Using Reinforcement Learning for Traffic Signal Control with Multiple Intersections

AASMA 2018/19 - Group 9

Mário Cardoso
Instituto Superior Técnico
83523

João Cardoso
Instituto Superior Técnico
81361

Francisco Miranda
Instituto Superior Técnico
83460

ABSTRACT

Traffic light control is one of the main means to control traffic flow. Developing effective systems to control these mechanisms can greatly improve traffic throughput and reduce congestion. In this report we describe this problem as a Markov Decision Process (MDP) and apply multi-agent reinforcement learning techniques to optimize traffic flow. (In this simulated environment, multiple agents are responsible for the traffic flow optimization around a single intersection, each agent controlling all the traffic lights within it

1. INTRODUCTION

Traffic control is, at the present time, a critical problem with serious consequences at the environmental, social and economic levels. It is estimated that, in 2018, people living in the city of Lisbon have spent, on average, 162 hours in congestion [1] which translates into several hours of productivity and its associated monetary value lost. Additionally, it poses a threat to our environment and people's respiratory system as the cars' constant stopping and starting leads to a higher amount of emitted dangerous gases to the atmosphere when compared with constant motion [2]. As a result, increasing the efficiency of traffic management would have a meaningful positive impact on our society.

Since traffic tends to be mostly disturbed in intersections managed by lights, these will be the focus of our study. As it stands, traffic lights in the real world will, in most cases, use predetermined times between light changes [3] which can lead to a lessened overall traffic flow in intersections, as traffic intensity changes during the day cycle. In this paper we intend to develop a multi-agent system capable of calculating the traffic intensity and, using that information, coordinate multiple lights in order to obtain smaller car queues, thus increasing the global traffic flow. To solve this, we model the problem as a Markov Decision Process (MDP) and utilize reinforcement learning (RL) algorithms to provision the agents with the ability to learn how to generate and keep updating an internal representation of the environment and, based on that, choose the best course of action.

2. PROBLEM SPECIFICATION

2.1 Environment

The environment used for this paper consisted on two connected intersections. These had two-way roads and cars spawned at the edge of each road. The spawn rate of the cars can be adjusted according to several distributions. These include a normal distribution, increased spawn rates in one of the directions or the same spawn speed for both directions.

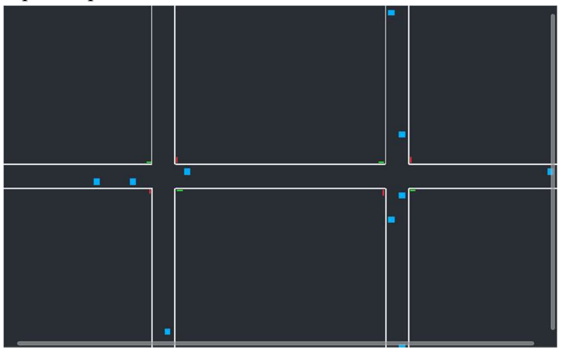


Figure 1-Environment

On each intersection there are 4 traffic lights that control the incoming vehicles by switching between red and green. We decided to omit the possibility of the lights to turn yellow as we found it would effectively act as a red or a green in this simplified model. Nevertheless, a possible future approach would make use of the yellow and try to understand how different levels of aggressiveness towards yellow by the drivers would affect the learning process and outcome.
The displayed cars simply drive forward from the end of road that they spawn on until they reach the other end, stopping only when they reach a red light or come close to a stopped or slower vehicle in front of them.
We decided to remove collisions since our simplified world model meant that cars would always stop at a red light and as a result, collisions would not occur. If, in a future approach, a yellow phase is implemented, then special care must be taken to ensure that collisions are present in the simulation. This is discussed in a future section.

2.2 Model

Traffic flow is inherently a fluctuating quantity in the real world. During rush hour, as people are either going from home to work or vice-versa, or when big events happen, roads tend to become flooded with vehicles. On the other hand, traffic flow tends to slow down between mid-day and late rush hour and is at its lowest during the night [4].

With these requirements in mind, we wanted to test different solutions based on different architectures and models and compare them with one another. The solutions developed were:

- A model that behaves like real-world semaphores, changing traffic light colors every time interval. We used this model as the baseline to compare future results.
- A purely reactive agent where the agent perceives the number of cars in each direction and, if the perception indicates that there are cars waiting, the agent acts in a way that lets them pass. Since each agent controls several traffic lights in different directions, the perceptions and actions were made in turns, with the agent choosing an action for a direction in a time interval x and for the other direction in time interval x+1.
- A Reinforcement Learning agent utilizing Q-Learning. We will discuss this model in a further section.

2.2.1 Agent specification

As previously mentioned, each agent controls the set of traffic lights within each intersection. In all the models described above, the same set of perceptions and actions were used. The agents had access to the following perceptions:

- Number of cars waiting for a green light at each traffic light.
- Total number of cars situated within a street connected to a traffic light.

These perceptions allowed the agent to react or deliberate in order to perform an action. The set of actions allowed for each agent were:

- Change horizontal traffic lights to green and vertical traffic lights to red. We will refer to this action as a1.
- Change vertical traffic lights to green and horizontal traffic lights to red. We will refer to this action as a2.

Controlling all lights in an intersection simultaneously was chosen as opposed to allowing independent control of each traffic light color for two main reasons. Firstly, individual traffic light control could lead to deadlocks as cars become stopped in the middle of the intersection. Secondly, this reduces the size Q-matrix used in reinforcement learning which, in turn, reduces the amount of exploration required to better approximate the optimal Q-values.

3. **REINFORCEMENT LEARNING AGENT**

3.1 Definition

Our agents must be able to constantly understand how the world around them changes, how they can interact with it and what effects their interactions had on the environment so they can later make better decisions when recurrent environment states appear. This set of requirements makes RL a suitable approach to solve the problem presented in this article.

Agents that work using RL algorithms interact with the environment by observing a state *x* and taking action *a*, according to some policy *π* while observing a delayed cost *c(x, a)*. The goal of RL is to find the optimal policy *π** that minimizes the discounted cumulative cost received, using a γ discount factor that can decide the importance of immediate vs future costs.

Underlying the model used in RL is a MDP. This model is a sequential decision process defined by the set (*X*, *A*, *P*, *c*). *X* represents the state space. States contain relevant information from the environment to enable decision making. The state space contains the set of possible states. *A* represents the action space which contains the set of possible action. An action is the means by which an agent can influence the environment. *P* represents the transition probabilities matrix which maps state-action pairs to a probability distribution over the states the agent will see after executing that action from that state. *c* represents the cost function, which is a function that takes as input a state and an action and outputs a value that evaluates that state-action pair. Given the nature of the problem, we decided that the agent is always successful in executing a state transition. In other words, we considered that the probability of reaching a state in time step *t+1* from a state and action at time step *t* is 1. In MDPs, environments verify a property known as the Markov property which says that the state at instant *t+1* depends only on the state and action at instant t.

$$\mathbb{P}\left[\mathrm{x}_{t+1}=y \mid \mathrm{x}_{0:t}=\boldsymbol{x}_{0:t}, \mathbf{a}_{0:t}=\boldsymbol{a}_{0:t}\right]=\mathbb{P}\left[\mathrm{x}_{t+1}=y \mid \mathrm{x}_t=x_t, \mathrm{a}_t=a_t\right]$$

Figure 2 - Markov Property

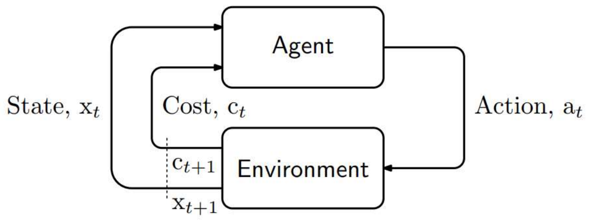


Figure 3 - MDP Representation

The main algorithm used was Q-learning, an algorithm used to find the optimal policy by using a Q function that we intend to minimize. *Q(x,a)* is the expected future cost of selecting action *a* in state x and this function can be estimated by successively updating *Q(x,a)* using the Bellman Equation, shown below.

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma \min_{a' \in \mathcal{A}} Q_t(x_{t+1}, a') - Q_t(x_t, a_t)\right]$$

Figure 4 - Q-Learning Bellman Equation

Where alpha is the learning rate, which defines how drastic updates to the q-table are. The higher the learning rate, the more drastic updates to the q-table become. *c* is the cost associated with taking action *a* in state *x*. Gamma is the discount rate, which defines how important future rewards are. The larger the gamma value, the more the agent only looks at immediate rewards. MinQ*(x',a')* is the minimum cost received by taking action a' in state *x*' (represented in the figure as *xt+1)*

For comparison, a model built with the SARSA algorithm was also developed. SARSA behaves similarly to Q-Learning but utilizes the following Bellman Equation:

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t)\right]$$

Figure 5 - SARSA Bellman Equation

SARSA and Q-Learning differ on the way the Q-value for the next state is calculated. Being an on-policy method, SARSA learns the values for the policy the agent is following. Q-learning on the other hand is an off-policy method since it learns the values of a policy while the agent is following a different policy. This is visible in the update rule, in the section corresponding to the Q-value of the next state. In Q-Learning, the value is computed from the greedy action at time-step t+1, independently of the action selected by the agent at that time-step. In SARSA, that value is computed from the actual policy used to sample actions during learning.

Selecting the next action in this update is done by following the Ɛ-greedy strategy which states that with probability Ɛ, the agent selects a random action and with probability 1- Ɛ it selects the greedy action. Initially, the agent starts with a high Ɛ to allow it to explore actions and fill the q-table, but, as time progresses, the Ɛ will slowly decrease to force the agent to start taking the optimal actions. This is known as exploration and exploitation respectively and finding a suitable trade-off between both is crucial to guarantee that the agent has enough information to make informed decisions, without spending too much time gathering information rather than taking the optimal actions.

Agents using RL will thus be able to learn a dynamic environment where it can be hard to connect actions and their effects making them perfectly suitable for the problem at hand in this paper.

Given that RL is being used it is now necessary to select the state and action spaces and to define a cost function to be minimized.

3.2 State Space

Finding an appropriate state space for this problem proved a complex task for two reasons. Firstly, standard RL algorithms require the state space to be discrete whereas several traffic indicators such as queue lengths and traffic flow or time in phase that could be used for states are continuous variables. It was thus necessary to discretize the possible state values in such a way that a high-dimensional state space would not be created. We ended up normalizing values in a range between 0-5.

Secondly, representing an intersection's state can be done in several ways. As a result, we ended up testing multiple state spaces such as measuring the time interval that the semaphores had been red since the last switch, the total number of cars stopped in the intersection and the total number of cars that were in the intersection whether they were moving or stopped.

In all the mentioned state spaces, each state was represented as a tuple (h,v) where the first value h represented the state in study of the horizontal lanes and the second value v the state in study of the vertical lanes. Due to the discretization range chosen, each state space was comprised of 36 states, which corresponds to all the possible combinations of values for h and v, with each value ranging from 0-5. (To obtain the location of a desired state (h,v) in the state space matrix we utilized the following formula: 6*h + v).

In the first state space mentioned above, each value h and v corresponded, respectively, to the time interval measured that the light was red in the horizontal lanes or vertical lanes.

In the second state space mentioned, each value h and v corresponded to the sum of the number of stopped cars in each of the lanes for each of the corresponding directions.

Finally, in the last state space, each value, h and v, is the sum of the number of cars situated in the corresponding lane, for each direction, whether they were stopped or not.

After testing each of these state spaces we realized that using the sum of the cars in the intersection proved to be the best representation of the world and, as such, is the state space being used in the final version of the model.

3.2 Action Space

The actions space defines the means by which the agent can influence the environment. The agent had access to 2 actions, as described in section 2.2.1.

3.3 Cost function

This function expresses the goal of the decision maker. Given a state and an action, this function instantly outputs a value that evaluates the decision. We utilized negative utilities (costs) to express loss in time.

Due to the complex nature of the problem and number of different variables interacting in the environment, several approaches could be taken when building this function.

With the goal of minimizing the cumulative wait time, a cost function was built that outputted the cumulative amount of wait time for the cars present in the direction opposite to the one that just turned green. For example, given a state (h,v) and the action a1, the cost function calculates and outputs the cumulative wait time for the vertical lanes. To do this, as soon as each car is spawned a timer is created. Whenever each car halts its movement, the timer is set off and increments its value based on the time the car has stopped. This timer is reset whenever the corresponding car exits a lane with a traffic light. The cumulative wait time is the sum of these timer values for each car.

This approach provided good results but revealed an issue. Since the cost function was implemented individually for each agent, each agents' decisions were done in isolation based on local perceptions and did not take into consideration the state of the environment surrounding the other connected agents. To solve this issue each agent was given information about the state of the environment surrounding other connected agents through a shared cost function. The cost function used was the previously mentioned one, but the cumulative wait time now included the wait times from cars situated in connected lanes of other agents. With this new cost function we hoped to achieve more effective action coordination between agents.

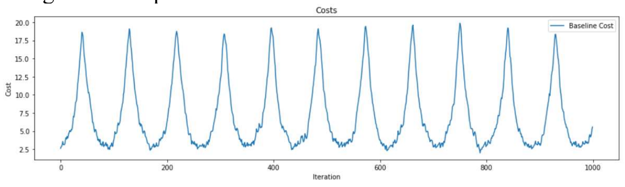
4. RESULTS

In this section we proceed to explain the means by which we trained and evaluated the performance of the agents that we have developed. All the tests assume that the car generators are using the normal distribution to decide the spawn rate.

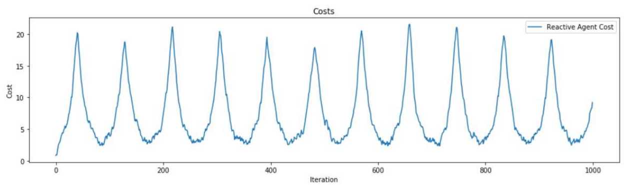
4.1 Training Process

All our models were trained with a fixed length time-step (0.5 seconds). This value is significantly faster than what would be used in a normal scenario. This was done to speed up training several times. With this same reason in mind the values provided by the Time library (counts difference between frames in real-time milliseconds) were multiplied by 5 to account for the speed up. At each time-step, the selected model behaves appropriately. The simulated real-world traffic light switches its colors. The reactive agent perceives whether there are stopped cars that need to pass. The reinforcement learning agent selects a current state, selects an action according to the Ɛ-greedy strategy, receives a cost and performs one iteration of the bellman equation. For the reinforcement learning agent, the parameters used were: $\alpha = 0.3$, $\gamma = 0.99$, lower bound of $\varepsilon = 0.1$, initial $\varepsilon = 1$, decrement of ε at each time-step = 0.001.

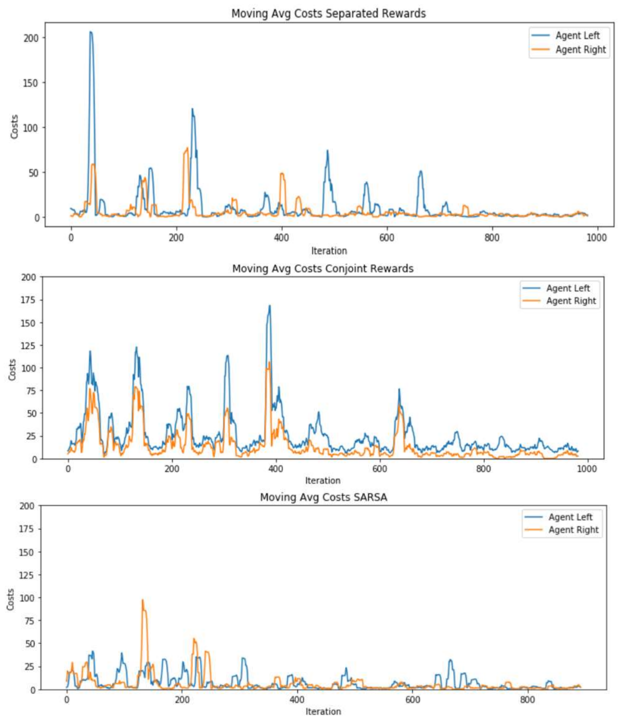
To plot the following graphics, we computed the moving average using a sub-sample of size 10 in a total of 1000 iterations



This graphic depicts the cost associated with the agent that switches the traffic lights' phases on predetermined times, as in the real world. This will set a baseline that will be compared to the other agents' results. Since the phase change is only dependent on time, the cost changes are expected to follow traffic flow changes according to the normal distribution. This can be seen on the graphic.



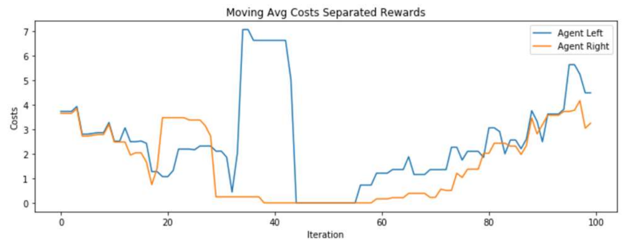
We can see that both the purely reactive agent and the real-world traffic light behavior model fluctuate costs according to what seems like a normal distribution. However, the costs in in the reactive agent end up with a slightly higher variation because unlike the real-world model, where only time influenced the light change, the reactive agent takes into account the number of stopped cars in the lanes. Since the reactive agent relies on stopped cars to decide its actions, whenever the light changes from red to green in a direction that lane no longer has stopped cars so in the next decision time-step the agent will execute the opposite action. This results in a model that tends to "flicker" traffic lights very similarly to the real-world model above.

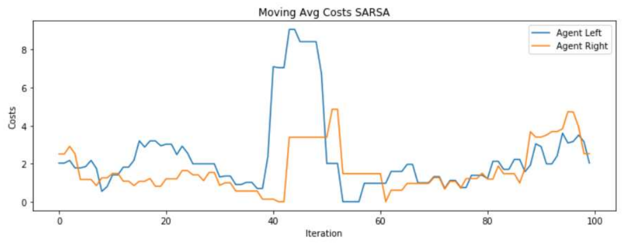


These graphics show the costs associated with the reinforcement learning agent with a cost function for each intersection agent using Q-Learning, with a conjoint cost function between them using Q-Learning and with a cost function for each intersection using the SARSA algorithm respectively. In the first two graphics the costs tend to vary greatly, as the agents are in an exploration phase, but as they start to take advantage of the optimal actions the cost ends up converging to a low value. Further analyzing these graphics, it is also clear that the separated rewards cost function performs better than both intersections having a shared cost. This could be a result of both intersections trying to synchronize their phase switches, but cars end up taking too long to reach the other intersection and when they eventually do reach it, that light has already changed to red because a new decision time step has been started. A possible way to solve this issue would be to make use of current traffic flow with relation to the maximum possible traffic flow in the reward function.

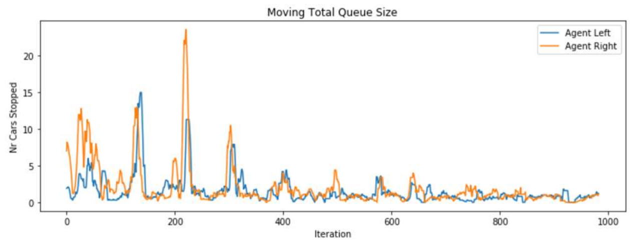
As a result, we decided to further explore the results from using the separated cost function.

Following this comparison, we analyzed how Q-Learning behaved when comparing to SARSA using the same cost function. By analyzing the first and third graphic we can verify that SARSA appears to converge faster and leads to more stable updates (less fluctuation between iterations).

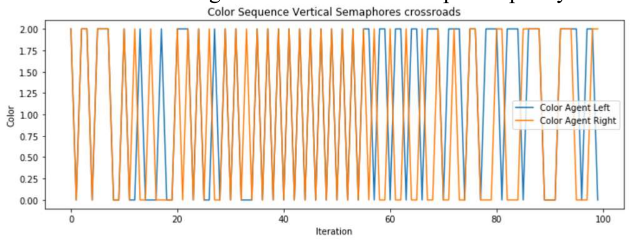




This graphic shows the cost for the last 100 iterations of the separated cost policy using Q-Learning. The second graph shows the same but for the model using SARSA. After verifying faster convergence rates and more stable updates we wanted to verify if SARSA outperformed Q-Learning after convergence was achieved and the agent started following the greedy policy more often. We noticed the two models displayed similar results in these conditions. In both models, it shows how our agents were able to achieve a better performance when compared to the reactive and real-world agents as they incur in smaller costs after they have learned the optimal policy. We proceeded to analyze other factors regarding the first model utilizing Q-Learning.



This graph depicts the total size of the queues across all directions, horizontal and vertical. Much like the previous graphs showing the cost change from the first iteration until the last, here the queue sizes start relatively high and then progressively become lower as the agent is able to learn the optimal policy.



This graphic presents the phase switches of the vertical semaphores in both intersections for the last 100 iterations. A value of 0 corresponds to the vertical lights being green at that iteration and a value of 2 means that those lights were red. We can observe that from around iteration 30 until 55 there was a higher traffic flow because the lights were consistently changing every iteration while from iteration 60 onwards traffic flow suffered a considerable reduction and lights could spend more time in each phase.

5. FUTURE WORK

In this section we present several improvements that could be made to further improve the learning and the world representation such that it is closer to the real world. These were not implemented due to time constraints.

Implementing a yellow phase on the traffic lights

One possible improvement to be made is to add the yellow phase to the traffic lights. As was stated this was not done because with our simple world representation they would merely act as a red or green light. However, adding a somewhat realistic yellow phase by having the cars ignore the yellow and go through the intersection if they have a speed higher than a set value would bring two significant changes to our model. Firstly, the level of aggressiveness of cars could be varied to better represent how antagonistically drivers follow rules in different areas of the world. Secondly, the addition of the yellow phase would require the simulation of collisions. At the present time these are not implemented because cars always stop on a red light and, as a result, collisions do not occur. Should a yellow phase be implemented, then collisions would begin to take place as cars irresponsibly ignore yellows. Our agents would then be required to take collisions into account when learning how to switch light phases and this would be represented be adding a higher cost to the action when a collision took place. These two changes would invariably alter the rate of convergence of learning and should be further explored.

Use Deep Q learning

As stated, standard Q-learning algorithms require the state space to be discrete and benefit from having a small Q-table, which means that our environment representation needed to be considerably simplified to suit the algorithm's capabilities. Deep Q-learning combines reinforcement learning with deep neural networks. Following this approach would allow us to take advantage of using a high-dimensional state space to better represent the world while reducing the effect of the "curse of dimensionality" because Deep Q-learning would be approximating the action and reward correspondence using a non-linear function.

Approximate real-world scenarios more accurately by utilizing uncertainty in perceptions with POMDPs

Instead of utilizing a fully observable MDP, we could model this problem utilizing a Partially Observable Markov Decision Process (POMDP). POMDPs are Markov Decision Processes where the environment is only partially observable. For this reason, POMDPs are defined by the set (X, A, Z, P, O, c), with S, A, P and c being the same components defined in MDPs. Z denotes the observation space and O denotes the observation probabilities. After a state transition, one of the observations in Z is perceived by the agent. As thus observations depend only on the current state and previous action. The observation probabilities map state-action pairs to a probability distribution over the observations the agent will see after executing that action from that state. This model is ideal for situations where the perceptions of the agent do not provide enough information to map to a real state in the state space. Instead the agent relies on its observations to calculate beliefs about the environment after which the agent can act upon those beliefs. A belief at a time step summarizes the history (all the agent saw) up to that time step.
This model could be applied to the traffic light control system in study if the agent could only obtain partial observability of the traffic state. This could happen due to sensor failure on a section of the corresponding lane, budget decisions or other reasons. Utilizing this model would provide a more robust and suitable for real-world scenarios solution.

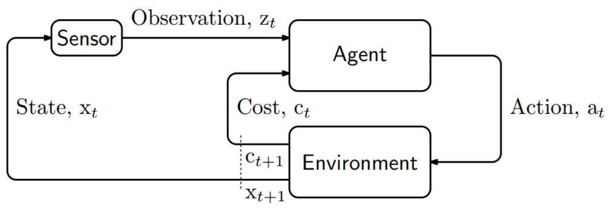


Figure 6 - POMDP Representation

6. REFERENCES

[1] http://inrix.com/scorecard/
[2]https://www.nrcan.gc.ca/sites/www.nrcan.gc.ca/files/oee/pdf/transportation/tools/fuelratings/2018%20Fuel%20Consumption%20Guide.pdf
[3] AASHTO, 2000 – Highway Capacity Manual. American Association of State Highway and Transportation Officials, Washington D.C.
[4]https://discover.data.vic.gov.au/dataset/typical-hourly-traffic-volume/resource/fe2d4fed-912b-4059-8289-7e8e62fbb47e