Reinforcement Learning for Traffic Optimization

Matt Stevens MSLF@STANFORD.EDU
Christopher Yeh CHRISYEH@STANFORD.EDU

Abstract

In this paper we apply reinforcement learning techniques to traffic light policies with the aim of increasing traffic flow through intersections. We model intersections with states, actions, and rewards, then use an industry-standard software platform to simulate and evaluate different policies against them. We compare various policies including fixed cycles, longest queue first (LQF), and the reinforcement learning technique Q-learning. We evaluate these policies on a varying types of intersections as well as networks of traffic lights. We find that Q-learning does better than the fixed cycle policies and is able to perform on par with LQF. We also note reductions in CO_2 emissions in both LQF and Q-learning relative to a fixed cycle baseline.

1. Motivation

According to a study by Texas A&M, Americans waste about 7 billion hours and 3 billion gallons of fuel in traffic each year (David Schrank & Bak, 2014). This makes up roughly 2% of all gasoline consumed in the United States (EIA, 2016). This means that reducing traffic can have a significant impact on people's lives, as well as their carbon footprint.

For this project, we aimed to reduce traffic by finding better traffic light policies at intersections. An ideal intelligent traffic light systems can reduce traffic through several techniques. It can turn lights green for longer in directions with more traffic, use sensors to dynamically respond to arriving cars and changing traffic conditions, and coordinate between lights to create runs of traffic that flow through many lights.

Reinforcement learning is a promising solution to this problem because it can represent all of these techniques. By using a general notion of actions, we can decide when to turn lights on and for how long. It excels at dynamic control and is designed to adapt to new conditions. Lastly, we can define our states to incorporate as much information about the system as we want and share it across many traffic lights to allow them to coordinate.

2. Algorithm

2.1. Q-Learning

We use Q-learning with function approximation to learn the best traffic signal actions. Before we detail our specific problem formulation, we first describe the Q-learning algorithm for a general Markov Decision Process (MDP). Under a MDP with fixed transition probabilities and rewards, the Bellman equation (Equation 1) gives the optimal policy.

$$Q(s,a) = R(s') + \gamma \max_{a'} Q(s',a') \tag{1}$$

If the Q function can be correctly estimated, then a greedy policy becomes the optimal policy, and we can choose actions according to Equation 2.

$$\pi(s) = \arg\max_{a} Q(s,a) \tag{2}$$

We use the function approximation abstraction from (Mnih et al., 2015). From our simulator we extract values for the state features, action, reward, and next state features: (s, a, r, s'). Note that the s variables are not states, but features of states. Thus, $Q(s,a)$ represents the approximate Q value for that state and action. Each (s, a, r, s') tuple is a training example for our Q function. In order to satisfy the Bellman equation (Equation 1), we minimize the loss between our current Q value and our target Q value over parameters θ subject to regularization λ (Equation 3).

$$\min_{\theta} \left(||r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)|| + \lambda ||\theta|| \right) \tag{3}$$

2.2. Q functions

The heart of Q-learning is the Q function used for estimation. In the naive case, the Q function can simply be a lookup table that maps states and actions to Q values, in which case Q-learning is essentially the same as value iteration. However, Q-learning lets us generalize this framework to function approximations where the table of states and actions cannot be computed.

Every part of Equation 3 is differentiable, so if our Q function is differentiable with respect to its parameters, we can run stochastic gradient descent to minimize our loss.

For our implementation, we use stochastic gradient descent on a linear regression function. We also performed SGD with a simple one-layer neural network.

3. Problem Formulation

We approach this problem as a MDP with states, actions, and rewards.

3.1. Simulator

To model an intersection as it would exist in the real world, and to evaluate our policies, we followed other researchers (Covell et al., 2015) in using SUMO (Simulation of Urban MObility), an open-source industry-standard software package for modeling road networks and traffic flow (Krajzewicz et al., 2012). In particular, we used SUMO version 0.26. SUMO allowed us to build different types of road networks, add cars to our simulation and determine their routes, and add sensors for the traffic lights.

The SUMO package also comes with TraCI, a Python API that allows a user to get information from a traffic simulation and modify the simulation as it runs in response to this information. We built an architecture that made use of the TraCI interface to get information about queue sizes, carbon emissions, sensor data, and traffic light states, in order to successfully deploy our algorithms.

3.2. Setup

Our road network featured 4 connected traffic lights in a 2x2 square grid, spaced 150m apart. Each traffic light was also connected to a 425m-long road, which we used to mimic a highway on/off ramp. Thus, we had a total of 8 source nodes and 8 destination nodes. Each road contained 3 lanes with a speed limit of 45 mph, and we used SUMO's default configuration for left-turn and right-turn lanes. Each traffic light was set to allow right-turns on red. Yellow lights were set to be 5 seconds long. On the roads in between the traffic lights, induction loop sensors were placed in each lane about 50m before each traffic light; on the roads leading into the network, the induction loops were placed about 100m before each traffic light. We used SUMO's default car configurations. Every second, for each possible $(source, destination)$ combination, we generated a car at with probability 0.01. Figure 1 shows the road network that we built.

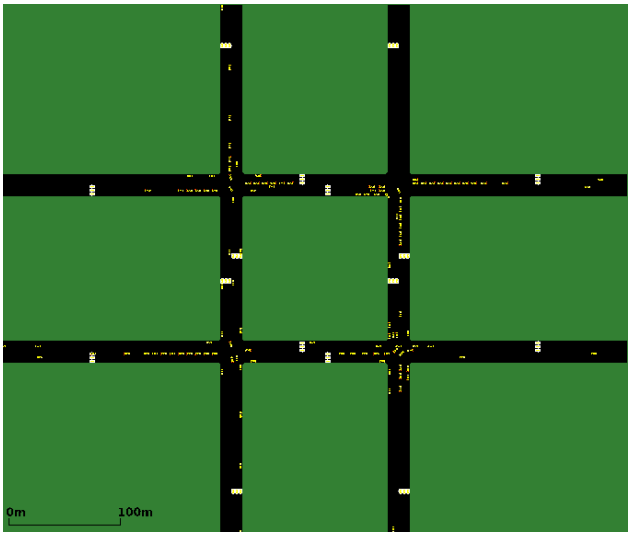

Figure 1. A zoomed-out view of the road network that we trained and tested our Q-learning algorithm on. The small yellow dots are cars traveling through the network, while the larger yellow rectangles represent induction loop sensors

3.3. Objective and Reward Function

There are many ways to formulate an objective function for traffic optimization. Ideally, we would try to model the lost utility of drivers' time spent waiting in traffic, and the cost and environmental effects of wasted gasoline. These effects are all difficult to estimate. Fortunately, however, all of these effects are positively correlated with increasing traffic, so any metric the captures the general trend of the amount of traffic will capture the general trend of these effects.

We specifically chose the throughput of cars through intersections as our reward function, where throughput is defined as the number of cars that pass through the intersection per unit time. While there are many other reasonable measures of the amount of traffic (e.g. wait time, CO_2 emissions, and total distance traveled in a given time interval), we chose throughput as our reward function for two reasons. First, this reward is more directly tied to the action taken, meaning that the learning algorithm has to do less work separating the signal from the noise in the rewards. Second, the reward varies more linearly with traffic flow than other metrics, which makes it easier to fit.

SUMO does not provide a direct way to calculate throughput, so instead, we calculated the sum of the speed of cars through the intersection per time step. This sum approximates a discrete integral of speed over time, giving the total distance traveled by the cars through passing through the intersection. Then, the total distance traveled divided

by the width of the intersection is equal to the number of cars that pass through the intersection. Thus, our reward function is proportional to throughput, and off by a constant factor of the length of the time step and the width of the intersection.

3.4. Actions

We formalize each traffic light as an agent that can perform a set of actions. After each 15-second interval, the traffic light chooses which direction to turn green. For example, a traffic light with no left turns may choose from the actions (North/South green, East/West green). We designed the traffic lights in our network to allow left turns, so they had two additional actions. While some approaches use signal durations as actions (Salkham et al., 2008), we chose this parameterization because it offers more flexibility. (Arel et al., 2010) We abstract these actions away so that the agent only knows that there are several distinct actions it can perform, and not which directions they correspond to.

3.5. Features

Our goal is to have each agent (i.e. traffic light) learn the optimal policy (i.e. which direction to turn green) based on inputs that would be available in the real world. To this end, we give our model three types of features:

1. **Sensor Data:** SUMO has the capacity to simulate induction loop sensors. In the real world, these loops are generally installed under streets to provide nearby traffic lights with information about cars passing above them. In our simulation, we placed induction loops in each lane before every traffic lights, for a total of 12 induction loops per traffic light. These induction loops inform their respective traffic lights with the number of cars that have passed over them in the last time step, along with the average speed of these cars. We also provide each stoplight with the previous five time steps worth of sensor information.

2. **Stoplight History:** We provide each stop light with the previous five time steps worth of its phases. A phase in this context refers to the specific permutation of lights colors for each lane in the intersection. Each phase is represented by a number in the feature array.

3. **Features from Adjacent Traffic Lights:** In order for each traffic light agent to learn to coordinate with the other agents, we provide each agent with the features given to the adjacent stoplights, as in (Salkham et al., 2008).

In total, we represent each state with 420 features.

Due to our backpropagation algorithm, it was important to normalize features to the same range. The gradient value is multiplied by the feature value during backpropagation, so features with high values get high weights during training. And since they had high values to begin with, their influence is increased quadratically in the final regression value.

4. Experiments

4.1. Parameters

To figure out the optimal hyperparameters for the Q-learning algorithm, we used a combination of random and manual search. We ran a smaller number of training and testing iterations to achieve acceptable models and then extracted the average number of cars waiting as our metric for comparison.

The discount factor γ parameter in Q-learning had no significant effect on results. Our learning rate α and regularization λ were fairly standard. For linear regression we used $\alpha = 10^{-2}$ and $\lambda = 10^{-2}$. For our neural network we used $\alpha = 10^{-4}$ and $\lambda = 0.1$. If we decreased regularization or increased learning rate by too much, our function values exploded due to the recursive nature of the Q function.

In Q-learning, it is important to balance the need to explore the state space with the need to choose good actions. For this reason, we used an ϵ-greedy algorithm which chose the highest-Q action most of the time, but occasionally chose a random action. We found that too many random actions could cause disastrous results across the grid, including gridlock, making it very difficult for our algorithm to learn well. To address this, we used the LQF algorithm described below to choose a heuristically good action some fraction of the time instead of the random action. This gave our algorithm a warm-start, and helped prevent gridlock. We found that choosing the heuristic action instead of a random action $\approx 50\%$ of the time worked best.

We also realized that we might get better results if ϵ decreased over time, reflecting the fact that the algorithm needs to learn as it converges. To this end, we incorporated a parameter denoting the half-life of the epsilon parameter, so that it would decay exponentially over time. We found that setting the half-life to 200 time steps worked best.

4.2. Baseline

We implemented three other algorithms for comparison against our Q-learning approach to traffic optimization.

1. **Short Cycle:** Changes the phase every 15 seconds in a round-robin fashion.

2. **Long Cycle:** Change the phase every 45 seconds in a round-robin fashion.

3. **Longest Queue First (LQF):** LQF chooses to let the direction with the highest number of cars be green. Previous research (Wunderlich et al., 2008) has shown that LQF is robust algorithm even under high load. In the real world, it is not possible to directly find queue lengths, and we extracted this information directly from the simulator. Although LQF is a greedy algorithm, it is given more information than Q-learning, which makes it a reasonable target for comparison.

4.3. Results

We trained Q-learning on an episode of 1000 time steps using the ϵ-greedy approach as described above, and then used the parameters to test on a new episode, choosing the action with the highest Q value every time.

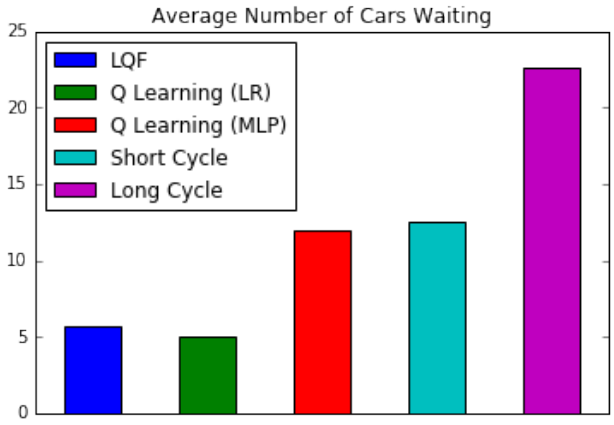

Figure 2. Average number of cars waiting at stoplights during the simulation. All differences are highly significant.

As shown in Figure 2, Q-learning with linear regression performs as well as our LQF baseline. Our neural network implementation performs slightly worse than linear regression. When we look at the CO_2 emissions data provided by SUMO, though, the performance is not directly correlated. we see that the two Q-learning algorithms are now tied, as well as the two cycle algorithms (Figure 3). We believe that this is a result of how Q-learning coordinates cars between the four intersections, reducing the amount of start-stop motion that creates the most amount of CO_2 emissions.

5. Conclusion

We believe that our linear regression algorithm performed so well because the hypothesis class of our features and actions is expressive, and tends to represent many common control paradigms for traffic lights. Cycles can be represented with the previous action features. For example, if we want to switch from action 1 to action 2 after four time steps, then action 2 can have a high weight for "performed action 1 four time steps ago." Actuated systems that respond to arriving cars can be represented by high weights on the sensor features. Lastly, networks that coordinate to let runs of cars through can be represented by our network as well by incorporating previous action features from neighboring lights. Looking at our feature weights and our simulation results, we saw all of these kinds of learning taking place together.

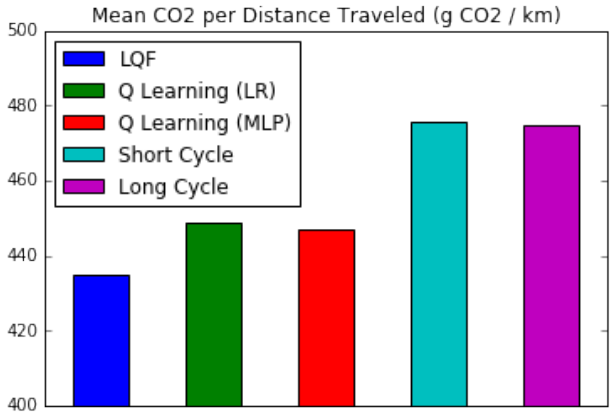

Figure 3. Average amount of CO_2 emissions per distance traveled by cars in the simulation.

Our linear hypothesis class was incapable of fully representing LQF, however. Estimating the number of incoming cars between the sensor and the light is simple. Estimating the number of cars beyond the sensor can only be done in expectation using features for neighboring lights and sensors. The fact that Q-learning can learn the number of cars arriving on average explains why Q-learning and LQF had comparable performance for average waiting time. And the fact that this estimation is noisy explains why we see more emissions in the Q-learning scenario, meaning that the cars stop and start more frequently as the light makes small errors. However, LQF should be representable as a general function of our features, if not a linear one. For this reason, we implemented a neural network model.

Our neural network model performed worse despite having a more expressive hypothesis class. In fact, because our neural network used a ReLU activation function, the hypothesis class of linear regression was a subset of that of our neural network. This means that under perfect training, our neural network can never do worse than a linear regression. Clearly, we were not training our neural net-

work perfectly. Since our hypothesis class was larger, the neural network had lower bias but higher variance. Since we are using a simulator, we have access to infinite training data, so in theory more iterations and a lower learning rate would solve these issues. However, these techniques had only modest gains for us, suggesting that the changes needed to match the performance of linear regression are drastic.

Overall, our results show that reinforcement learning is effective at learning good traffic light policies. Our algorithm was able to perform as well as LQF, which had more information about the system than any real-world algorithm would have available. Though not the optimal policy, it is a high bar to meet, and suggests that Q-learning can be a powerful technique.

One of the key takeaways of this work was that domain knowledge can be used to confine the hypothesis class to a set of more reasonable policies. We are not the first to come to this realization. Algorithms that are now standard for traffic control choose over a library of possible light timings (Jayakrishnan et al., 2001). This limits the hypothesis class to a set of actions that are all reasonable, but some are better than others. We did something similar in our model, in that our formulation of actions did not allow the controller to control yellow lights, so all of our actions were guaranteed to not cause a car crash. Clearly there is a tradeoff between expressiveness of actions and the potential for bad results. Future collaboration with traffic engineers to bound the problem and use domain knowledge to eliminate the worst actions may yield an algorithm with real-world applicability. Skeptical users of the system could be guaranteed that the algorithm would give reasonable results even in the worst case, which is something that our system cannot guarantee.

Q-learning shows promising results, and model-free systems like Q-learning can use the power of machine learning to discover trends that are overlooked by the heuristics and approximations of explicit optimization algorithms. A reinforcement learning system has the potential to provide adaptive control and coordination, theoretically matching the current state of the art. On top of this, multi-agent reinforcement learning is a distributed technique, which gives it fault tolerance as well as the potential to scale up to a larger network. For all of these reasons, we believe that reinforcement learning is a promising paradigm for traffic control.

6. Acknowledgments

We thank our CS 325 classmate Alex Tamkin (atamkin@stanford.edu) for his contributions to this project. We also thank the SUMO team for their well-documented traffic simulation software suite.

References

Arel, Itamar, Liu, Cong, Urbanik, T, and Kohls, AG. Reinforcement learning-based multi-agent system for network traffic signal control. *Intelligent Transport Systems, IET*, 4(2):128–135, 2010.

Covell, Michele, Baluja, Shumeet, and Sukthankar, Rahul. Micro-auction-based traffic-light control: Responsive, local decision making. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pp. 558–565. IEEE, 2015.

David Schrank, Bill Eisele, Tim Lomax and Bak, Jim. Urban mobility report. Technical report, Texas A&M, College Station, TX, 2014.

EIA, Mar 2016. URL https://www.eia.gov/tools/faqs/faq.cfm?id=23.

Jayakrishnan, R, Mattingly, Stephen P, and McNally, Michael G. Performance study of scoot traffic control system with non-ideal detectorization: field operational test in the city of anaheim. In *80th Annual Meeting of the Transportation Research Board*, 2001.

Krajzewicz, Daniel, Erdmann, Jakob, Behrisch, Michael, and Bieker, Laura. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5 (3&4):128–138, December 2012.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Salkham, As' ad, Cunningham, Raymond, Garg, Anurag, and Cahill, Vinny. A collaborative reinforcement learning approach to urban traffic control optimization. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*, pp. 560–566. IEEE Computer Society, 2008.

Wunderlich, Richard, Liu, Cuibi, Elhanany, Itamar, and Urbanik, Tom. A novel signal-scheduling algorithm with quality-of-service provisioning for an isolated intersection. *Intelligent Transportation Systems, IEEE Transactions on*, 9(3):536–547, 2008.