On Demand Transportation Scheduling

Progress Report

Weini Yu `weiniyu`

Zhouchangwan Yu `zyu21`

Yutian Li `yutian`

November 17, 2017

1 Introduction

Our project aims to develop an intelligent model that provides a solution to the single vehicle pickup and delivery problem (SVPDP). For now we are only going to look at the pickup and delivery problem of a single vehicle and without considering carpool situation.

2 Model

We have approached this problem using an MDP model, with the intuition that at a certain location (state), the driver could choose to take one of the requests that come in (action). For each action the driver receives a reward. At the end of the day, the driver has a total reward for all the rides he or she takes. And through solving this MDP, the driver can figure out which request to take at a location in order to maximize the total reward.

2.1 State

We originally had states that include the driver's current location and a list of requests at the location. However, this makes the state space very large (10^5) which is not desired. Also the relative order of requests does not really matter. So we have simplified it such that it only contains the driver's current location, a zone index number, reducing the state space to 10. For example, `state=1` means that the driver is at location 1. Note that we are not losing anything by not including requests. Requests are encoded into actions.

2.2 Action

Since state is the current location the driver is at, the action would be the possible request the driver receives, which is a tuple `(source, destination)` where source is the pick up zone index and destination is the drop off zone index. In our specific model, the driver is given 5 requests to choose from at the start of the game and the end of each ride. So at any step, there are 5 legal actions. And the driver must choose one from them.

The requests are generated in the following way. The source location is randomly sampled with lower location indices having a higher weight. The destination location is sampled uniformly randomly. By generating requests in a nonuniform way, we favor locations that have smaller indices and create an imbalance between states. We hope that the agent will figure out some locations are inherently better, and use this knowledge to achieve higher rewards.

2.3 Reward

The reward of each action or ride will be (fare $- \alpha \cdot$ travel time). The intuition here is that gas cost needs to be subtracted and the longer the travel time is the less happy the customer is so there is the negative impact on the reward. The constant α balances fare and travel time, and could be seen as a measure of productivity. In our specific model, α is chosen to be $\frac{1}{150}$.

3 Algorithm

3.1 Baseline and Oracle

Based on this model, we implemented baseline as a driver that randomly picks one request at each step. And the oracle is a driver that knows the true reward function for all the requests, and picks greedily according to it.

3.2 Q-learning

For our agent, we used Q-learning as a first step to solve this problem. For a smaller sized problem we could run exhaustive search, but it is clearly not scalable. We chose Q-learning instead of TD learning, because intuitively states do not matter that much in our model. Though some states are inherently better due to the request generation probability, actions are much more important. Besides, if a driver takes a request and returns to the same location, the state is same but clearly the total reward is not.

For function approximation, we used linear regression to predict the Q value. As a first step, we manually picked a few features.

3.2.1 Features

Distance: The travel distance is one of the main concern for the drivers. Here we define two features related to the travel distance, the distance from current location to source, and the distance from source to destination.

Zone: Some zones are more attractive to people than others, so the drives may have preference on where they would like to go. Each zone as current location, source, or destination are all considered as features.

We define indicator features $\mathbf{1}\{\text{current zone} = z\}$, $\mathbf{1}\{\text{source zone} = z\}$, and $\mathbf{1}\{\text{destination zone} = z\}$ for all z's.

3.2.2 Training

Using features described above, we learned the respective weights using gradient descent. We updated weights using exponential weighted moving average as $w_i \leftarrow w_i - \eta(Q(s,a) - r)\phi_i(s,a)$, where $\eta = 0.1$. Note that we set discount $\gamma = 0$. This is because there is no way for the driver to know the legal actions (requests) available at future states. They are randomly generated only when the driver gets to a new state. $\max_a Q(s,a)$ is hard to define, hence $\gamma = 0$ for simplicity.

4 Data Preprocessing

At the end of data preprocessing stage, we generate a `city.p` pickle file that has everything we need about the environment. It has a list of locations, travel times between locations, fare estimates, and coordinates of the locations.

Locations (a.k.a. zone indices): We downloaded from Uber Movement travel times in Washington D.C. during second quarter of year 2017. There are 558 zones in total, and we picked the first 10 zones for downsampling.

Travel times: We picked the arithmetic mean travel time as the travel time between zones. The distribution of travel times is shown below.

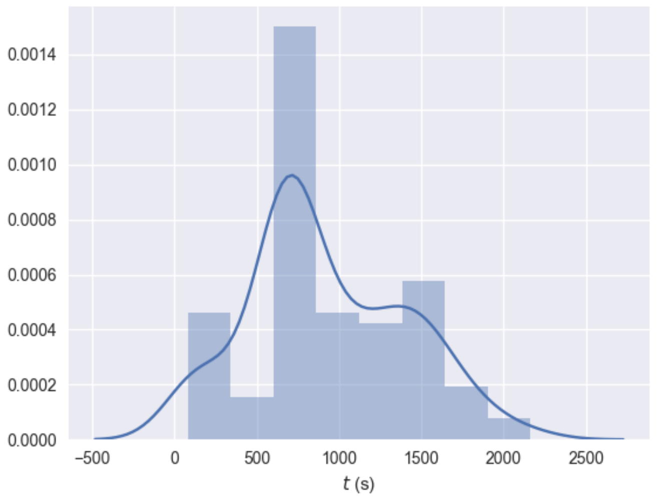

Figure 1: Distribution of traveling time.

Coordinates: We calculated the arithmetic mean longitude and latitude as coordinates of the locations. **Fare**: The fares between these 10 locations are requested form Uber Developers API. Given the longitudes and latitudes of any two locations, the low estimate and high estimate of the fare are generated for various types of vehicles. We use the average of the fare for UberX and keep the data in a 10×10 matrix. The distribution of fare estimates are shown below.

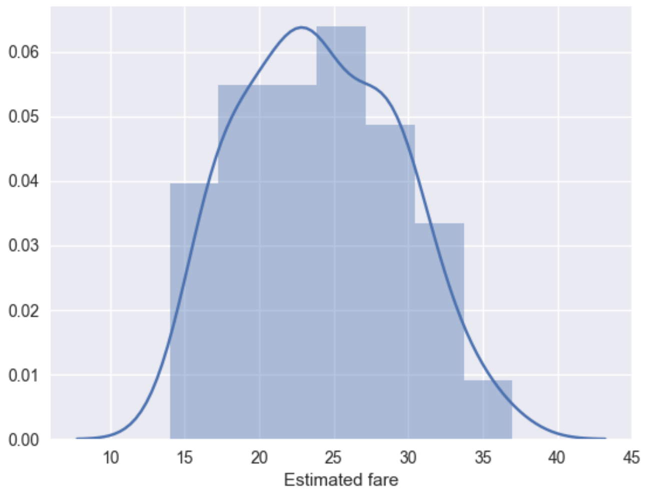

Figure 2: Distribution of estimated fare.

5 Example

For a concrete example, we assume the driver is at location s. Five requests (s_i, d_i) for $1 \le i \le 5$ are dispatched to the driver. The driver evaluates $Q(s, (s_i, d_i))$ for all five requests using linear estimator $\sum_i w_i \phi_i(s, a)$ and picks the one with the highest estimation. The driver takes this action. Then the reward r is given to the driver, who in turn uses $w_i \leftarrow w_i - \eta(Q(s, a) - r)\phi_i(s, a)$ to perform gradient descent to update the weights w.

6 Results

We trained our algorithm and compared to baseline and oracle. The resulting reward curve is shown below.

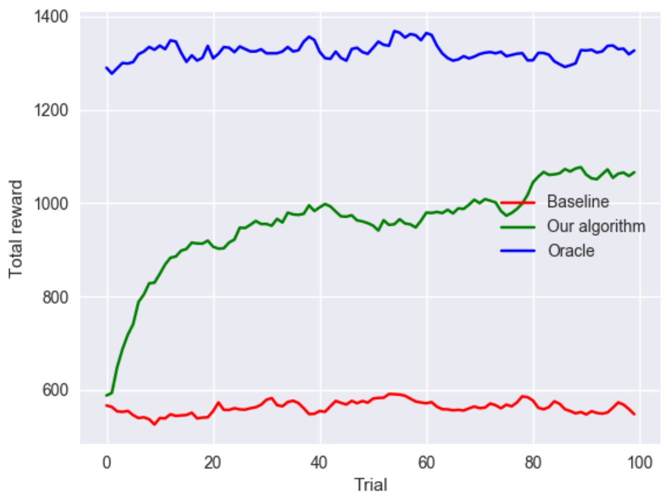

Figure 3: Reward curve.

At first our algorithm is acting randomly because all the weights are just initialized. Then quickly it ramps up and learns the Q function. At iteration 100, it is already receiving twice the reward of baseline, but still about 30% worse than the oracle.

7 Next Steps

We will first implement our Q-learning algorithm on a larger sample size (100 zone locations) to see how it performs as the problem is getting more complicated. We will also considering more advanced learning algorithm. We could try to use a neural network instead of hand craft linear features.