# Minimizing Car Delay at Traffic Lights Using a Reinforcement Learning Policy

**Timmy A. Hussain***
Department of Aerospace and Astronautical Engineering
Stanford University
timmyh@stanford.edu

## Abstract

This project investigates using a Reinforcement Learning based approach as a substitute for a periodic traffic light system functioning at an intersection. The objective is to minimize the average car delay at the intersection by developing a robust learning policy.

## 1 Introduction

Traffic light systems which function on a purely periodic basis or have a schedule determined a priori are unable to adapt to the conditions that evolve in the environment they operate in. This project will develop a traffic light system which minimizes the average delay per vehicle by developing a policy capable of adapting to the conditions of the environment. The environment will be fully observable and information such as number of cars and their relevant delays will serve as inputs to the system and the output of the system will be a policy which minimizes the delay: an action in an action space of traffic light signals.

## 2 Related work

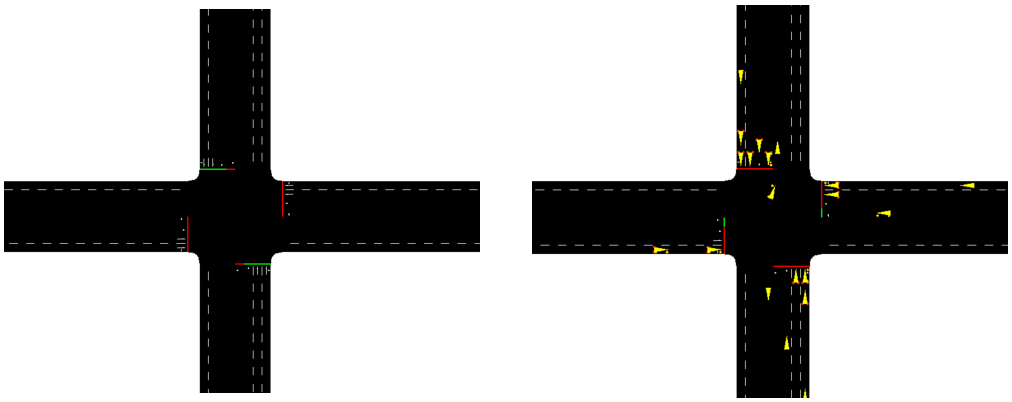
Recent work exists which approaches the issue as a reinforcement learning problem. The problem is well suited to be approached in this way as there is a finite action space and the task of minimizing delay or some similar metric lends itself well to the reinforcement learning characteristic of a reward function to be optimized. Some RL problems are based on a positive reward scheme which rewards good behavior (positive actions) while others are based on a negative reward scheme which penalizes bad behavior (negative actions). Work in the traffic agent RL space typically takes the latter approach by incorporating the metric into a negative reward function such as negative reward proportional to the amount of delay a car experiences due to the action taken by the agent. I will be drawing on the work done by Andrea Vidali[1] for much of the foundation for this project.

## 3 Dataset and Features

As I plan to use reinforcement learning for this project, having an environment which facilitates the learning process will be essential to training the model to develop the right policies. I will be using the Simulation of Urban MObility (SUMO) developed and maintained by the Institute of Transportation Systems at the German Aerospace Center[2].

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)



(a) Road intersection environment

(b) Road populated with vehicles

Figure 1: Intersection model

I am taking advantage of a codebase provided by [**Vidali**] which includes code to set up and interact with the simulation environment for the purposes of a reinforcement learning project. The environment pictured in 1 consists of 4 roads which meet at an intersection.

Vehicles travelling on each of these roads follow predetermined routes that include maintaining initial heading, turning left and turning right; the distribution of cars following each of these variations is governed by a Weibull distribution. The optimal policy simply becomes what sequence of green light signals to give to optimize the chosen metric (the duration of the opposing red light signal is necessarily determined by the duration of the green light signals and the duration of the yellow light signals is fixed and determined a priori by legislation). However, as can be seen in 1, traffic light signals are paired which reduces the action space from an initial 8 traffic light signals to 4. The environment is configurable with parameters such as the number of vehicles spawned, the total number of simulation steps (with each step representing a second of real time), the duration of the traffic light signals and more.

A primary challenge with RL programs comes in the representation of states and the state space. There exists a trade-off between high-fidelity state space representation and feasibility of current RL algorithms such as Q-learning to traverse the corresponding action space and find an optimal policy in reasonable time. However, by designing lower fidelity state space models through techniques such as discretization, it becomes increasingly likely to arrive at sub-optimal policies due to the information gap between the state representation and reality that is introduced with a lower fidelity model.

I am currently preserving the state representation used by VidaliE which essentially discretizes the roads into grids with blocks of variable sizes with increased granularity closer to the traffic lights and decreased further away. The state is then represented by the number of cars in each block. This allows us to know with increased precision the number of cars closest to the traffic light which is extremely useful for developing a policy capable of dealing with situations where there are fewer cars on the road (a motivating use case for a smart traffic light system).

## 4 Methods

The proposed algorithm here will be the Q-Learning reinforcement learning algorithm. An agent exists within an environment space and learns an optimal policy over time by choosing actions within an action space. Actions have rewards associated with them which are contingent on the environment state and action. Negative or 0 rewards might penalize and action and positive rewards might encourage an action. The agents learns an optimal policy by attempting to choose sequences of actions which ultimately maximize the reward over the duration of the training period. The Q-learning equation is as follows:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q^*(s_{t+1}, a_{t+1})$$

where $r(s_t, a_t)$ is the reward function associated with making action $a$ in state $s$ at time $t$.

As mentioned above, the reward function used in literature is tied heavily to the metrics of interest and takes the following form:

$$r(s_t, a_t) = \beta_1 w_t - \beta_2 w_{t-1}$$

where $\beta_i$ are hyperparameters and $w_t$ is the current wait time accrued at time t. Although there is no explicit dependence on state or action, the total wait time accrued will increase by varying amounts depending on what action is chosen. This implicit relationship is what allows the agent to learn an optimal policy.

The Q-Learning approach is governed by Bellman's Optimality principle which states that the optimal solution to a problem is reached by choosing the optimal solutions to sub-problems which compose the original problem. For the Q-Learning equation above, $Q^*$ represents the optimal reward obtainable given some state $s_t$. It can be more formally defined as

$$Q^*(s_t, a_t) = \max_{a \in \mathcal{A}} \{Q(s_t, a_t) | s \in \mathcal{S}\}$$

where $\mathcal{A}$ and $\mathcal{S}$ represent the Action and State space respectively. If the function $Q(s, a) \in \mathcal{C}$ where $\mathcal{C}$ represents the set of convex functions, then the max over the total set of Q(s, a) is also convex in s and a and therefore an optimum exists.

In lower-dimensional problems, it is tractable to reasonably store and access Q-values for each state and action pair in reasonable time, however, for most problems it is not. This is where the use of a Neural Network as an approximator to the Q-value function is useful. The neural network takes the form

$$f : \mathbf{R^n} \rightarrow \mathbf{R^m}$$

where $n$ is the size of the state space and $m$ is the size of the action space. It takes as inputs $x$ which represent the state and outputs the Q-values associated with each possible action in that state. The neural network is trained to output Q-values close to the target Q-values for each action given the input state. Therefore a mean-squared loss is appropriate in this situation. However, since the target Q-values are not known a priori, the target ("real") Q-values have to be discovered over the course of the learning process. This introduces some issues in convergence due to constantly changing input and output distributions that can be mitigated in a few different ways. One of these approaches is a Double Q-Learning approach which trains two models at the same time. This is the approach I will be taking and differs from the work done by Vidali.

## 5 Current Results

Currently, I have spent most of my time getting the environment set up on my computer. I am now able to run the simulation with the desired parameters and interact with it as necessary including pulling the state information and feeding in the chosen action when needed. As made clear in the introduction, a motivating case for this project is doing better than simple periodic traffic lights especially in low-traffic scenarios. Therefore I have elected to make this my baseline. I have run the simulation with a simple periodic action sequence which loops through the action space one after the other. The results are shown in the following graphs.

## 6 Conclusion/Future Work

I have yet to begin working on implementing the neural network architecture for the Q-function approximator. The necessary components are in place, however, to facilitate this. I will need to develop the architecture and train it, expecting convergence on Q-values that will lead to an optimal policy. The issue of convergence is one that Vidali addressed by making use of experience replay. This reduces the effect of correlation between subsequent training examples as well as mitigates the recency bias effect seen with RL by refreshing the model on earlier states too.

## References

[1] Andrea Vidali. "Simulation of a traffic light scenario controlled by a Deep Reinforcement Learning agent". MA thesis. University of Milano-Bicocca, May 2018.

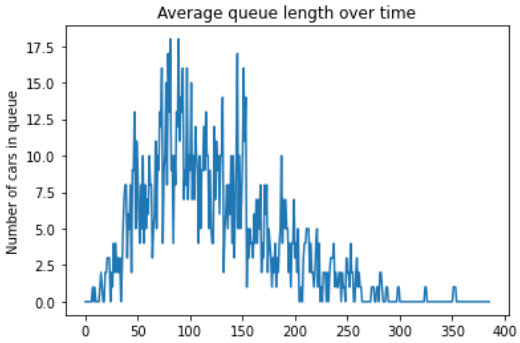



Figure 2: figure

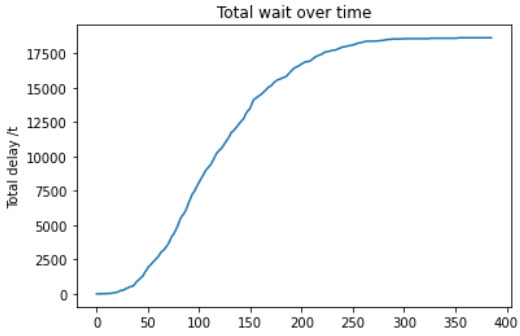



Figure 3: figure

[2] Daniel Krajzewicz et al. “Simulation of modern traffic lights control systems using the open source traffic simulation SUMO”. In: *Proceedings of the 3rd Industrial Simulation Conference 2005*. EUROSIS-ETI. 2005, pp. 299–302.