Optimal Parking Problem

Soham Choudhury
Department of Computer Science
San Jose State University
San Jose, CA, U.S.A.
soham.choudhury@sjsu.edu

***Abstract*— Parking is a struggle for many drivers around the world. Although it may seem simple, parking accurately in between the allotted lines is something that licensed drivers fail to do at every parking lot across the country. The people who pay for this inadequacy are the other drivers who are no longer able to park in the spaces adjacent to the one that the previous driver parked in because those spaces have other cars overlapping the lines. Another reason this problem should be solved is because in the process of parking into a space, human drivers oftentimes hit the car in the spot next to them. We model a parking lot environment to adequately simulate a car getting to its ideal spot by utilizing Reinforcement Learning algorithms.**

Keywords—Reinforcement Learning, OpenAI, Proximal Policy Optimization, Deep Q Network

I. Introduction

To model the environment, a design needed to first be put in place. The parking lot would be representative of a birds-eye view of any regular, single-floor parking lot. There would be two rows of parking spaces, two aisleways on the outside of both rows and another going in between them. To properly emulate the objective of parking in a parking lot, a store entrance is also placed at one end of the lot, opposite to the side that the car starts at. Since complicated behavior of the car cannot be represented in the resources available to the software that will be used, the car is limited to the four cardinal directions: north, south, east and west. Further details of the environment are explained in later sections.

II. Methodology: Setting up Gridworld

A. Environment Components

The dimensions of the Gridworld implementation that will be used are 7 by 18. The width accounts for the three aisleways and the two parking rows, each with a width of two.

The car itself only takes up a 1 by 1 grid. There are ten total parking spots, of which seven will be preoccupied by another car. The other three spots will be treated as targets for our car to get to. On three of its four sides, each parking spot is surrounded by parking lines. The motivation behind this is to create an obstacle for our car to avoid. Although hitting another car is a episode-ending offense, simply going over parking lines will just be penalized. The store entrance is also represented by a 1 by 1 grid, however, it will not be a physical obstacle for the car to avoid. Instead, it is just a placeholder for the car to realize how far away from the entrance it is.

B. Randomization of Component Locations

As mentioned earlier, only three of the ten total parking spaces are available for the car to park in. However, this will not be an interesting problem if these three spots' locations were constant. After all, in any packed parking lot, the locations of the open spaces are not guaranteed to be at the same place at different times of entering the lot. Hence, every time the environment is reset after the termination of the previous one, the open and, subsequently, occupied parking spaces are randomized. The locations of all parking spaces as a whole are still constant.

The location of the store entrance is also randomly selected to be at any lateral position on the furthest row of the parking lot. In other words the domain of its x-coordinate is {0,6}. The motivation behind this decision is to reflect how the entrance of stores in real life are not always at the center of one end of the parking lot. Sometimes, it is off-center and our agent should learn to cope with this imperfection.

All other components of the environment stay constant across episodes during learning. Our car starts at the middle of one end of the parking lot and the parking lines never change position or orientation.

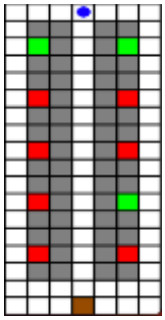

Figure 1: Gridworld environment

III. Methodology: Creatiing the Environment

This environment follows OpenAI's Gym documentation. Every environment needs init, step, and reset functions. The step function takes as input the action the agent takes and returns the resulting state of the environment and a reward for that experience. This is also coupled with a flag which determines if the current episode is terminated or not.

A. Observation Space

This is essentially a Python dictionary which contains four items: agent location, store entrance location, target locations, and parked car locations.

Both the agent location and store entrance location have a dimensionality of 1 by 2 because each of them are represented by a single pair of x and y coordinates. Since there are multiple targets (i.e. empty spots) and parked car locations, they have dimensionalities of 3 by 2 and 7 by 2, respectively.

The parking lines do not have to be recorded as a component of the observation space because they are constant obstacles in the environment. In other words, the agent will learn to avoid these obstacles' locations as it progresses in learning.

B. Action Space

As mentioned in the introduction, the car has only four options at every timestep: move up, down, left, or right. Obviously, this means that the car cannot move diagonally in one step. Additionally, if the car is at an edge or corner and it takes an action which would put it outside of the boundaries of the Gridworld environment, then the car remains at its previous position without accruing any penalty in that experience. This functionality is achieved by the numpy library's clip function.

If an action leads to the agent being placed on a parking spot, preoccupied or free, then the episode terminates there. Otherwise, the episode continues.

C. Rewards

Due to there being a store entrance, the three parking spots has an inherent ranking in regard to which one a driver may want to end up parking in. Intuitively, a driver will want to park as close to the store entrance as possible in order to minimize the distance they have to walk. Hence, the closest open spot to the entrance should have a higher reward than the other open spots if the agent reaches it.

We want to harshly penalize the agent for hitting another car and lightly penalize it if it goes over parking lines. Otherwise, if the agent simply goes over whitespace, then there is no point in handing it a negative reward. However, we also do not want the agent to meaninglessly travel around the parking lot and avoid parking at an open spot. To appease both sides of this argument, the reward for progressing onto an empty grid space was simply 1.

Result from Action	**Reward**
Getting to best parking space	100
Getting to 2nd best parking space	50
Getting to 3rd best parking space	30
Hitting another car	-30
Going over line	-5
Otherwise	1

IV. Results

The two algorithms used to train the agent in this environment are both developed by OpenAI. They are Proximal Policy Optimization (PPO) and Deep Q Network (DQN).

PPO [1] aims to take the biggest possible improvement step on a policy using the data that it currently has, but without stepping so far that it accidentally causes a performance collapse. It

supports multiple stochastic gradient descent (SGD) passes over the same batch of experiences.

Figure 2: PPO architecture

DQN [2] approximates a state-value function in a Q-Learning framework with a neural network. Like PPO, DQN has an implementation to store experiences in memory, often referred to as a replay buffer. It uses its learning to update a target network, which is updated every *n* steps (where *n* is a hyperparameter).

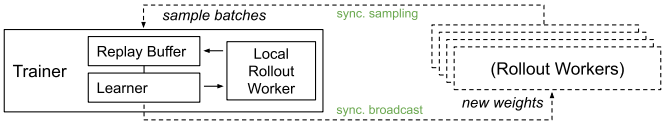

Figure 3: DQN architecture

A. Figures

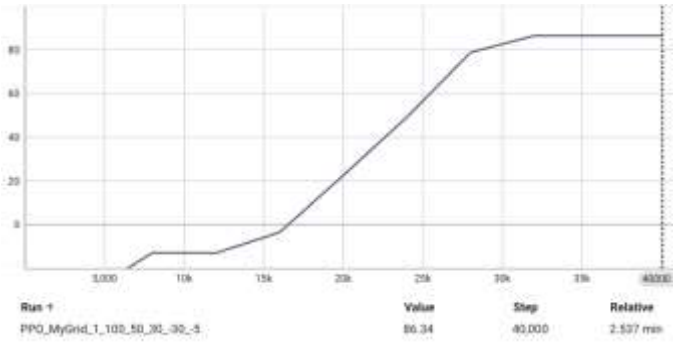

Figure 4: PPO, Mean episode return

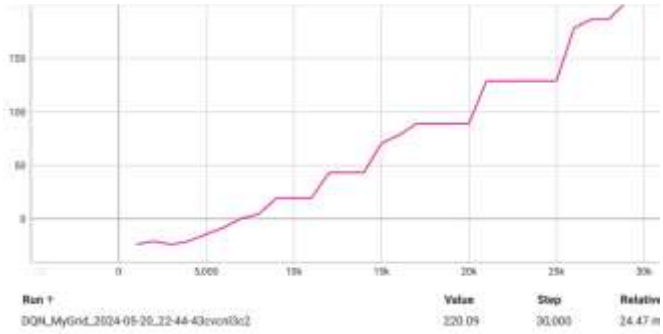

Figure 5: DQN, Mean episode return

V. Discussion

We see that both algorithms train its agents to avoid all preoccupied spots early on during the learning process (before the 5,000th timestep). However, we look a bit more closely at the differences in their outcomes.

PPO successfully learns where the optimal parking spot is. We can see that it stabilizes learning after around 33,000 timesteps. However, it does not learn the optimal path. We can confirm this by acknowledging that the mean return from an episode never goes above 100, which the optimal path should achieve. This is because on the way to the best parking spot, the optimal policy of an agent should avoid all the parking lines so that it does not incur any unnecessary penalties. However, since the mean reward peaks at 86.34, the agent crosses a few parking lines on its way to the optimal parking spot.

Although it takes much longer to train, DQN was able to train the agent to avoid hitting the parking lines. Although it achieved a final mean return of 220, the agent spent a lot of time aimlessly going around the whitespaces of the parking lot. Nevertheless, the frozen target network of the DQN can be seen from the results above; the sharp increments in the mean return can be seen every 2,000-3,000 timesteps starting approximately from timestep 7,000.

VI. Conclusion

To retain the optimal path from the agent's starting position to the best parking spot, there has to be better selection of either the rewards or the algorithm used for training the agent in this environment. Due to the limited scope of the resources of this project, including time and knowledge, pinpointing the best algorithm was deemed to be infeasible. Choosing the best rewards for this environment in order to optimally facilitate the agent's learning is a task in itself, despite how trivial it may seem.

Training this agent on parking lots of various dimensionalities would be helpful and practical for itself to perform well in the real world. This can go hand in hand with the advancements in computer vision; a complicated real-life parking lot can be reduced to a Gridworld representation. Then, simply place the agent in the latter and train it.

With the normalization of smart cars in the U.S., it is not a far-fetched idea that stores can give customers an optimized policy for parking in their parking lot. This can happen when a customer enters the lot in their car. The car first accepts the

policy to enter their system. Then, the policy does its work by hands-free driving the car to the best parking space in the lot. Of course, further research in the field of Multi-Agent Reinforcement Learning (MARL) is needed to bring this idea into fruition.

REFERENCES

[1] P. Dhariwal, O. Klimov, A. Radford, J. Schulman, and F. Wolski, "Proximal Policy Optimization" OpenAI, July 2017

[2] J. Schulman and S. Sidor, "OpenAI Baselines: DQN" OpenAI, May 2017