Traffic Control With AI

67842 Introduction to Artificial Intelligence - Final Project - August 2022

Yossi Doctor
yossidoctor98@gmail.com

Adi Borochov
adi.borochov@mail.huji.ac.il

Nimrod Alon
bnrustkui@gmail.com

Nadav Alon
nadavalon98@gmail.com

Abstract

Inefficient traffic lights may cause congestion, impacting economic productivity, health, and quality of life. Many approaches can be taken to reduce traffic congestion and lessen its adverse effects on societies in urban areas. In this project, we coordinate traffic signal schedules using Q-Learning and genetic algorithms to improve waiting times. Compared to the default behavior, the trained Q-Learning agent cut down the average waiting time of vehicles by up to 11 percent, and the agent using genetic algorithms cut down the average waiting time by up to 7 percent.

Introduction

Traffic congestion has a non-negligible impact on health. A 2015 study [1] found that the concentration of harmful particles was 29 times higher in places with congested driving conditions compared to areas with free-flowing traffic. A 2010 study [2] by the Harvard Center for Risk Analysis associated fine particles emitted by congested traffic with 3,000 premature deaths in the US in 2007.

Moreover, traffic congestion has a significant economic impact. A 2021 study [3] found that in the US, the average driver lost that year $564 due to congestion. This cost was significantly lower than the pre-COVID cost of $1,374 in 2019. As the world comes out of the pandemic, we can assume these costs will rise again.

Lastly, standing in traffic is not a pleasant experience (citation needed).

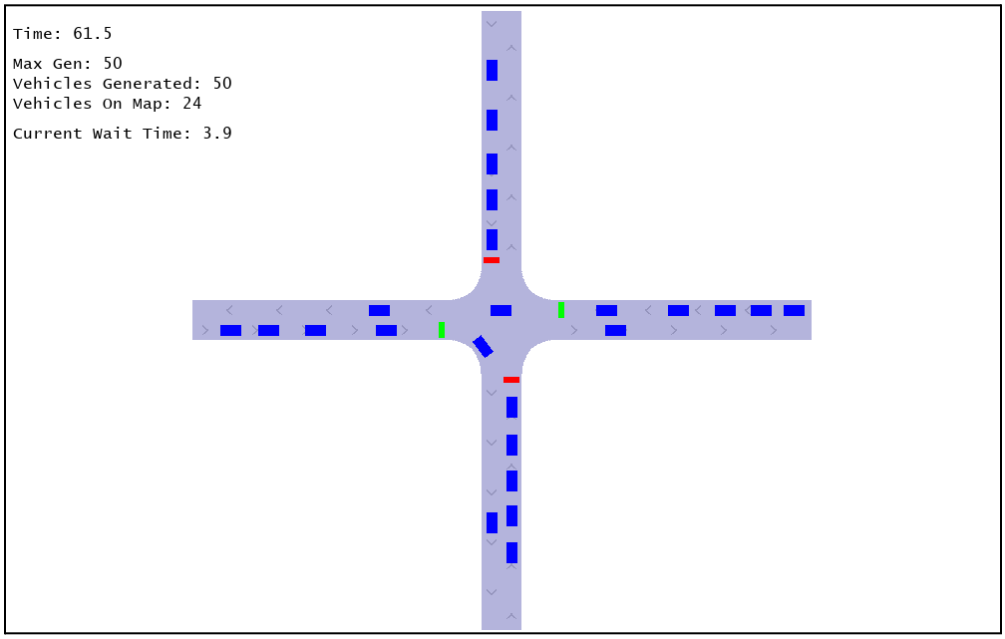
In this project, we build AI agents that can improve the average wait time of vehicles in a two-way junction.

Methodology

Simulation

The simulator is a modified version of a simulator taken from GitHub [4]. The simulator was adjusted based on the project needs, including significant performance improvement and major optimizations, collision detection, data gathering methods, and adaptation for reinforcement learning environments and genetic algorithms.

The setup used in our project is a single-lane two-way junction, with two main phases of green-red light and two intermediate phases of yellow light.



Reinforcement Learning

The main objective of our Q-Learning agent is to reduce the average wait time of the vehicles. Employing an MDP framework, we defined the state space as a tuple of the following parameters: the state of the traffic signal at the junction, the number of vehicles on the north-south route, the number of cars on the east-west highway, and an indicator of whether the junction is free of cars. The capacity of roads in each direction is 14 vehicles (without counting cars inside the junction), so the state space size is 784 ($2 \times 14 \times 14 \times 2$).

The state space we chose is small enough to train on and yet sufficiently expressive to distinguish between different states.
The action space consists of two actions - changing the traffic flow direction or doing nothing.
After each action, the simulation runs for three time units to allow vehicles to progress on their journey.
The reward function is based on the difference between the number of cars on inbound roads after the previous action and the number of cars on inbound roads after the current action. If the number of vehicles goes up, the reward is negative, and if it goes down, the reward is positive.

Genetic Algorithms

Unlike regular graph search algorithms, genetic algorithms allow us to look not only at a single action ahead but rather at a sequence of multiple actions at a time. This is better for our purpose because the cars on the roads and at the junction need time to accelerate to pass the junction once the traffic light switches from red to green, and they need time to slow down once it switches back from green to red.

To decide what chain of actions to do next, the agent takes a snapshot of the simulation and tries to maximize the number of cars passing the junction. It generates a default set of sequences, checking if any of them passed at least 50 percent of the vehicles through at a fast enough rate. If it finds such a sequence, no more evolution is done. Otherwise, using crossover and mutations, it generates sequences until it finds one good enough.

Experimentation

Measuring Default Behavior

Every traffic light signal has either a timer or sensor that directs traffic flow. Timer-based traffic lights are modeled with a “fixed cycle” (FC) behavior - the direction of traffic flow switches in cycles of fixed length. Traffic lights using a combination of a timer and a sensor are modeled with a “longest queue first” (LQF) behavior.

Every cycle iteration of fixed length, the route with most vehicles gets the green light. Evaluating 500 iterations of these methods in cycles of different lengths (in simulation time units), we get the following results:

	Average Waiting Time		Average Collisions Rate	
Cycle Length	FC	LQF	FC	LQF
10	3.81	3.47	0.05	0.05
15	3.40	3.24	0.11	0.08
20	3.59	3.49	0.04	0.02

As expected, because LQF considers the number of vehicles on the road, it gets better results than FC for every cycle length.

Measuring Q-Learning Agent Behavior

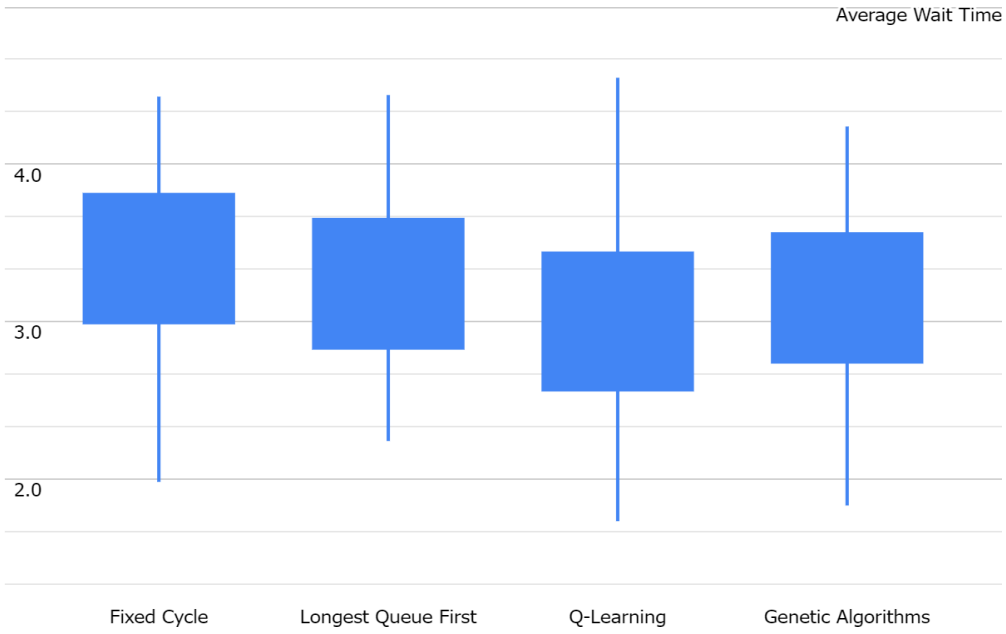
After training the agent for 10 thousand episodes, each containing 50 vehicles, we got the following results: the average wait time is 3.00 simulation time units, and the average collision rate is 0.03. That is a 7.5 percent improvement over the best LQF average wait time result and an 11 percent improvement over the best FC average wait time result. Additionally, the collision rate went down. Interestingly, training for 20 thousand and 50 thousand episodes did not improve the results further - this is because the state space size is 784, and there are two actions, so the amount of state-action pairs is only 1568. Therefore, 10 thousand episodes are enough to get the best results for our chosen reward function.

Measuring Genetic Algorithms Results

When testing the agent for 500 episodes containing 50 vehicles, we got the following results: the average wait time is 3.15 simulation time units, and the average collision rate is 0.03. That is a 3 percent improvement over the best LQF average wait time result and a 7.5 percent improvement over the best FC average wait time result. Here, too, the collision rate went down. We use sequences of 5 actions, which get the best results from all the options we checked. Testing the

agent under the same environment with sequences of 3, 4, and 6 actions gave an average wait time of 3.48, 3.61, and 3.40 respectively.

Conclusion and Discussion



We were able to train AI agents and get a non-trivial improvement in waiting time over the default behavior of traffic lights.

Possible Improvements For Q-Learning

We can tweak several hyper-parameters to change the results - all the Q-Learning parameters (exploration threshold, learning rate, discount rate) and the amount of time the agent waits between actions. Modifying these values might lead to a better learning process. Additionally, we can change the representation of the state space and the reward function. As we have seen, our current state space is compact and allows the training process to be relatively short. However, it stands to reason that making the state space more expressive will lead to better performance, as it can more accurately pick the right action. One such way is recording the vehicles' positions and

velocities, rounded to a certain degree of accuracy to make the state space discrete. The size of such a state space will be in the tens of thousands.
To properly train an agent, it needs to see all the possible combinations of state-action several times, so the number of training episodes required will be in the hundreds of thousands - this requires more computational resources than our state space implementation.
For the reward function, instead of using the change in inbound vehicles, we can use throughput, wait time, vehicle speed, or any combination of those.

Real-World Implementation

It is worth noting that such frameworks are being studied, tested, and implemented in real-world environments. In 2019, the company Alibaba implemented an AI-controlled traffic lights network in Hangzhou, China. Hangzhou once ranked fifth among China's most congested cities, but two years after the deployment of the AI platform has now dropped to 57th on the list [5].
Moreover, a 2022 deep reinforcement learning project in Germany [6] is being trained on traffic data captured from a junction in Lemgo, Germany, using high-resolution cameras and radar sensors. The researchers demonstrated a 10-15 percent improvement in traffic flow in their simulations.
This goes to show that such AI-guided solutions are feasible. Employed in tandem with other approaches like better public transport, city planning, and infrastructure, these solutions can reduce traffic congestion.
Furthermore, as technology advances and new developments in artificial intelligence emerge, better models could be trained, reducing traffic congestion even more.

Appendix: Running the Code

Run main.py with the non-optional flags:

- -m METHOD

  Where METHOD is one of the following options: fc, lqf, qlearning, search.
- -e NUMBER-OF-EPISODES

The flag -r (render) is optional and shows the current simulation.

References

[1] Goel, Anju, and Prashant Kumar. "Characterisation of nanoparticle emissions and exposure at traffic intersections through fast response mobile and sequential measurements."

[2] Levy, Jonathan, et al. "Evaluation of the public health impacts of traffic congestion: a health risk assessment."

[3] INRIX. INRIX 2021 Global Traffic Scorecard

[4] BillHim. github.com/BilHim/trafficSimulator

[5] Toh, Michelle, and Leonie Erasmus. "Alibaba's 'City Brain' is slashing congestion in its hometown."

[6] Fraunhofer-Gesellschaft. "Traffic lights controlled using artificial intelligence."