An Exploration of Tabular Q-learning to Optimize Traffic Lights

Author: Ryan Strotman
GitHub Repository: https://github.com/strotman-ryan/TrafficLightRL

1 Introduction

Traffic is a large problem today, costing many people valuable time every day. Traffic causes lost time and an increase in gas emissions from idling cars and from constant stopping and accelerating. More cars are on the road everyday causing traffic to grow and a solution to traffic even more important. Traffic lights or signals have a great effect on traffic and therefore the issue of traffic can be greatly reduced if traffic lights work to the best of their ability. However, most traffic lights today do not. Most traffic lights used fixed time schedules to operate the lights. These fixed plans work well under constant traffic that does not fluctuate. Therefore, these plans do not work well under real traffic conditions that are constantly changing. Reinforcement learning has been used to try to solve this problem of traffic light optimization. Reinforcement learning can adjust to the fluctuations in traffic and handle many different traffic patterns. Many types of reinforcement learning agents have been studied in this area with Deep-Q networks having the most success. In this paper, tabular Q-learning will be used to solve this traffic light problem. The hyper-parameters for this algorithm will be experimented with and analyzed. Also, state aggregation will be used to improve upon the raw tabular version.

2 Problem Definition

I will run tests on a simulated intersection. The intersection will be 4-ways with four 1,000 meter roads meeting at a right angle. Each road will have one outgoing and one incoming lane. Each road comes from a different direction; North(N), South (S), West (W), and East (E). The cars entering the intersection must continue in the same direction. For example, a car entering upon the E lane must exist to the W lane. No turning is allowed in these tests. There will be two light phases; denoted NS and EW. NS means the N and S roads have a green light or yellow light while the E and W lanes have a red light, and vice versa for EW. After each green light there is a yellow light for 3 seconds then the phase switches to the next phase. Each phase has a minimum green time of 1 second.

2.1 RL Environment

The Reinforcement learning environment is a description of the state, reward, and action space.

2.1.1 State Space

The environment of this scenario is all the area surrounding the intersection. The state will be representative of this area. The state will be the current phase (either EW or NS) and for each lane the number of cars on it within 150 meters on the intersection. This state space is the same as the space used by [1][2]. The space is very simple and therefore it will be easier to learn an optimal policy [2], especially when only using tabular methods.

2.1.2 Action Space

The actions the agent chooses will be to either stay at the current phase or to go to the next one.

2.1.3 Reward Space

The reward will be the opposite of the sum of all queued vehicles on all lanes. A queued vehicle is a vehicle that has a speed less than or equal to .1 m/s. This was chosen as the reward since it was proven that minimizing the number of queued vehicles is equivalent to minimizing the average travel time of all vehicles [2]. This reward is very simple. Most rewards in this area of research are a weighted sum of a number of different values and changing any of these weights can change the performance drastically [2]. These types of rewards do not perform as well as the I will use [2].

2.1.4 Q-learning

Q-learning is an algorithm to find the best policy; what action to preform for each given state. Q-learning uses the above state, action and reward to solve this bellman equation:

$$Q(S_t,A) = Q(S_t,A) + alpha(R_{t+1} + discount * max_A(Q(S_{t+1},A)) - Q(S_t,A))$$

Q-Learning is very popular for this problem since it learns online. The algorithm updates a state-action pair only one step after that state-action pair was seen. This is very important in continuous tasks since we cannot wait for the end result to update a value like in Monte Carlo methods. The agent will take actions according to an E-greedy policy. This means a random action will be taken E percent of the time instead of the current max action.

3 Methods

The above state, action, and reward space will be used along with the controlling Q-learing algorithm. The agent will receive the current state and reward each time step and the agent will make an action. Each time step is one second long. If the state of the intersection is in a yellow light state the action of the agent is not used since the light is already changing.

4 Results

The simulations were run using an open source software called SUMO(http://sumo.sourceforge.net/). This is a popular tool for simulating traffic and intersections. Simulations were run using 2 different traffic patterns, denoted in table 1 (all tables are in the Appendix). WE means both cars going in the West and East direction and similarly for NS. The first configuration has constant flow in both directions. The second configuration has an abrupt switch in the middle. These configurations were inspired by Hua Wei et al [1].

4.1 Evaluation

The results of simulations will be evaluated based on the average travel time of all vehicles in the simulation. The lower the better. This value will be compared to the value gotten from the fixed time algorithm. This algorithm has a fixed time for each signal phase and does not react or change with the environment. This is the technique most seen today in the real world.

4.2 Analysis of Changing Values for Configuration 1

The algorithm was run on the first configurations with differing discount values. The results are in table 2. The alpha and epsilon values are kept constant. The Q-learning algorithm was able to do better than the fixed time algorithm. The discount value determines if the agent should try to maximize short term of long term reward. A value of 0 will try to maximize the immediate reward and does not work well. An intermediate value is optimal.

I also tried different epsilon values. Epsilon controls the exploitation to exploration ratio. If the epsilon value is low the agent is more likely to choose the action that has the highest expected reward. While a high epsilon value will explore actions that are not the best but may help the agent find a better action in the long run. In table 2, the best value is one that favors exploitation with a little bit of exploration. A value of 1 is equivalent to a random agent. This agent did not perform well.

The learning rate was also varied. The results are shown in table 4. The learning rate defines how much weight the most recent state-action visit effects the value for that state-action pair. As seen in table 4 an intermediary learning rate produces the best results. A learning rate of 0, means the values never change, did not preform well.

4.3 Analysis of State Aggregation Techniques for Configuration 2

Configuration 2 has heavy flow in one direction and none in the other then the traffic patterns swap. This heavy flow causes the state space to be much larger than the state space in configuration 1. The number of cars in a lane at a time in configuration 1 was about 2 cars, while the number of cars during configuration 2 was much higher around 11. With a much larger state space the agent does not visit each state enough times to learn its true value. One way to deal with this is through state aggregation.

State Aggregation is grouping similar states together and treating them all the same as one state. This shrinks the state space and the agent can learn how to react to a state that it has never seen before. I tried a couple of difference state aggregation techniques for configuration 2 the results are in table 4. With no state aggregation the agent preforms very poorly. A log function was used to group the number of cars in a lane together. The function was $\lfloor \log_2(\#\text{ vehicles in lane } +1) \rfloor$ for each lane. This way the agent treated two or three cars in a lane as the same state. This greatly helped performance. Another technique was the min function. I calculated the state using minimum(1, # vehicles in lane) for each lane. This shrunk the state space into a binary encoding plus the phase at the time. This greatly reduced state space caused the agent to be able to learn the policy faster and perform better. This minimum function reduced the granularity of information passed to the agent, but for this scenario it contains all the information needed since traffic only went in one direction at a time. I do not expect this to work well in all scenarios.

5 Conclusion

A number of simulations were run on a 4-way intersection and the results were analyzed. Q-learning was the chosen algorithm to find the optimal solution because of its ability to learn online. Three hyper parameters: the learning rate, the discount factor, and epsilon, were varied. It was shown that an intermediary of these parameters is best and each plays a role in making Q-learning successful. State aggregation helped tremendously when the traffic rate was increased. In the future I would like to explore Deep Q-Learning so highly dimensional state spaces do not slow learning. This will allow me to explore larger intersection.

6 References

[1] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. 2018. IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220096

[2] Zheng, G.; Zang, X.; Xu, N.; Wei, H.; Yu, Z.; Gayah, V.; Xu, K.; and Li, Z. 2019b. Diagnosing reinforcement learning for traffic signal control. arXiv preprint arXiv:1905.04716.

7 Appendix

Table 1: Traffic Patterns

Configuration	Direction	Rate (cars/lane/s)	Start Time (s)	End Time (s)
1	WE	.011	0	7200
1	NS	.011	0	7200
2	WE	.133	0	3600
2	NS	.133	3601	7200

Table 2: Exploration of Discount values for Configuration 1

Alpha and epsilon values are .1 .1

Method	Discount	Avg Travel Time (s)
Fixed Time	NA	161.64
Q-learning	0	162.19
Q-learning	**.5**	**159.91**
Q-learning	.75	160.00
Q-learning	.8	160.01
Q-learning	.9	160.23
Q-learning	.95	160.21
Q-learning	1	160.40

Table 3: Exploration of epsilon values for Configuration 1

Alpha and discount values are .1 and .5 respectfully

Method	Epsilon value	Avg Travel Time(s)
Fixed Time	NA	161.64
Q-learning	1	162.08
Q-learning	.5	161.46
Q-learning	**.1**	**159.91**
Q-learning	.05	161.38
Q-learning	.01	160.18
Q-learning	0	160.6

Table 4: Exploration of the learning rate for configuration 1

Epsilon and discount factor are constant at .1 and .5 respectfully

Method	Learning rate	Avg Travel Time(s)
Fixed Time	NA	161.64
Q-learning	1	161.49
Q-learning	.5	160.63
Q-learning	**.1**	**159.91**
Q-learning	.05	160.57
Q-learning	.01	160.39
Q-learning	0	162.08

Table 5: Exploration of State Aggregation for Configuration 2

All instances have their hyper parameters tuned for near optimal performance

Method	State Aggregation Technique	Avg Travel Time (s)
Fixed Time	NA	179.74
Q-learning	No state aggregation	339.61
Q-learning	Log function	176.68
Q-learning	**Min function**	**171.41**