On Demand Transportation Scheduling

Weini Yu weiniyu
Zhouchangwan Yu zyu21
Yutian Li yutian

October 27, 2017

1 Introduction

As the mobile and telecom technologies advance, there has emerged many on demand service platforms. One of them is on demand transportation service platforms like Uber and Lyft that match riders (demand) with drivers (supply). This type of model enhances efficiency and adds job liquidity to the labor market and has a great positive impact on the economy as a whole.

One fundamental problem these platforms and services all need to solve is how to effectively match the customer requests with the vehicles, given the distance, travel time, request density within a range and many more factors. Our project aims to develop an intelligent model that provides a solution to a simplified version of the this scheduling problem, the single vehicle pickup and delivery problem (SVPDP)[1].

2 Related works

Artificial intelligence has been widely applied to on demand transportation problems. For example, a market formation algorithm was developed based on machine learning for Liftago, a mobile app that connects customers and taxi driver by taxi request, matchmaking, accept & bid, and driver selection[2]. With the proposed SImple Data-driven MArket Formation (SIDMAF) algorithm, the authors improve the efficiency of matchmaking by increasing the average ratio of "accepts" per ride order, and decreasing the average number of selected drivers per ride order. Another blog post discussed the Simulated Annealing algorithm that adds randomness to the exploration and enhances their TransLoc OnDemand system[3]. Although the goal and the approach of these projects is different from that of our project, there are aspects that we could refer to, such as how they frame the on demand transport problem and extract key features.

3 Task Definition

3.1 Scope

As mentioned above, to not overcomplicate the system, we are only going to look at the pickup and delivery problem of a single vehicle and without considering carpool situation. On a given map, the initial position of the driver can be anywhere. The driver is given a snapshot of current rider requests nearby whenever it is time to pick up a new rider, which is at start time of the game and the end time of each trip (when it just dropped off a rider). From the list of requests the driver selects the next customer to pick up. The list of requests and locations are dynamic so at any time the snapshot may look different. The reward of each trip is fixed, so is the driver's total driving time. But the travel time for each trip is different. The incentive is that the more requests the driver picks up the more reward he or she earns in the end.

3.2 Setup

Map: A list of 10 location zones of a city.
Location: An integer between 0 to 9 representing the location zone the driver or rider is at.
Input:

- A 10×10 map that gives the travel time between any source-destination pair. Data obtained from Uber Movement API.
- A list of requests represented as `(source, destination)` pairs.

Output: Ordered series of requests taken by the driver.
Reward: A fixed value of reward upon completing a single trip.
Goal: Maximize the drivers reward at the end of the driving period.
Topic: Reinforcement learning. Markov decision process. Hamiltonian path.

4 Example

For a concrete example, we gathered travel time data from Uber Movement for city Washington D.C. aggregated over the second quarter of year 2017. We downsampled the number of regions to 10. And we generated the travel time between these 10 locations. Missing data are augmented with mean travel time. Riders are generated using uniform random distribution among 10 locations. The driver could see 5 closest riders, but does not know the travel time beforehand. The driver must select a rider to pick up and drop off each round, and will be given a fixed reward for each trip. The driver will drive for 7200 seconds (driving period). The output is the sequence of riders the driver chose. In the end we compare how much reward the driver obtains.

5 Evaluation

We have implemented the basic framework, and a working baseline and oracle.
Baseline:

The baseline approach is for the driver to randomly pick the next request to take on. Under this approach, the driver's total reward is 7 on average, meaning that the driver takes on 7 ride requests on average.
Oracle:

The oracle is given the travel time of the requests in advance so it always chooses the rider with the smallest travel time. Since the driver cannot see all requests across the map, this may not be the optimal solution but should be very close. As the nature of the problem is $\mathcal{NP}$-complete, there is no other easier solution to this problem. Using this approach, the driver takes on 12 rides on average.
Gap:

In our implementation, we see a reasonable gap between the baseline and oracle, because of the extra information the oracle has. Our own approach of using reinforcement learning should render an schedule that has higher reward than the baseline approach but no better than the brute force approach.

6 Extensions

We could expand this model to incorporate carpool possibility, giving the car a capacity of more than one rider, which will make this problem more interesting and much more complicated. We could also add in more drivers to compete with each other. We could complicate the reward using fare estimate from Uber. We could also sample travel time from a distribution as opposed to a fixed value. At this stage, we find the above mentioned problem more challenging. They could serve as a extension of the easier problem.

References

[1] Manar I. Hosny, Christine L. Mumford. *The single vehicle pickup and delivery problem with time windows: intelligent operators for heuristic and metaheuristic algorithms.* Journal of Heuristics, 2008.

[2] Jan Mrkos, Jan Drchal, Malcolm Egan, Michal Jakob. *Liftago On-Demand Transport Dataset and Market Formation Algorithm Based on Machine Learning.* 2016.
`https://arxiv.org/abs/1608.02858v1`

[3] Michael Fogleman. *Ridesharing Algorithms in TransLoc OnDemand.*
`https://techlog.transloc.com/ridesharing-algorithms-in-transloc-ondemand-7acfeddbfe1f`