Reinforcement Learning Algorithms With Applications to Traffic Flow Control

CS221 Artifical Intelligence Project Progress, Spring 2020

Akshita Agarwal
akshita@stanford.edu

Khalid El-Awady
kae@stanford.edu

Mayank Gupta
gmayank@stanford.edu

I. Introduction

Autonomous systems are a very active area of industrial application, such as in manufacturing, exploration, automation and others. These systems are characterized by a need for decision making under uncertainty. Reinforcement learning (RL) is a very general framework for learning sequential decision making tasks.

In this project we study the class of RL algorithms and their applicability to a real world problem, namely **traffic flow control**. This represents a non-trivial example where RL has been shown to yield results on a real-world complex system [8], [9]. Further, a simulator has been identified that can readily provide input/output data [10].

II. Related Works

Transportation accounts for 28% of energy consumption in the US. Workers spent on aggregate over three million driver years commuting to their jobs [1], with significant impact on nation-wide congestion. Based on 2012 estimates, U.S. commuters experienced an average of 52 hours of delay per year, causing $121 billion of delay and fuel costs annually [2].

This extreme expenditure in cost, lost time, pollution, and quality of life, has led to a large investment in the study of traffic and its control and creating simulators to model traffic flow and allow the testing of new control schemes. One such simulator that has received acclaim and attention is SUMO – Simulation of Urban MObility, an open source, highly portable, microscopic and continuous road traffic simulation package designed to handle large road networks [3], [4]. We will utilize SUMO as the simulation engine behind our data generation. In addition to SUMO, we will be using Flow, an open-source computational framework for RL and control experiments [5].

The works of [7], [8] were instrumental in demonstrating the applicability of RL to improving traffic flow through agent-controlled traffic lights. Much of the current research in RL control of traffic lights revolves around implementations of deep learning algorithms [9]–[11].

In this project our objectives are to:

- Setup a simple traffic light environment with 2-way traffic flow in SUMO with a typical model of vehicle arrivals,
- Simulate the flow of the traffic through the intersection using a typical static control agent and compute statistics around vehicle waiting time and speed – this will be our baseline,
- Implement a basic RL algoritm to learn an alternative policy for control of the light and compare the resulting vehicle statistics to those of the baseline,
- Explore one or more advanced RL algorithm to attempt to improve performance beyond what was achieved in our first attempt.

In this progress report we present and compare the results of the baseline and initial RL-based approach, and propose our next steps for the modifications we will look at in the balance of this project.

III. Dataset

In SUMO, a user defines a network which represents the elements of the traffic environment to be simulated. In our case the environment is made up of a north-south road met by an east-west road at an intersection controlled by a traffic light. A screen shot of the environment is shown in Figure 1.

Our network represents a pair of local city roads with a speed limit of 35 mph and an enforced minimum gap between cars of 2.5 secs of travel (subject to an assumed car deceleration rate of 7.5 m/h^2). Cars can enter the network from the endpoints of any of the respective four arms of the network at a distance of 300 yards away from the intersection. The car proceeds to the intersection and either passes through it if the light is green in that direction, or waits if the light is red. Once the car passes through the intersection it continues for 100 yards until exiting the network and the simulation.

SUMO injects new car traffic into the network at the beginning of each edge using a uniform distribution with a given mean inter-arrival time while also enforcing our prescribed minimum distance between cars. To create some variation in the flows we have chosen to vary the rate of cars entering each of the arms. The reader might be able to visually discern the difference in traffic flows from the figure (the bottom arm shows much higher density than the top arm for example).

The traffic light experiences so-called phases which represent the light's color in each of the four directions. A typical static light might cycle through phases of red and green, each of length 30 seconds each with an optional short transition period when the light that was green turns yellow. A typical simulation might last for 300 seconds and involve around

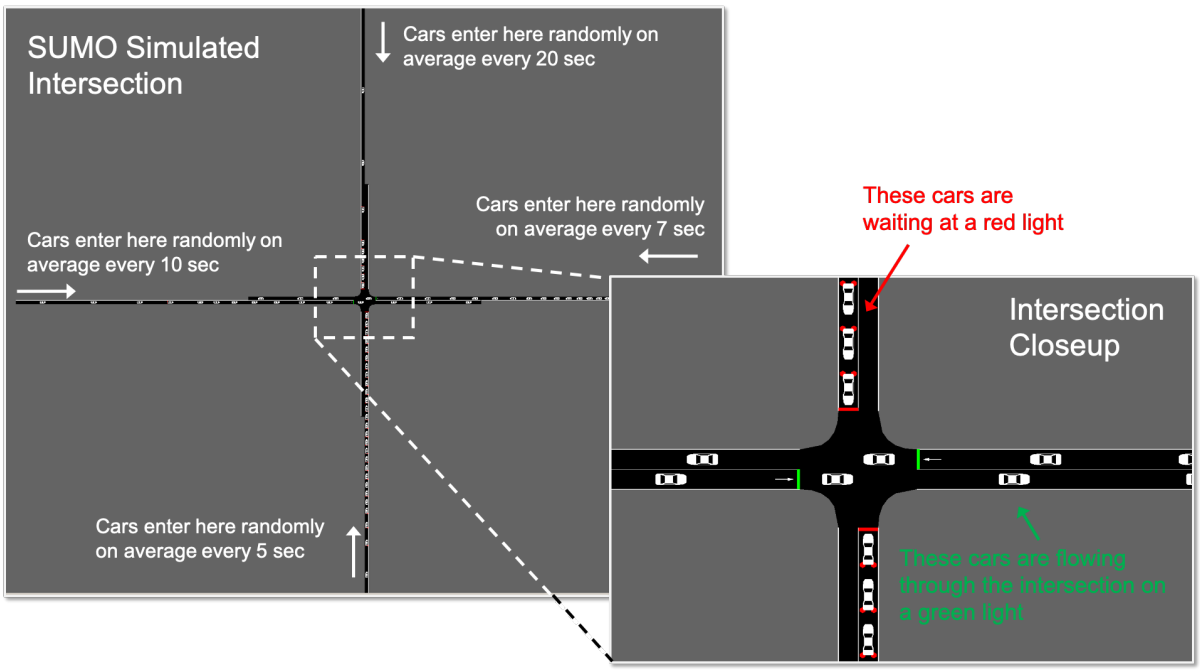


Fig. 1. Screen shot of the SUMO simulation environment of a simple intersection with a traffic light.

2,500 cars. A car traveling at around 20 mph that does not stop at the intersection will pass through the network in under 30 sec.

The output of the simulator is a file of data called an ”emissions” file that comprises a set of time samples for each vehicle in the simulation with a set of vehicle parameters at the designated simulation sample output interval. The main data points of interest to us in the emissions file are the ’waiting’ time and ’speed’ parameters. The waiting time represents the cumulative time (in secs) the vehicle has been waiting as of that time sample. Speed is a measure of the instantaneous speed of the car at the time sample.

Figure 2 shows a sample of the vehicle data for one vehicle (id: flow_00.10). In this vehicle’s case, it enters the simulation at time 40 with a speed of 10 mph heading west to east. The vehicle eventually decelerates to a stop at the traffic light by time 65. It then waits a cumulative 31 seconds at the intersection before accelerating and leaving the simulation at time 113 going at 13.99 mph.

time	id	waiting	speed
40	flow_00.10	0	10
45	flow_00.10	0	13.63
50	flow_00.10	0	14.02
55	flow_00.10	0	13.7
60	flow_00.10	0	2.91
65	flow_00.10	1	0.1
70	flow_00.10	6	0
75	flow_00.10	11	0
80	flow_00.10	16	0
85	flow_00.10	21	0
90	flow_00.10	26	0
95	flow_00.10	31	0
96	flow_00.10	0	0.6
100	flow_00.10	0	5.28
105	flow_00.10	0	11.1
110	flow_00.10	0	12.61
113	flow_00.10	0	13.99

Fig. 2. Sample data from one vehicle in the SUMO simulation.

IV. Baseline

Our baseline will be a traffic light that employs static phases. There are two phases that our light assumes. These are: 1)green in north-south direction, red in east-west direction – 20 sec, and 2)red in north-south direction, green in east-west direction – 20 sec. The durations were chosen based on recommended ’typical’ values suggestion by Flow.

Figure 3 shows the results of the static traffic light simulation. The top plot shows the histogram of wait times for cars. From the plot we can see two roughly equal cohorts: ”flowers” who generally breeze through the intersection and wait little, if at all, and ”waiters” who end up at the traffic light. The waiters show a most likely wait time of 16 seconds. These results would seem to match intuition.

In addition we show the histogram of average car speed through the network. The distribution shows a mean of 6.3 mph, which will be another importance baseline metric for comparison with our RL agent.

V. Main approach

During this phase of the project, we successfully setup Flow libraries and the SUMO simulator and implemented two approaches to the Traffic Light Control Problem:

1) Fixed duration Control. This is the baseline for our system as explained above.
2) Deep Q-Learning algorithm.

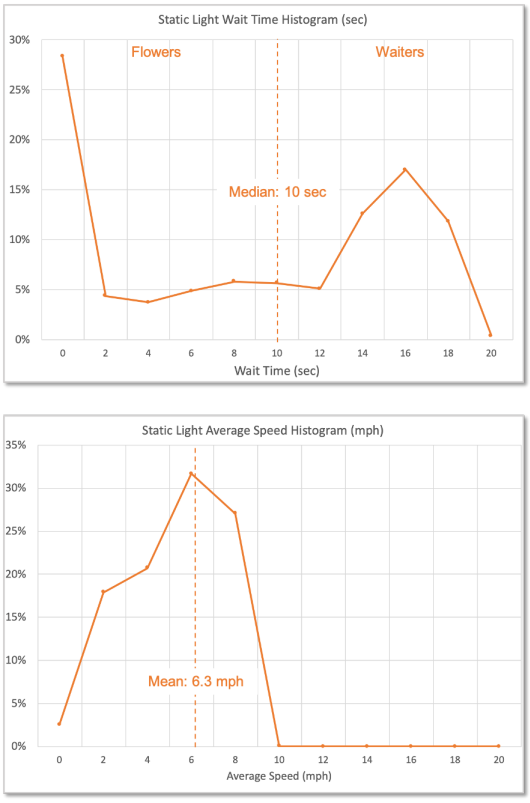


Fig. 3. Baseline metrics: distribution of wait times at the intersection and average speed through the network for a static 20 second traffic light.

A. Experiment Setup

1) Environment: : We have used the TrafficLightGridEnv available in Flow. This environment acts as a simulator and performs the change of state of the system given the action returned by the RL agent. It also gives the reward to the agent based on the action and the current observation space.

2) State: : The state includes key parameters for each vehicles currently present in the network, including:

- the distance from the vehicle to the intersection,
- the speed of the vehicle, and
- Which edge the vehicle is on.

3) Action: : The action space is Discrete with values 0 or 1, signalling whether the traffic light should switch its state or not. The corresponding phase of the four traffic lights is encoded in a string representing the color of each of the four lights: 'GrGr', 'yryr' or 'rGrG'.

4) Reward: : The reward is negative of the cumulative vehicle delay less a penalty for vehicles standing still.

B. Algorithm - Deep Q Learning

The Q-Learning algorithm is used to learn the Q values for a given state and action using the Bellman update equation.

$$Q^{\star}_{(}s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s,a) \max_{a'} Q^{\star}_{(}s',a') \quad (1)$$

However, Q learning is a tabular algorithm, meaning it needs to compute Q values corresponding to all states and actions. In our case, the state space is very large. The number of cars in the network at any one time can vary between 0 and 100 with positions in the range of 0 to length of the road under observation and speeds between 0 and our maximum allowed 35 mph. For such a large state space, a tabular approach doesn't perform well. Hence we applied the **Deep Q Learning** algorithm that takes in a state and gives the action based on a multilayer perceptron that essentially performs function approximation on the Q values. The algorithm for this is given below:

Algorithm 1 Deep Q Learning for Traffic Flow Control

$Q \leftarrow 0$
$s \leftarrow InitialState$
$done \leftarrow False$
while $done \neq True$ **do**
  $a \leftarrow Agent.GetAction$
  $s', r, done \leftarrow Simulator.Step(a)$
  $tdError = Q(s,a) - (r + \gamma max'_a Q(s',a'))$
  Backpropagate the error to update the neural net

end while

1) Training: We used Open AI baseline [**?**] DQN algorithm with mlp (multi-layer perceptron) network which had 2 fully connected layers with 64 hidden units each. We used following hyper-parameters for learning:

- Learning rate (η): e^{-3}
- Exploration fraction (ϵ): start with 0.1, goes till 0.02
- Total time-steps to train: 20000

2) Evaluation: During training, the trained policies and checkpoints were stored in a pickle file were loaded during the evaluation. The experiment was setup to have the state action loop. Finally we evaluated the policy by running this created experiment on TrafficLightGridEnv for 500 episodes and reporting the average results.

VI. Evaluation Metric

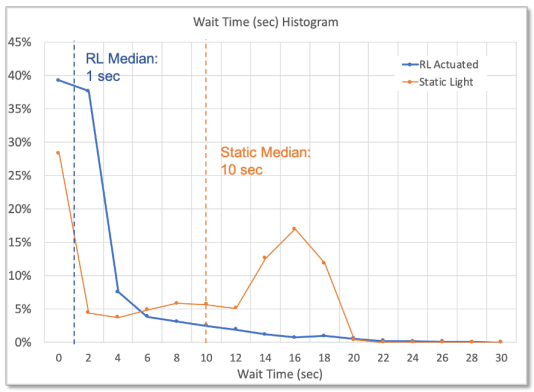
Our evaluation metrics will mirror those used in the baseline case: namely the distribution of wait times and average speed through the network. See the following section for a description and analysis of the results.

VII. Results Analysis

Figure 4 shows the performance of the RL agent. For the wait times, we notice a dramatic improvement in the wait times. The median wait time drops to 1 sec, fully one-tenth that of the static light. The wait time does exhibit a fairly long tail, though, and appears to tradeoff very low wait times for the many against a few longer waits for the few.

The improvement in average speed though is not as dramatic. Here we see an improvement of 10% or so in mean, though a larger portion of the cars see higher speeds. This is another artifact of the long tail of the wait time distribution.

It is instructive to observe the policy generated by the RL agent. This is shown in Figure 5. The plot shows the phases



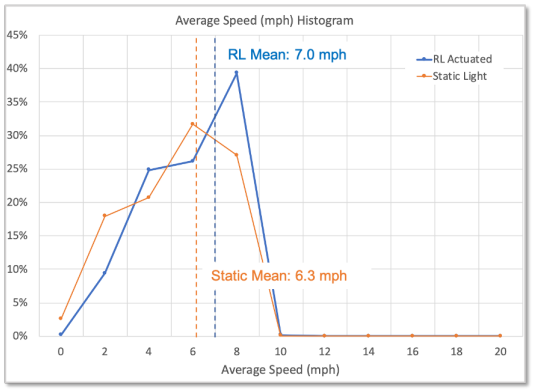


Fig. 4. RL agent performance: distribution of wait times at the intersection and average speed through the network with comparison to the baseline static 30 second traffic light.

of one of the lights (with the other being the inverse of this). Comparing this to the static light policy of 20 seconds green - 20 seconds red, we observe:

- The RL policy results in more frequent switching. Over our 300 sec simulation the RL agent has 19 green phases (vs 15 for a static agent with a 20 sec cycle).
- The phases are often very short. 11 of the 19 phases lasted only 4 seconds. One phase lasted 33 seconds (the first). There was also one 19 sec phase, one 11 sec, and the remaining 5 phased lasted 6 or 7 seconds.

We note that phases of 4 seconds long are probably too short from a practical standpoint. So while the RL agent provides dramatic reductions in wait time, the policy is not likely to be applicable in the field. One area of future investigation is to attempt to tune this behavior by making appropriate changes to the reward function to impose a larger penalty on frequent switching and short duration phases.

VIII. Future Work

There is lot of room for improvement on the current RL policy as well as to expand the problem and make it more challenging. A number of areas we will consider exploring in the balance of the project are:

1) Rewards tuning: currently, the reward function prefers higher speed of traffic and penalizes standstill traffic. We can add more factors such as penalizing for too frequent switching of the lights.
2) Adding more lanes and allowing left and right turns in the network. This will expand the action space and make the problem more complex and analogous to real life traffic lights.
3) Trying out more sophisticated RL approaches such as Experience Replay or Double DQN and see how they perform on this problem.
4) Using a richer observation space for training – currently we use only the 2 nearest vehicles from intersection in each edge.
5) Framing the problem with continuous action space and see the performance.

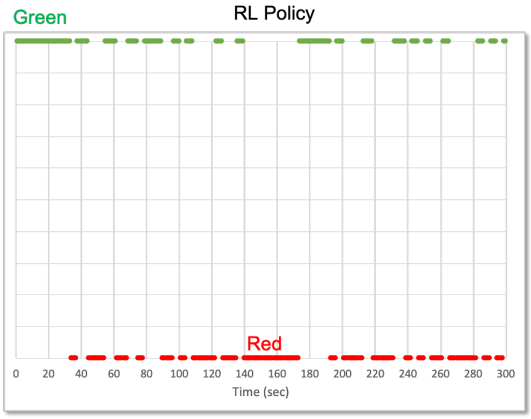


Fig. 5. RL agent policy (for one of the lights): The RL agent results in more frequent light switching and shorter durations than the static agent.

References

[1] U. DOT, "National transportation statistics." Bureau of Transportation Statistics, Washington, DC, 2016.
[2] D. Schrank, B. Eisele, and T. Lomax, "Ttis 2012 urban mobility report," Texas AM Transportation Inst. The Texas AM Univ. System, 2012.
[3] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, International Journal On Advances in Systems and Measurementspenalize standstill of traffic, vol. 5, no. 34, 2012.
[4] https://sumo.dlr.de/docs/index.html.
[5] https://flow-project.github.io/index.html.
[6] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control
[7] V. Joshi. "How reinforcement learning can help in solving real-world problems?
[8] I. Arel, C. Liu, T. Urbanik, and A.G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control", IET Intelligent Transport Systems, 2010, Vol. 4, Iss. 2, pp. 128– 135.
[9] G. Marion. "Deep reinforcement learning on a traffic light system," https://arxiv.org/pdf/1803.11115.pdf.
[10] Flow: A Deep Reinforcement Learning Framework for Mixed Autonomy Traffic. Mobile Sensing Lab, University of California Berkeley. https://flow-project.github.io/index.html
[11] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, A. Bayen, "Flow: Architecture and Benchmarking for Reinforcement Learning in Traffic Control," CoRR, vol. abs/1710.05465, 2017.
[12] W. Genders and S. Razavi, "Using a deep reinforcement learning agent for traffic signal control," arXiv preprint arXiv:1611.01142, November
[13] OpenAI Baselines, https://github.com/openai/baselines