Indian Crossroads Traffic Warden

Student A[1,2] and Student B[3]

[1]*Department of Mathematics and Informatics, St. Kliment Ohridski University of Sofia, 5 James Bourchier Blvd, 1164, Sofia, Bulgaria*
[2]*Ablera*
[3]*Department of Physics, St. Kliment Ohridski University of Sofia, 5 James Bourchier Blvd, 1164, Sofia, Bulgaria*

(Dated: January 23, 2021)

Abstract: Traffic congestions contribute by a large part to people's loss of time and money in unforeseen situations. The main problem occurs when drivers are stationary at an unregulated crossroads, and do not take the optimal actions. Sometimes drivers are not concentrated enough and hence they do not fully optimize their actions in a given situation. Often at crossroads the preceding vehicle needs to exit the crossroads, in order for another one to enter. We propose a solution to this problem. We train a deep reinforcement learning agent, using an Advantage Actor Critic algorithm, which constantly controls the vehicle and the curvature of each vehicle in the crossroads, until a car leaves successfully the crossroads. Our ablation study shows promising results. The solution could be introduced to self-driving vehicles, and after further development could limit the needs of traffic signals. Our contribution in this paper is two fold. Firstly, we created an open-source environment which can be used for testing and development of reinforcement learning algorithms. We also created a solution for a problem, which to our knowledge, has not been created yet.

Problem Statement: Traffic congestion is becoming an increasingly more impactful problem in our every day lives. A lot of statistics and researches are published annually, which aim to point our attention to this problem and the consequences it brings. Statistics show that the average number of hours per year a U.S. driver spends in traffic is 41. This numbers is a lot bigger in some of the more congested states, like Los Angeles, where residents spend an average of 102 hours annually in traffic. Some solutions to this problem involves traffic lights, restrictions to traffic in peak hours, higher fees for vehicles, public transport, shared commute to work, and others. There are two main factors that greatly influence the outcome of a given traffic crossroads event. A traffic crossroads event in this paper is defined as the time spent in the crossroads by all cars.

The first factor, which influences the outcome of this event is the delay that occurs between each driver's personal evaluation of the event and the action the driver takes. The second factor is the accuracy of each driver's estimate of the situation. In this paper we propose an innovative solution to the problem, which consists of an intelligent system that learns to control the speed and the rotation of each car. This is the exact problem that we tackle using reinforcement learning. The objective of this research is to minimize the time spent on a crossroad, in a manner that is safe for all the participants in the traffic.

In the past, researchers have created various reinforcement learning agents that aim to control traffic in some sense, whether it be through solving conflicting traffic signals via phase competition [1], managing traffic light signals based on the amount of traffic [2] or through managing the whole traffic system at once [3]. Although all the above-mentioned methods are successful, we would like to focus our research on a topic which, to our knowledge, has not yet been tested. Our goal is to create a reinforcement learning algorithm, which will act as a traffic warden at an unregulated traffic crossroads. In our project, we focus our attention on crossroads with no traffic lights. Furthermore, we would like to maximize the number of vehicles that pass through the crossroad, while keeping the least number of stationary vehicles possible. For reference, check the following video of a crossroad in India [4].

Some non-trivial questions that we answer are:

- Can the full area of the traffic crossroads be used, but in a more controlled way than in the video, hence achieving better performance - less time spent on the crossroad by each car?
- Can the problem be solved if each vehicle has only one point in space, which satisfies the condition of an exit?
- Can all vehicles move at the same time, without waiting for each other to evaluate the different actions? Instead of using multi-agent reinforcement learning setups, we propose the usage of a single agent, which controls all vehicles in the environment, while modifying each car's velocity and curvature.

Reinforcement learning is a suitable approach to this problem, because it gives the possibility to explore and evaluate different scenarios. The problem can be approached in two ways: model free algorithms and model-based algorithms [5], [6]. The latter use a predictive model which learns the outcomes of taking certain actions. In contrast, the model free methods do not use a predictive model, instead they learn a control policy directly. Model-based methods are guided by a predictive model, while model free methods learn the environment dynamics through exploration of the possible actions. Although recent approaches have combined model-based and model free approaches [7], our team decided to create a model free model. In this task, the agent needs to explore different scenarios and estimate the q-values at

FIG. 1: All four cars start at the four entrances of a four-ways intersection. Each of the cars gets assigned, at random, a target exit, which is one of the other 3 exits.

FIG. 2: All four cars move simultaneously inside the crossroads.

FIG. 3: The goal of the agent is to learn to drive each car to its respective target exit.

each step. The q-values in this setting show how good a pair of actions - velocity, curvature - are for a certain car in a certain state, looking at all other cars.

Environment Design: We created our environment. Our environment is a bounded 10 x 10 square box, with 4 exits at the exact centers of the sides of the square. Four cars are spawned at the four exits (one at each exit) and each of the four cars is assigned a target exit at random - one of the other three exits. The intuition behind this is that each car has an exact exit that it needs to reach in order for the task to be considered successful. Each car is represented by its x and y coordinates, so the state space is a 4 x 2 matrix where each row is the pair of coordinates of a car. A car can only exit from the target exit. Each car is represented by a circle with a center. The cars cannot leave the specified area of the crossroads, unless they reach their respective target exits. A car crash is defined using the Euclidean distance between two points. The "unsafe distance" is a hyper parameter in the environment. If the center points of two cars are located at a distance lower than the "unsafe distance" hyper parameter, a car crash occurs. For any non-moving car, the agent receives a negative reward. At each step of the environment, all of the above-mentioned conditions are checked and the total reward is calculated. The velocity is calculated as follows:

$$V_2 = V_1 + a/10$$

We divide the acceleration by a factor of 10, because we would like to make smaller steps in the environment, which would result in a more continuous movement of the cars in the environment.

Supplementary Rewards:

- time reward - a positive reward for each car which is not static
- static reward - a negative reward for each car which receives a velocity of 0
- crash reward - a negative reward for each crash between two cars
- success reward - a positive reward for each car which successfully exits through the target exit
- close-to-success reward - a positive reward for each car which is located in a close proximity to its target exit
- boundary reward - a negative reward for each car which tries to leave the boundaries of the environment.

The total reward is the summation of all supplementary rewards. In order to move a car from point A in the crossroads to a point B, using its velocity and rotation, we use the following formula:

$$x_2 = x_1 + d * cos(ad * \pi/180)$$
$$y_2 = y_1 + d * sin(ad * \pi/180)$$

where $x1$ and $y1$ are the initial coordinates of a car, d is the velocity of the car, and ad is the angle in degrees of a car's rotation.

Methodology: The purpose of our agent is to learn to move each car towards its target exit. Our agent observes the state space and modifies only the acceleration and the rotation of each car. We have constructed the problem in an episodic scenario. The complexity of the task consists of the continuous nature of the action space and the state space. A suitable approach to the problem are the policy gradient methods, which try to directly optimize the policy. These methods are more suitable than value based methods, because in a continuous action space the value based methods cannot explore all possible actions, because there are infinitely many such actions. The policy gradient method also has a significant drawback. This method simulates taking actions in

an environment, in order to calculate the final reward of the episode. Once the algorithm compares the rewards of the different episodes, it chooses the actions that maximize the reward.

However, the significant drawback is hidden in the Monte Carlo simulation itself. The policy gradient method does not have a way to avoid taking bad actions. It takes the actions that maximize the reward in an episode. This may be harmful, as certain bad actions could be taken, no matter that they are bad, as long as they maximize the reward. One possible solution to this problem is the Actor Critic methods. This family of algorithms makes updates the policy at each step. Recall that the Reinforce algorithm waits for the whole episode to end, and then computes the total reward. The main drawback of the REINFORCE algorithm, defined as:

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta log_{\pi\theta}(a_t|s_t)(G_t) \tag{1}$$

where the total reward is G_t

The two main problems which may occur with the REINFORCE algorithm, are that the sum of the rewards in a trajectory may sum up to 0, effectively zeroing the gradients, and that the gradients have a high variance. One way to overcome these problems is to substract a baseline from the cumulative reward. However, a better solution is to introduce a value function which estimates the cumulative reward, instead of waiting for an episode to finish to calculate it. These are the Actor Critic methods.

There are many variations of the Actor Critic methods. We decided to implement an Advantage Actor Critic algorithm (A2C). The A2C algorithm is a type of a policy gradient algorithm. An Actor Critic algorithm consists of two parts:

- critic - estimates the average value of a state
- actor - updates the policy in the direction suggested by the critic

Different Actor Critic algorithms use different value functions (Q-value, Advantage, Temporal Difference). The Advantage function is defined as the difference between the Q-value at a certain state for a certain action and the value of that state.

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t) \tag{2}$$

where the Q-value is defined using the Bellman equation as:

$$Q(s_t, a_t) = E[r_{t+1} + \gamma V(s_{t+1})] \tag{3}$$

The action-value function (Q-value) is the expected return of a certain action in a given state. It is the expectation of the return of the next state, having taken a certain action, summed with the discounted value of the next state. The action-value function emphasizes the importance of taking a specific action in a given state. This allows us to rewrite the equation for the Advantage function as follows:

$$A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V_v(s_t) \tag{4}$$

The update equation for the Advantage Actor Critic algorithm, parameterized by v is given by:

$$\begin{aligned}\nabla_\theta J(\theta) &\sim \sum_{t=0}^{T-1} \nabla_\theta log_{\pi\theta}(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)) \\ &= \textstyle\sum_{t=0}^{T-1} \nabla_\theta log_{\pi\theta}(a_t|s_t) A(s_t, a_t)\end{aligned} \tag{5}$$

The intuition behind the Advantage function is that it provides a measure of how much better it is to take a specific action in a state, in comparison to the average value for this state. From equation (1) we can clearly see that the actions that are better on average in a given state would result in a positive advantage and vice versa.

Because in our environment the action space is defined as a 4 x 2 matrix, where each row represents the pair of acceleration and rotation for each car, we construct 3 neural networks to estimate four parameters, given the 4 x 2 matrix which represents the state space. Recall that the state space presents the coordinates of each car (x, y). The parameters that are approximated using neural networks are:

- value - the average value of a given state (a real number)
- alpha - alpha parameter of a beta distribution (a positive number)
- beta - beta parameter of a beta distribution (a positive number)
- sign - the sign of the acceleration (a 1d array with probability distribution) - we would like to model the cars accelerate and decelerate when needed

In our problem formulation, we define the acceleration as the negative of a sample from a Beta distribution, constructed using the alpha and beta parameters from the neural network, iff $argmax(sign) = 1$.

Results: The trained system is able to maximize the rewards from the environment, and to successfully drive a car towards its target exit. Our experiments show the systems is very unstable, as it heavily relies on the ratios between the different rewards. We trained the system for 2 500 episodes, where each episode consists of 10 000 steps. We explicitly chose such long training procedure, because the positive rewards in the system are sparse, and the agent needs a substantially large explored space of state-actions. Reaching a single target point of exit turns out to be impossible by our algorithm. The problem is solvable if the target exit is presented as a collection of states between two points. It is possible to solve the problem while all vehicles are moving, and last, but not least, we make sure our algorithm optimizes a significantly larger portion of the area, compared to what human drivers would.

Implementation Details: We construct 3 neural networks. The first NN consists of 2 Linear layers with a Relu activation between them. Each layer has 200 hidden neurons. The second NN consists of 2 Linear layers with a Relu activation, and a Softmax activation for the latter. The softmax layer outputs the probability distribution for the sign of the acceleration. The third NN consists of 2 shared Linear layers with a Relu activation between them and 2 heads. Both heads consist of a single Linear layer, which outputs the alpha and the beta parameters respectively, which are used to model the beta distribution. We clamp the outputs for the NN which estimates the alpha and beta parameters to a range between 1.5 and 2.5, because we found this works good in practise. We use Adam optimizer with an initial learning rate of 0.0005 which decreases by a factor of 10 every 30 episodes. Furthermore, we connect the agent to Unity. Let's start by explaining what we are aiming to do with Unity. Our goal with this project is to simulate the movements done by the agent in a more animated way, which is easier to understand visually. In order for us to do that, first we need to build the environment, i.e the crossroad and the cars themselves. Let's start with the car, there are several ways we could make the car move in Unity, for example we could give the object speed and direction, or a point towards which it moves, we could give it only acceleration at certain intervals and make it move like that. For our purposes we will need to make it in such a way, that for each step our agent makes, we can reproduce that step in the game itself. To that end we tell the car to move with speed and rotation, i.e it moves towards a certain angle. Because want to simulate 2D movement, we exert a force, with the z value being a constant, and the x and y are the cos and sin of the angle accordingly. This is done by applying a vector to the car, that follows a formula, almost identical to the one used by the agent itself . And now we have a moving car, that will move accordingly to the agent. The next step is to read the starting positions, exit positions, speed and rotation from the agent. We will use a text file, divide it in lines, and the lines themselves will be further divided into the 4 values that each car needs. The last step we have to do is match up the time of each step the car makes in Unity, with the time of each step in our agent. To achieve that we just limit the amount of updates the car gets from the agent per second. That's it, we've got a system where the car will move in whichever direction the agent tells it to, whenever it tells it to, so that in the future when we expand this project for more complex crossroads, we need only to train the agents. [8].

Future Work: The problem could be extended in different directions: different environments, different reinforcement learning paradigms. Currently, the authors of the paper are currently working on a model-based solution in order to compare it with the current results produced by the A2C algorithm. Another addition is to the project could be an integration with convolutional neural network models, which would take as input images from a camera located above the crossroads.

Acknowlegements.— We thank Professor Marin Bukov for teaching the course of Deep Reinforcement Learning. The authors are pleased to acknowledge that the computational work reported on in this paper was performed on an NVIDIA Tesla V100 32GB.

[1] Zheng, Guanjie, et al. Learning Phase Competition for Traffic Signal Control. 12 May 2019, arxiv.org/abs/1905.04722v1.

[2] IntelliLight: A Reinforcement Learning Approach for ... faculty.ist.psu.edu/jessieli/Publications/2018-KDD-IntelliLight.pdf.

[3] hzw77, and hzw77. Hua Wei, 15 Sept. 2019, sites.psu.edu/huawei/2019/09/15/colight-cikm-2019/

[4] Indian Crossroads video, https://www.youtube.com/watch?v=7aSkJCUDAes

[5] Hui, Jonathan. "RL - Model-Based Reinforcement Learning." Medium, Medium, 2 Nov. 2019, jonathan-hui.medium.com/rl-model-based-reinforcement-learning-3c2b6f0aa323

[6] I. (2018, October 29). What is Model-Based Reinforcement Learning? - the integrate.ai blog. Medium. https://medium.com/the-official-integrate-ai-blog/understanding-reinforcement-learning-93d4e34e5698

[7] Feinberg, V. (2018, February 28). Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning. ArXiv.Org. https://arxiv.org/abs/1803.00101

[8] Our official implementation can be found at https://github.com/MihailMihaylov97/indian$_c rossroads$