ORIE6590 Final Report

Wangwei Wu, Yucheng Chen

May 2021

1 Introduction

Nowadays, ride-sharing platforms are playing more and more important roles in transportation all around the world. Compared to the traditional ride-hailing service providers, like taxi companies, those emerging ride-sharing platforms, such as Uber, Lyft, Didi Chuxing, and etc., have richer information collected from passengers and drivers. The huge amount of data enables the central planner of a platform to more precisely balance trip requests and idle drivers so as to create higher trip matching efficiency and gross merchandise revenue (GMV). However, the large-scale prompt-up dispatching and routing problem brings computational challenges to the central planner. Recently, Reinforcement learning (RL) shows its strength in tackling the challenges in the ride-sharing platform use cases and there are several walks of articles in this avenue illustrating their proposed algorithms in their corresponding settings.

In [1], the authors considered a Markov decision process (MDP) model of a ride-hailing service system, framing it as a reinforcement learning (RL) problem. The simultaneous control of many agents (cars) presents a challenge for the MDP optimization because the action space grows exponentially with the number of cars. They proposed a special decomposition for the MDP actions by sequentially assigning tasks to the drivers. This project implemented the solutions proposed in this paper and compared the results claimed in the paper and the results of our implemented solutions.

2 Jointly dispatching with empty-routing problem

In this section we describe our model of the ride-hailing service and transportation network, following [1]. The service consists of a centralized planner, passengers requesting rides, and a fixed number of geographically distributed agents (cars). In the following, we will briefly introduce the problem settings, the proposed policy proximal optimization (PPO) algorithm in [1], and several numerical experiments with different PPO schemes.

2.1 Problem definition

The transportation network consists of N cars distributed across a service territory divided into R regions. For ease of exposition, we assume that each working day ("episode") of the ride-hailing service starts at the same time and lasts for H minutes. We assume that the number of passenger arrivals at region o in the t-th minute (i.e., t minutes elapsed since the start of the working day) is a Poisson random variable with mean $\lambda_o(t), \forall o = 1, \ldots, R, t = 1, \ldots, H$. Upon arrival at region o, a passenger travels to region d with probability that depends on time t, origin region o, and destination region d: $P_{od}(t)$, $o, d = 1, \ldots, R, t = 1, \ldots, H$. After a trip from region o to d has been initiated, its duration is deterministic and equals to $\tau_{od}(t)$, $o, d = 1, \ldots, R, t = 1, \ldots, H$. In addition, patience time denotes a new passenger's maximum waiting time for a car. We assume that each passenger has a deterministic patience time and we fix it as equal to L minutes. We assume that the centralized planner knows the patience time.

In real time, the centralized planner receives ride requests, observes the location and activity of each car in the system, and considers three types of tasks for the available cars: (1) car-passenger matching, (2) empty-car routing, and (3) "do nothing" (a special type of empty-car routing). We assume that each passenger requires an immediate response to a request within the first decision epoch. If the centralized planner assigns a matching between a passenger and an available car, we assume the passenger has to accept the matching. A passenger who is not matched with a car in the first decision epoch leaves the system.

The state space S^{Σ} of the MDP includes states $s_t = [s_t^e, s_t^c, s_t^p]$, such that each state consists of three components: current epoch $s^t := t$, cars status s_t^c , and passengers status s_t^p.

At each epoch t, the centralized planner observes the system state st, and makes a decision at that should address all I_t available cars, where $I_t := \sum_{o=1}^{R} \sum_{\eta=0}^{L} s_t^c(o, \eta)$ and $s_t^c(o, \eta)$ is the number of cars in the system whose final destination region is o, and the total remaining travel time ("distance") to the destination is equal to η.

We let A^{Σ} denote the action space of the MDP. We propose to decompose every decision $a_t \in A^{\Sigma}$ into a sequence of "atomic actions", each addressing a single available car, to overcome the challenge of the large action space. We let A denote the atomic action space. We note that $A = \{(o, d)\}_{o,d=1}^{R}$.

We call the sequential generation of atomic actions a "sequential decision making process" (SDM process). We let $s_{t,i}$ denote a state of the SDM process after $i1$ steps, for each decision epoch $t = 1, \ldots, H$. Given an action (o, d), the agent will assign the car closet to origin o to a passenger trip from o to d among the cars heading to o within L time steps. If there is no passenger trip from o to d, then we let a car idling at o do empty-routing to d or let a car heading to o do nothing. Any passenger trips not fulfilled with I_t sequential decisions will disappear.

We consider the fulfill rates as the total reward for this finite horizon problem. That is, each matched passenger trip brings the system reward 1, otherwise, the reward is 0.

2.2 Algorithms

In the following numerical results, we tried the PPO algorithm proposed in [1]:

Algorithm 1 PPO

Result: policy π_{θ_J}

1: Initialize policy function 0 and value function approximator $V_{\psi_{-1}}$
2: **for** $j = 1, \cdots, J$ **do**
3:
Run policy $\pi_{\theta_{j-1}}$ for K episodes and collect dataset (1)
4: Construct Monte-Carlo estimates of the value function $V_{\theta_{j-1}}$ following (2)
5: Update function approximator V_ψ minimizing (3)
6: Estimate advantage functions $\hat{A}(s, a)$ by (5)
7: Maximize surrogate objective function (4) w.r.t. θ and update θ_j
8:

$$D_\xi^{(K)} := \{((s_{t,1,k}, a_{t,1,k}, \hat{A}(s_{t,1,k}, a_{t,1,k})), ..., (s_{t,I_{t,k},k}, a_{t,I_{t,k},k}, \hat{A}(s_{t,I_{t,k},k}, a_{t,I_{t,k},k})))_{t=1}^{H}\}_{k=1}^{K} \tag{1}$$

$$\hat{V}_{t,i,k} = \sum_{j=i}^{I_{t,k}} c(s_{t,j,k}, a_{t,j,k}) + \sum_{l=t+1}^{H} \sum_{j=1}^{I_{l,k}} c(s_{l,j,k}, a_{l,j,k}) \tag{2}$$

$$\sum_{k=1}^{K} \sum_{t=1}^{H} \sum_{i=1}^{I_{t,k}} \left\| V_\psi(s_{t,i,k}) - \hat{V}_{t,i,k} \right\|^2 \tag{3}$$

$$\begin{aligned} \hat{L}(\theta, \xi, D_\xi^{(K)}) := \frac{1}{K} \sum_{k=1}^{K} [\sum_{t=1}^{H} \sum_{i=1}^{I_{t,k}} \min(r_{\theta,\xi}(s_{t,i,k}, a_{t,i,k}) \hat{A}_\xi(s_{t,i,k}, a_{t,i,k}), \\ \text{clip}(r_{\theta,\xi}(s_{t,i,k}, a_{t,i,k}), 1-\epsilon, 1+\epsilon) \hat{A}_\xi(s_{t,i,k}, a_{t,i,k}))] \end{aligned} \tag{4}$$

$$\hat{A}(s_{t,i,k}, a_{t,i,k}) = \begin{cases} c(s_{t,i,k}, a_{t,i,k}) + V_\psi(s_{t,i+1,k}) - V_\psi(s_{t,i,k}) & \text{if } i \neq I_{t,k} \\ c(s_{t,i,k}, a_{t,i,k}) + V_\psi(s_{t+1,1,k}) - V_\psi(s_{t,i,k}) & \text{otherwise} \end{cases} \tag{5}$$

2.3 Numerical results

We implemented three numerical experiments with different hyperparameter values and different versions of algorithms. Firstly, we will illustrates the results of recreating experiments done by [1], we adopted the same learning architecture and hyperparameters.

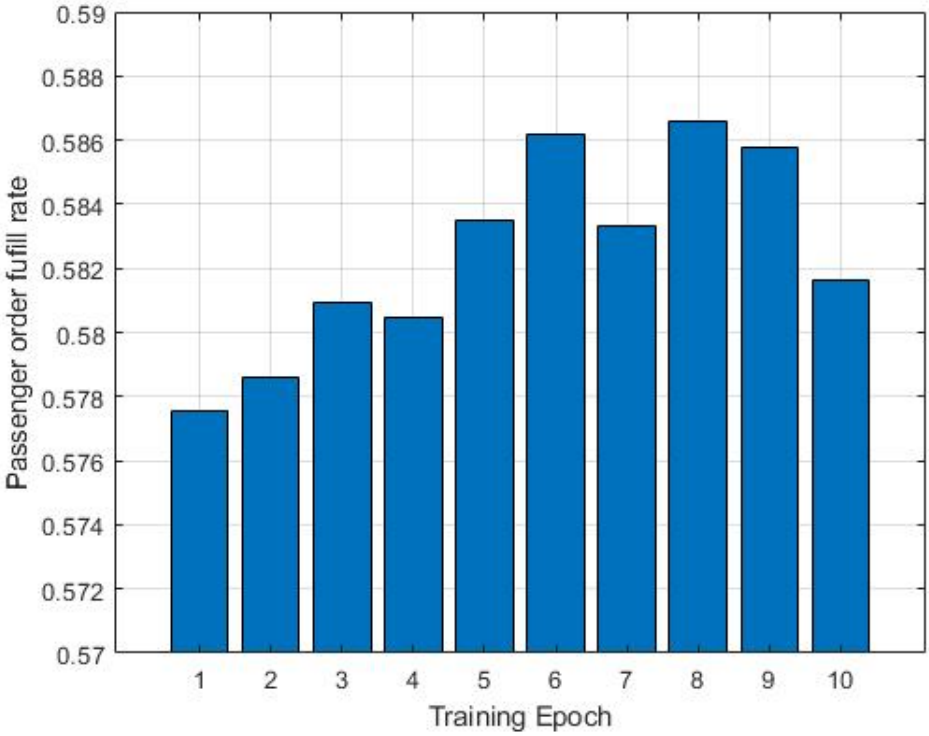

Figure 1: Fulfill rate with empty routing (sln 1)

This solution (sln1) as shown in Fig. 1 is implemented exactly as described in the paper [1]. The performance is not as good as demonstrated in the paper. Particularly, the passenger order fulfill rate doesn't increase significantly as the training goes. The problems we encountered include: 1). the training process was very slow: we simulated the data from 15 20 days for training and the time for one epoch training took up to 1 hour and a half; 2). our computer memory can only store simulated data up to 30 days, and we suspected that not using enough data to train the networks might be one of the reasons that causes this outcome; 3). it was too time consuming to tuning the hyperparameters.

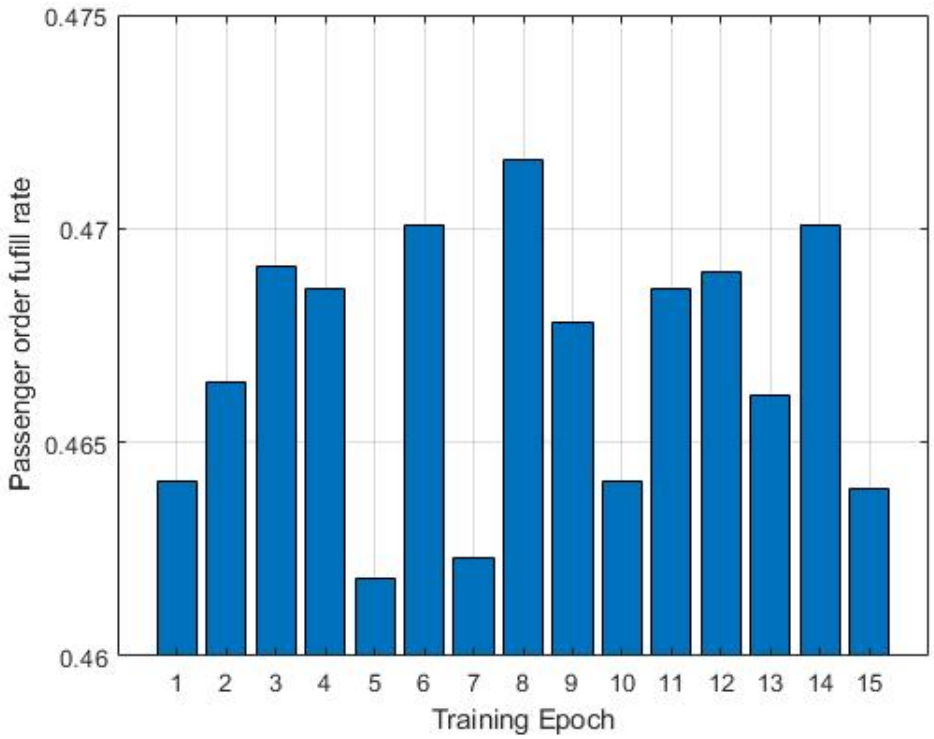

Figure 2: Fulfill rate with empty routing (sln 2)

This solution (sln2) as shown in Fig. 2 is implemented exactly as described in the paper [1] and the parameters are exactly the same as those in the paper, except that we were not able to store the training data for 300 days. The performance is not as good as demonstrated in the paper as well.

3 Problem without the complexity of empty-routing actions

In this section, we consider a similar problems setting without the added complexity in action space about empty routing. For each atomic action defined in the previous section, "empty-routing" is not allowed. Particularly, For each feasible car heading to or idling at o with an assigned action (o, d), it will be matched with a passenger trip if exists or "do nothing" if no passengers left.

3.1 Numerical results

In this setting, we still used the same PPO algorithm 1. Even though we don't have enough time to present the numerical results in this report, we will try to present those in our presentation and later updated versions.

References

[1] Jiekun Feng, Mark O. Gluzman, and J. Dai. Scalable deep reinforcement learning for ride-hailing. *ArXiv*, abs/2009.14679, 2020.