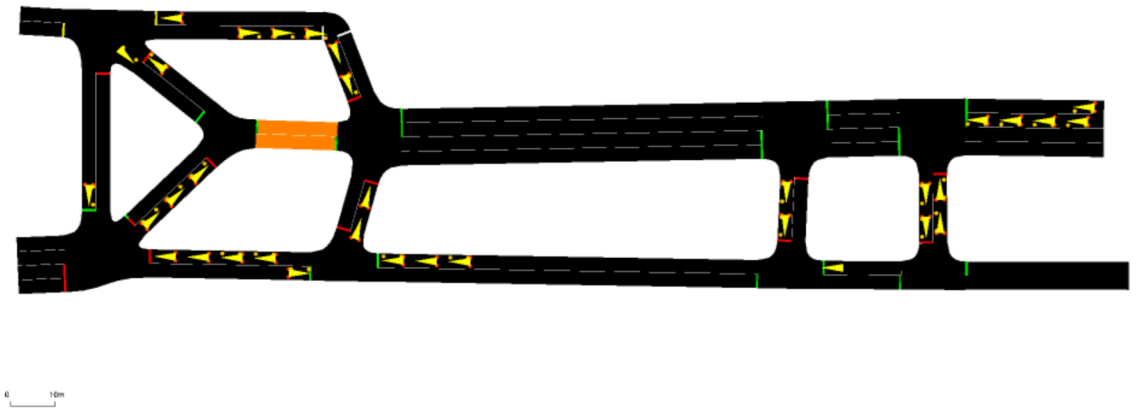


Figure 4: Screenshot taken during a simulation on the small network

The simulation is managed within a python class as it will be explained latter on, this class implements all the interaction between the simulator and the Reinforcement Learning agent. In order to make it as standardize as possible this class implements all the function from the openai gym environment which is a common practice within the Reinforcement Learning area. Thus our simulator class inherints from the open ai gym environment class.

3 Reinforcement Learning

As explained in 1.2.2, Reinforcement Learning is a field of unsupervised Machine Learning. It offer the advantage to be able to find optimal solutions in very complex problem with a fairly simple framework, through the design of a reward function. The reward function is a powerful tool that rewards some transition in the field. As discussed in 1.2.2, the approach chosen for this project is Deep Q-learning, that estimates the Q-function, a RL tool to score state and action pair, with a neural network architecture.

The RL paradigm intuitively fits the traffic problem using modelling tools. Indeed, traffic regulation has clearly defined objects, actions and states and, while there is no labels for smoothly running traffic, there are multiple indicators for reward functions. Implementing RL for this problem nevertheless still requires careful design choices. In particular, proper design of state representation, i.e. which information does the agent have, and reward function are key to ensure the agent has the information to learn the optimal policy, and that the optimal policy is optimal in the sense described by the main goal. Such design challenges and the choices made for this project are discussed at length in the following subsections.

State representation choice is a major challenge as several specifities of the project, described in section 1.2.1, increase the complexity behind it. First the fact that the simulation step does not match the Reinforcement Learning step means that our state representation needs to represent the simulation for given time frame, i.e. not only at a given point. A related problem is the representation of traffic lights. The goal of the project is to reduce and smooth out traffic. Therefore any group of cars not advancing should be considered bad by the reward function and the agent but traffic lights, entirely out of the agent's control, have this exact effect on cars. A state representation where the traffic light status are given to the agent in order for it to understand the situation may work but, since the simulation step is not equal to the RL step, the status of traffic lights will change within a RL step. Efficiently transferring traffic light status information to the RL algorithm at each step is thus highly non-trivial. Nevertheless, within the RL paradigm, agents are known to efficiently learn patterns that are quite complex by observing a large number of simulations. Therefore it might learn by itself the impact and functioning of traffic lights on traffic as it should follow a fairly consistent pattern.

3.1 Deep Q-Learning

As explained in 1.2.2, this project implements Deep Q-Learning for traffic regulation, where the agent has access to information on the entirety of the network and chooses global speed limits. While DQN may in theory be off-policy, the framework of this project allows on-policy learning. The policy chosen for generating episodes is a mix of random and optimal policy, to ensure proper exploration. The probability of choosing a random action is reduced in the last episodes to fine tune estimates in state transitions that are frequent under optimal policy.

In order for the agent to explore the state and action space and to learn rewards on it, the agent start with an exploration probability of 1.0. This variable represents the probability of the agent taking a random action instead of a predicted action, i.e. following the estimate of the optimal policy at the current step. This is done so that the agent explores the solution space. At the end of each episode, i.e. the end of an entire SUMO simulation, the exploration probability is updated, slowly decreasing, and the actual policy moves from random actions to predicted ones. The evolution follows the formula, for step $t+1$, $p_{t+1} = p_t \exp(-\delta)$, where δ is the decay parameter and p_t is the value at the step t. If the number of Reinforcement Learning steps in a simulation is higher than the batch size chosen (which in the experiments described in 5 is, after a single episode, constantly the case) the agent neural network

is fitted with a random batch of episodes from the agent's memory. Once the agent is trained, another episode starts and a new simulation is launched. This process is repeated until the number of episodes wanted is reached. All those values are again entirely configurable in the XML configuration file. This learning mechanism is, with the arrangement made to adapt it to this case study, standard practice and is explained in further details in the original paper introducing DQN [8].

3.2 State representation

State representation corresponds to the information from the simulation that is available to the agent at each step. It is the data upon which the agent chooses the action to take, i.e. which speed limit to set. It is also, along with the action taken, the input of the Q-function.

State representation should in theory include all information that is important and useful for the agent to make its decision. Very often perfect state representation is simply not possible: the route a road user is going to take is unlikely to be known with certainty by an algorithm, and should thus not be included in the representation. State representation should in spirit be limited to information that may be available to a real-life decision system, e.g. traffic lights information, position and speed of cars. While one may thing that all available information should be rerouted to the agent, there may be several pitfalls to such approach. First, there are some entries that are simply of no use to the decision process. Adding noise to the representation is likely to make learning the Q-function harder. Similarly preprocessing the data, while it may reduce the quality of the information available to the agent, may simplify the learning process. Additionally, increasing the state representation size may, when the solution is scaled up, cause computation issues.

For this case study, multiple attributes of the network and its elements were considered for building a smart state representation. Their definition, advantages and drawbacks, along with final design choices, are discussed in this section.

3.2.1 Image Observation

The first possible state representation explored was to consider pictures of the network, i.e. give the agent images of the simulation. The idea is to provide the agent one or several gray scale images (To tackle the time dimension) of the simulation at each step. When exploring how to obtain such images with SUMO the only possible option (confirmed by SUMO's suppport team) was to call in the previous step a function that makes a screenshot and saves it into an image file. Afterwards, in the next simulation step, the program is able to read the created file and obtain the image. Beyond being clearly not ideal in terms of computation time, this also caused several additional problems. First for the screenshot function to work, the GUI had to be enabled which again increases the running time. Secondly the dimensions of such screenshot depend on the size of the created GUI window, which depends on the computer itself. This means that there is no way to get in advance the dimensions of such screenshot. In order to adapt to that, the creation of the agent's neural network was delayed until the first batch of images is received as the neural network requires the size of the inputs to be fixed and known. Additional in the running of those experiments if the user changes the dimensions of the sumo-GUI window, the dimensions of the screenshot change and the program crashes. No possible fixes could be found for this problem as the new received screenshot cannot be simply resized to the original dimensions as parts of the network may have been lost in the process. Once all these problems were solved or circumvented, the program worked correctly on the small network. An example of the images obtained can be seen in figure 4. This figure shows the image prior to it being processed into the state representation. The processing consist of passing the image to gray scale to reduce its size, get rid of

the color dimension (that will be replaced with the time dimension) and down scaling it by half. The image after preprocessing still correctly showed the roads, cars, traffic lights status and the road where the event is happening. The agent also needed some modifications: the neural network structure is changed to a Convolutional Neural Network structure accepting 3D images. This new CNN is composed of 3 convolutional blocks (3D convolution followed by max pooling and batch normalization) followed by a global average 3D pooling and 2 dense layers with a dropout layer in between.

The following implementation worked on the small network, although the running time was important due to the technical aspects described earlier. However, when moving to the big network, representing the Eixample neighborhood, a major problem came out. The screenshot of this network was the same size as the one from the small network and while on the ones on the small network one could easily see all the network, in the big network it is not possible to see any vehicle or traffic light which makes this state representation much less usable. No fixes were found for this issue and this proposition of state representation was abandoned.

3.2.2 Edge information

To avoid the pitfalls encountered with the Image state representation, a careful process of creating a custom state representation where the data given to the agent is carefully selected was designed.

The first such representation, called "junctionsObservations", takes an approach centered on edges. Edges are unidirectional, meaning that if a street has both driving direction it will be represented by 2 different edges. For each edge, the following information is given to the agent: The junction from which the edge starts, the junction where the edge ends, the number of vehicles that drove on that edge since the last Reinforcement Learning edge, the average speed on the edge, the number of lanes the edge has and finally a "blocked" boolean representing if the edge is the target of the event or not. The idea is to provide to the agent a comprehensive picture of the network structure as well as an overall tractable understanding of the ongoing traffic. The data of all the edges is then flatten into a vector fed to a neural network composed of 4 dense layers.

One of the big advantages of the following state representation is its scalability. The agent memory can be increased as the data size is considerably smaller and the execution time is drastically reduced (The training is faster since the neural network is smaller, the GUI is not needed for the simulation and there is no need to read and write to disk memory). Nevertheless, it is uncertain that this is enough information upon which to efficiently learn a policy. It leaves open the question of whether this information is enough for the agent to have a good understanding of the situation to base its decisions.

3.2.3 Edge analytics

The lack of time related information is a major problem in the "junctionsObservations" approach. I thus developed another approach that put more focus into it. This second state representation, called "edgeAnalyticsObservations", computes the state representation with the following information: Per edge, the total number of cars that went through it, the minimum and the maximum number of cars in the edge at a simulation step, the standard deviation, the average speed, the number of lanes and finally the "blocked boolean".

I decided to remove the information related to the junctions as the agent should be able, from the given information, to understand the traffic flow between the edges. My hypothesis was that, from the given information, the agent could recover the relationship between edges.

3.2.4 Edge vehicles

Another approach, similar to the "edgeAnalyticsObservations" approach was also implemented, where instead of preprocessing the array of cars that went through an edge and giving the agent different statistical objects, I directly provided the agent with the entire array and let it find its correlation from it. The state representation was thus simply an array of number of cars per simulation step from the last Reinforcement Learning step, the average speed, the number of lanes and the "blocked" boolean. This state representation was named "vehiclesObservations". The small downside of this approach is that the state representation is bigger which may make the learning harder as the agent has to extract himself the relevant information from the vehicles array.

3.3 Reward function

The reward function is central to the Reinforcement Learning paradigm. Through the learning process, as it describes immediate rewards, it shapes the Q-function, which is in turn optimized to derive the optimal policy. The reward function defines, by rewarding differently different configurations, what the objective of the agent is. Notice that it need not reflect such objective perfectly but simply aims at being a good simple proxy. It simplifies complex objectives into scalar scores. Subtle differences in reward function design may cause huge shifts in the strategy of the solution. For example, for traffic regulation, considering global values may ignore specific individuals stuck in traffic to favor the fluidity of big roads, while taking min/max values may overly focus on such individual cases.

The design problem around the reward function is thus very complex. Not only does it have to properly reflect what the objectives of the task are, it also aims to be comprehensive and be sensible to small improvement of the situation, as it may substantially Q-learning algorithms' efficiency. Many metrics are available in this case study and their advantages and drawbacks, along with final design choices, are discussed in this section.

For this project, the goal is to reduce the traffic density and to ensure trips in the network are smooth and not disturbed by traffic. Therefore the first measure I considered is, intuitively, the waiting time of the cars. The reward function defined is thus the inverse (i.e. the negative) of that measure.

The first reward function first computes at each Reinforcement Learning step the average cumulative waiting time since the last RL step. As the agent tries to maximize the reward function and the goal is to reduce as much as possible the waiting time, the computed average cumulative waiting time is then negated. The main problem found with this reward function is that it didn't take into consideration how many cars were left in the simulation so that single car blocked in front of a red light, if alone in the network, give the same reward value as an entire jam.

To try to solve such issue the measure considered was modified to consider the total sum of the waiting time in the network. The main advantages of this idea is that it should push the agent to aim at having as few cars as possible within the simulation. Vehicles leaving the simulation represents an end of the trip, and should thus part of the objective of the agent.

Finally, a last measure I considered is is the average vehicle speed within the network. This represents a completely different approach. Again, it faces the same problems as the average cumulative waiting time. However here considering the sum doesn't solve the problem as it will make the algorithm find that two cars going at an average speed is just as good as if one car goes at a high speed and the other at a low speed, which is suboptimal. Indeed what the project aim at is a smoother traffic and not a disparity between the situation of the cars within the network. Here a solution that could be explored is, for example, a squared sum, to ensure it penalizes high differences.

4 Implementation and Execution

In order to be able to execute the code attached to this project it is necessary to have installed tensorflow and SUMO (This project was done with the version 1.12.0) both can be installed from their respective website. Once the libraries and program are installed, you simply need to enter either the src/imageObservation if you want to use the screenshot state representation or the src/customObservation for the others state representation. Then adapt the configuration file and run "python main.py".

The XML configuration contains all the variables that can be modified for an experiments, most of the attributes are quite self-explanatory. However some may need clarrification:

1. The environment.state_representation for the custom observations allows to select the state representation you wish to use, there exist three posibilities: edgeAnalytics, edgeVehicles and junctions
2. The rl.reward_function for the custom observations allows to select the reward function you wish to use, there exist three possibilities: avg_waiting_time, average_speed and sum_waiting_time
3. The sumo_configuration.route-file represent the route-file you wish to use, if you wish to generate a new route file, this field must be blank.
4. The event options allows you to change the simulation step when the event takes place and the targeted edge on which the event take place.

The attached file contains two folders: src and experiments.

The src folder is in turn separated into two folders, one named "imageObservation" that contains all the code in order related to the screenshot state representation. The second one "customObservation" that contains all the code for the others state representation. Even though some code is repeated the following decision was taken to make the code as simple as possible for the custom observations as the image observation forced a lot of changes in both the environment class and the agent class. This was done for execution time optimization. Every single class and function has been commented to explain their parameters, what they do and what they returns.

The code inside both of these folders follow the following structure:

1. main.py is the main script and calls at all the other parts.
2. mainLoop.py this file contains the main loop where the simulations are repeated and the Reinforcement Learning agent acts.
3. reward_function.py this file contains the implementation of the different rewards functions.
4. sumo_event.py this file contains the functions that trigger the events in the simulation.
5. utils.py this file contains a bunch of functions that didn't fitted in any other file.
6. generateTrips.py this file contains the script to generate the trips with SUMO randomTrips tools.
7. generateConfig.py this file contains the script that generate the sumo configuration.
8. environment.py this class controls the SUMO simulations.
9. environment_actions.py this file implements the actions needed for the SUMO simulation.

10. configuration.xml this file contains the configuration for the experiment.
11. configuration.py this file reads and provide access to the xml configuration file.
12. agent.py this class represents the Reinforcement Learning agent.
13. action_space.py this class defines the range of actions that the agent can do.
14. the sumo folder contains the files related to the small sumo network.
15. the sumo_eixample folder contains the files related to the big sumo network.
16. the state_representation folder contains the different implementation of the state representation.
17. the result folder is where the text file obtained from the experiments are saved.
18. the tmp folder for the imageObservation is where the screenshots will be saved.

The experiments folder contains the output of experiments done. The experiments are saved in separate folders and their names indicates what they represent with the following format: "exAnalyticsCumulative6225"

1. "ex" for experiment
2. "Analytics" for the state representation with the following options: Analytics, Junctions, Vehicles and Junctions.
3. "Cumulative" for the reward function with the following options: Cumulative, Sum and Speed
4. "6225" for the execution time in seconds

Additionally two other experiments have been done that does not follow this pattern is the "testChangedActions" that was done as is name indicates changing the actions the agent can take in order to prove the algorithm is able to learn. And the "testNoTrafficLights" which is an experiments where the traffic lights have been removed from the small network. While all the experiments have been done with a period equals to 5, one single one "exImage1911_period4" has been done with a period equals to 4 as its names indicates.

All the experiments folders contains the same, first the configuration file from the experiments that allow to check the exact configuration that I used for this experiments. Take into consideration that some of these XML file may differ from the final version of the configuration file as they were done some time ago. Additionally the experiments folder contains a result folder that contains for each episode within the experiments a text file containing for each step the Reinforcement Learning step, the speed chosen and the reward obtained.

In order to visualize these results the experiments folders contains two python scripts, one plotting the cumulative reward per episode for an experiment and the other plotting the speed chosen and/or the reward obtained for a given episode from an experiments. Take into account that you need to change the values at the top of the scripts in order to select the correct experiment and attributes.

5 Experiments

In this section, I present the experiments run in the frame of this project. The implementation details of the experiments are described in section 4. This section focuses on presenting and discussing the results of the experiments. Technical and theoretical limitations of the chosen approaches are also discussed.

5.1 Small/custom network

As explained in 1.2.1, a simple small network was first implemented. This network had the advantage to be a small, controlled environment. The network and experiment setup are easily modifiable using the Netedit tool and changing variables in the XML file respectively. Experiment run under reasonable time and with no important computational requirements. Many of the experiments run on this network have been tested under slightly different scenarios to explore behaviors of the RL model.

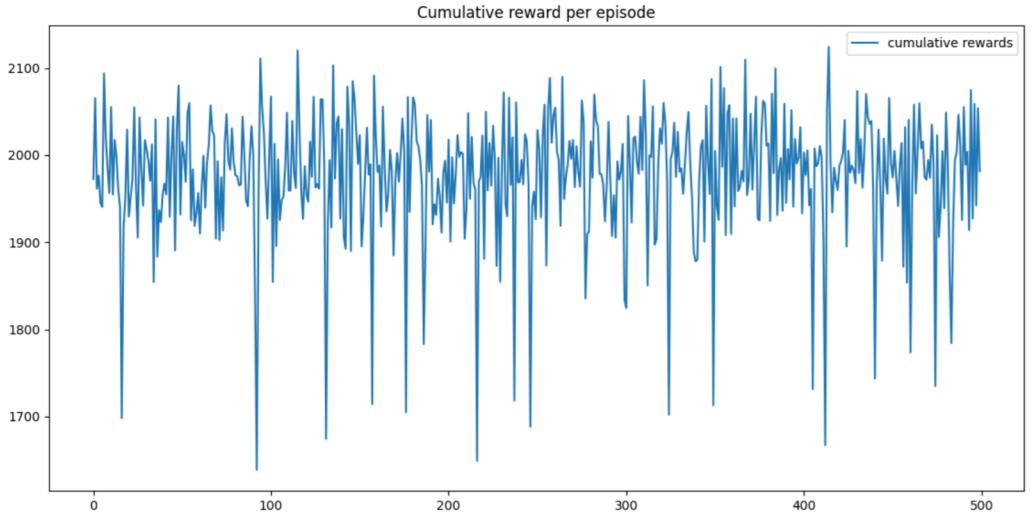


Figure 5: Cumulative reward per episode obtained with the Analytics state representation and the average cumulative waiting time reward function

While the agent does set a given policy, the chosen policy does not seem to yield better rewards than random actions, for all considered reward functions and state representations. Indeed the cumulative rewards, as shown in figure 5, on episodes are roughly the same at the beginning, where actions are chosen at random, and towards the end, where agent choices are almost exclusively defined by the chosen policy. This may describe either an incapacity by the agent to learn the proper optimal policy or that random is equivalent to optimal in such setting. An additional experiment, where possible actions (2,4,8,10 and 30 km/h) included very low and obviously unoptimal speed limits shows that the agent is able to learn a policy which yields significantly and consistently better results than random, as shown in 6. This result guides us towards the second hypothesis and confirms the doubts expressed on the relevance of such network for this task in section 1.2.1. The impact of speed limits for traffic regulation, beyond usual safety concerns, is likely of a global scale nature, acting on the behavior of the flow of cars in a densely connected network of roads. Such effects are likely not observable on the small network.

Cumulative reward per episode obtained with the Analytics state representation and the average cumulative waiting time reward function

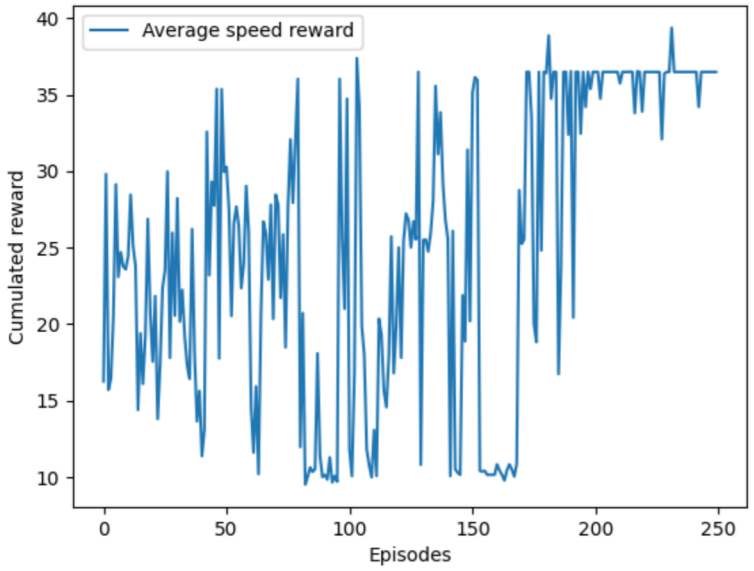


Figure 6: Cumulative reward per episode obtained with the junction state representation and the average speed reward coupled with ill-defined action space.

5.2 Big/Real network

The realization of the experiments on the big network representing the Eixample neighborhood required important preprocessing effort. For example SUMO doesn't implements tunnels and thus they need to be removed from the network. Similarly, dead ends are considered entry and exiting points to the network, for this reason, as there exist several dead ends within this neighborhood, these were removed as they do not contribute in any way to the traffic. When importing from osmWebWizard, the tool to import real world networks into SUMO, the majority of big junctions, junctions like Plaza España or Diagonal with Passeig de Gracia are extremely poorly implemented as seen in figure 9 and thus the traffic within them is far from reality and highly problematic if not corrected. Important work to improve the model while keeping it faithful to the reality is needed.

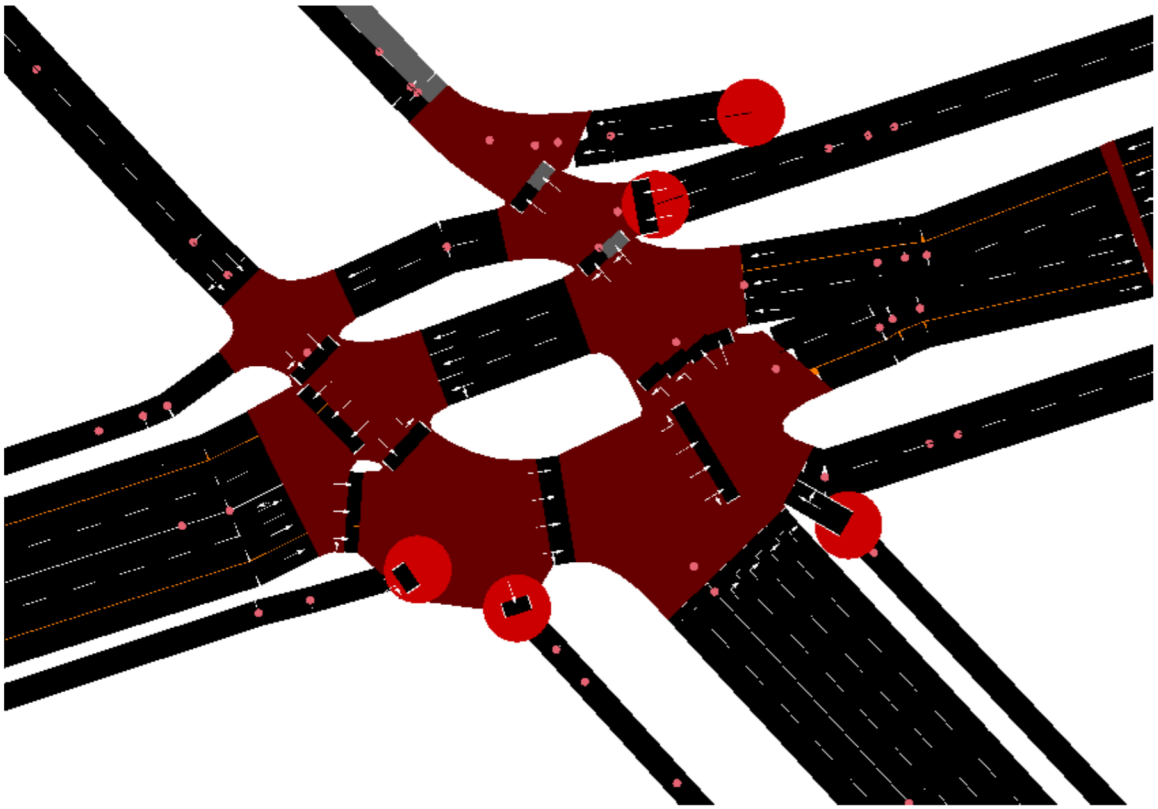
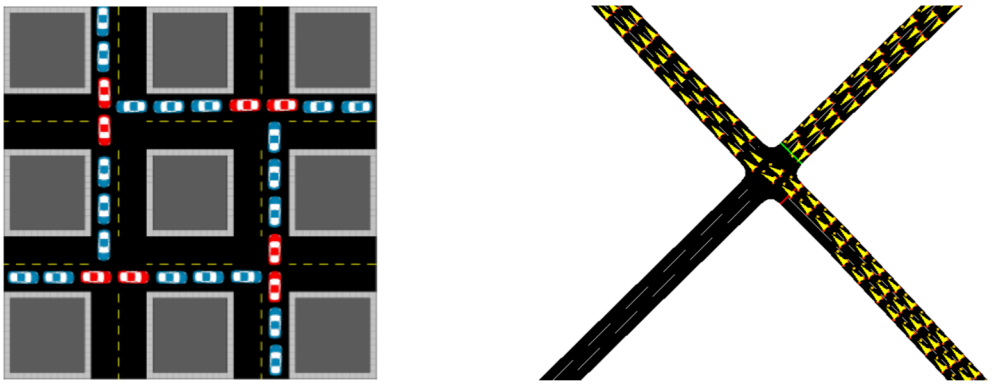


Figure 7: Example of poorly imported junction, here Diagonal with Passeig de Gracia

Another additional problem is the traffic lights gestion. SUMO traffic lights are generated randomly within a junction and doesn't take into consideration the other junctions around. This does not reflect current urban planning strategies. Recreating a realistic model of such crossings is very hard and time-consuming to put in place. For example, in plazas like Plaza Universitat and surroundings the fixing of the traffic light management is extremely complex.

Another major problem that couples with the traffic lights problem is what in gridlocks. A gridlock is a situation where traffic at a crossing is blocked, as figure 8 displays it. This kind of scenario completely and permanently blocks the traffic and completely prevents the vehicles from leaving the simulation. This problem appears really often in SUMO, as SUMO allows vehicles to enter a junction even if it blocks the other routes as also seen in figure 8. SUMO offers a solution to this problem is to allow cars blocked in such situation, after a given number of steps, the simulator ends up letting it pass through the others cars. Although it solves gridlocks, such solution is not realistic when trying to study traffic congestion. SUMO thus struggles to realistically model high traffic densities.

An additional problem that appears within the simulation seemingly randomly and with no clear reason is that, in some scenarios, cars end up not advancing and creating fake traffic as seen in figure 9, where cars on the right do not advance even though the traffic light is green. The event shown in this figure goes on until the end of the simulation.



Gridlock schema

An example of non-stopping simulation

Figure 8: Gridlock phenomenon and non-stopping simulations

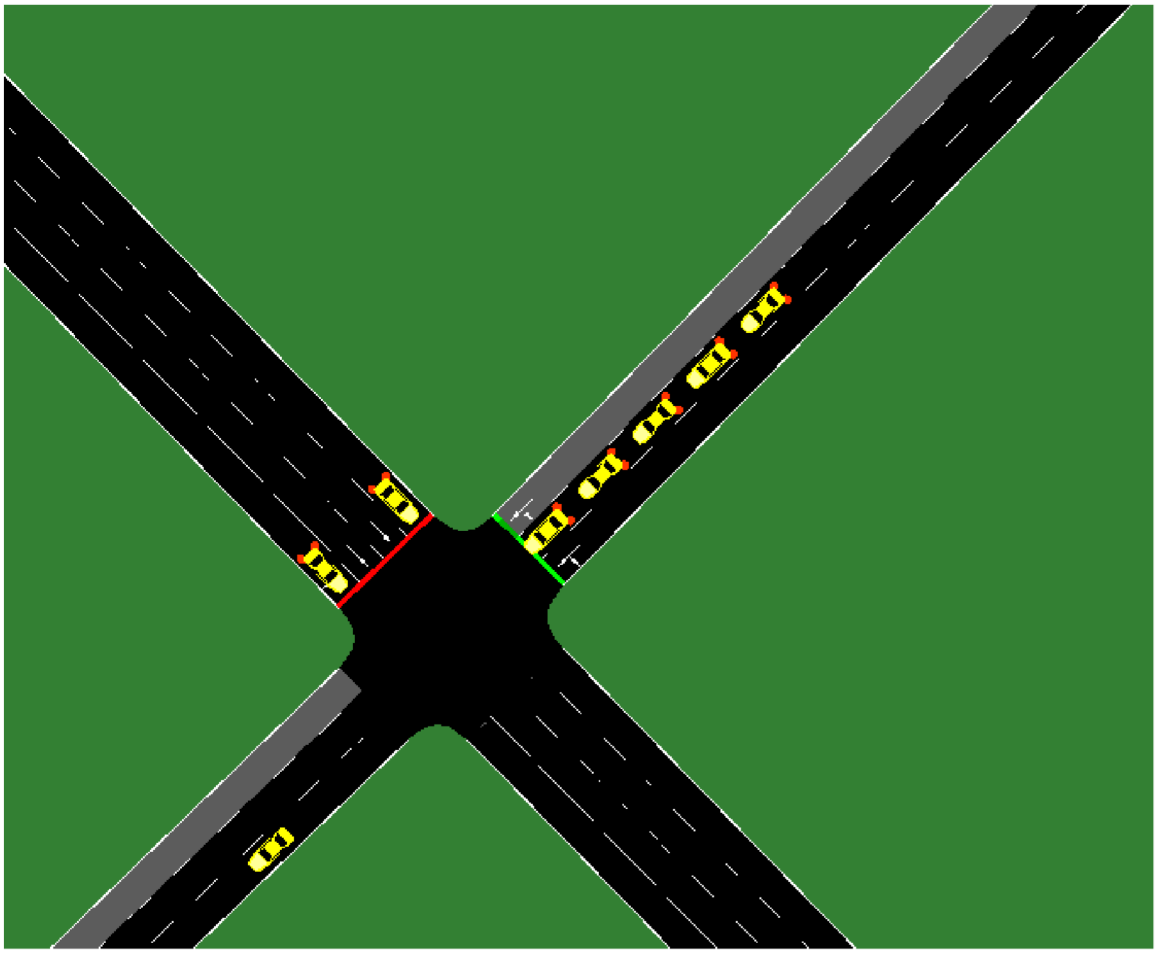


Figure 9: Fake traffic created in the simulation (The cars end up being blocked like that until the end of the simulation)

Due to these problems, SUMO struggles to model high density traffic, i.e. intermediary situation between light traffic and a complete traffic congestion where the vast majority of cars are in a never ending jam. This greatly compromises the value of conclusions of experiments in this network.

An additional factor to take into consideration is the execution time. In this scenario, the execution time can be considered small when there is little to no traffic, but it grows exponentially with the traffic situation, reaching situations where simulations for 2 hours of real-time traffic lasts more than 4 hours of computations. Therefore training a Reinforcement Learning agent on such network may be extremely long.

After spending a considerable amount of time to tackle such problems without finding real solutions, and due to the difficulties of SUMO to model high traffic situations, the realization of useful experiments within this network in the framework of this project seemed highly impossible and was thus in the end abandoned. A better simulator seems necessary in order to fulfill the objectives of this project on such network.

6 Conclusion

This project builds a framework for the implementation of a Deep Q-Learning Reinforcement Learning model on the SUMO simulation tool for urban road networks on the model of the Eixample neighborhood in Barcelona. This report includes the description and explanation of the methods implemented, along with the problems encountered and tackled. The speed limit tool in urban areas is, in the simulation, controlled by the agent to fluidify road transit and prevent traffic perturbations and jams. Multiple approaches are tested with the design and experimentation of multiple reward functions and state representation, from scalable pre-processed statistical measures to unaltered images of the network. Most such configuration struggle to yield interesting and efficient results on speed limit strategy implementation on a small network. Speed limit regulation as a tool for regulating traffic is likely to only have a significant impact on big-scale networks. For bigger scale road networks, the computational cost of the SUMO simulation makes it virtually impossible to run such Reinforcement Learning model. SUMO is not adapted to such task and another simulation tool may be needed.

References

[1] C. for Economics and B. Research, "The future economic and environmental costs of gridlock in 2030," *Report for INRIX*, 2014.

[2] L. N. Alegre, "SUMO-RL." `https://github.com/LucasAlegre/sumo-rl`, 2019.

[3] C. Wu, "Flow." `https://arxiv.org/abs/1710.05465`, 2017.

[4] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML 2005* (J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo, eds.), (Berlin, Heidelberg), pp. 317–328, Springer Berlin Heidelberg, 2005.

[5] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," 2015.

[6] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Mar. 2016.

[7] M. Marfeychuk, "Learning control policies in smart cities from physical data." `https://fenix.tecnico.ulisboa.pt/cursos/meic-t/dissertacao/846778572212554`, 2020.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.