Multi-Objective Vehicle Rebalancing for Ridehailing System using a Reinforcement Learning Approach

Yuntian Deng¹, Hao Chen¹, Shao.367@osu.edu, Jiacheng Tang¹, Jianzong Pi¹, Abhishek Gupta¹

Abstract

The problem of designing a rebalancing algorithm for a large-scale ridehailing system with asymmetric (unbalanced) demand is considered here. We pose the rebalancing problem within a semi Markov decision problem (SMDP) framework with closed queues of vehicles serving stationary, but asymmetric demand, over a large city with multiple stations (representing neighborhoods). We assume that the passengers queue up at every station until they are matched with a vehicle. The goal of the SMDP is to minimize a convex combination of the waiting time of the passengers and the total empty vehicle miles traveled. The resulting SMDP appears to be difficult to solve yielding closed-form expression for the optimal rebalancing strategy. Consequently, we use a deep reinforcement learning algorithm to determine the approximately optimal solution to the SMDP. We show through extensive Monte Carlo simulations that the trained policy outperforms other well-known state-dependent rebalancing strategies.

Keywords: Reinforcement Learning, Ridehailing System, Multi-objective Optimization

1. Introduction

Mobility-on-Demand (MoD) systems, developed by transportation network companies (TNCs), are believed to make the urban transportation system more efficient and affordable. Uber, one of the leading companies, has achieved over $65 Billion in gross bookings in 2019, with a 31% year-over-year growth [1]. Such systems often induce an imbalance between the vehicle supply and the travel demand in some areas, especially during rush hours. Customers usually experience high prices and long waiting times while the drivers move around the city waiting for ride requests in low demand areas. This leads to fuel wastage and high passenger waiting times. Such issues increase the operational costs of the...
drivers and fleet managers and reduce the quality of service for the passengers.
As a result, TNCs experience an instant loss in revenue because of the imbalance
and a loss in business over time as fewer drivers and customers will enter the
market if the imbalance persists for a long duration.

Thus, rebalancing vehicles to address the imbalance is one of the major prob-
lems faced by ridehailing systems around the world. In this problem, assuming
the number of drivers is sufficiently large, there is an inherent trade-off between
customer waiting time and Empty Vehicle Miles Traveled (EVMT). Systems
with small passenger waiting time usually will have significantly larger EVMT,
thereby increasing the operational cost for the drivers (on average). Further, it
is not feasible in practice to know in advance the exact time of customer arrivals,
and thus it is difficult to make predictions and relocate vehicles in advance. To
address the last issue, past demand data can be used to construct a probabilistic
model for demand distribution over the city. This model can guide the choice of
rebalancing policies to minimize the expected operational cost and at the same
time, provide a reasonable expected waiting time for the customers.

Accordingly, in this paper, we model the rebalancing problem as a semi
Markov decision problem (SMDP) over a complete graph, in which nodes (re-
ferred to as stations) are the neighborhoods of the city and the edges represent
the road network with the associated travel time. In our model, we assume that
the passengers arrive at every station according to a Poisson distribution with a
known arrival rate. Every 100 seconds, the TNC platform makes a rebalancing
decision. Since this rebalancing control is applied every 100 seconds, it is a
semi Markov decision problem. The rebalancing policy of the platform takes as
input the current imbalance in the supply and the demand, and determines the
rebalancing control action to be taken under that state; as a result, this is a
state dependent rebalancing policy. The cost function for the SMDP is taken to
be a convex combination of the waiting time of the passengers and the EVMT.
The goal of the platform is to determine the optimal rebalancing policy.

Since the problem is difficult to solve analytically, we use reinforcement
learning to determine an approximately optimal state-dependent rebalancing
policy. We developed two tools that helped us use the reinforcement learning
algorithm:

1. We built a data processing tool that takes as input the New York taxi trip
data from December 2019 (a total of 7 million trips) and uses statistical
techniques to extract the arrival distribution of the passengers from the
Manhattan area of New York. The details of this data processing tool are
provided in Appendix A.

2. We built a simulator that takes as input the arrival distribution of the
passengers at each station, the rebalancing policy, and the fleet size of the
vehicles, and runs a Monte Carlo simulation to compute the waiting time
of passengers and EVMT traveled under that rebalancing policy. The code
for this simulator was written in Python.

We use proximal policy optimization (PPO) with a deep neural network as a
value function and policy function approximator to determine the approximately
optimal state-dependent rebalancing policy. We also compare the performance
of the trained rebalancing policy with other rebalancing policies in the literature
and show through Monte-Carlo simulations that the rebalancing policy designed
using PPO outperforms the other rebalancing policies in both metrics: Empty
Vehicle Miles Travelled (EMVT) and average passenger waiting time.

The rest of the paper is organized as follows. Section 2 reviews the current
literature on rebalancing policies and reinforcement learning-based approaches
for transportation systems. The SMDP problem is formulated in Section 3. Section
4 discuss the PPO algorithm and the experiment setup is presented in Section 5.
In Section 6 we present the simulation results and compare the trained policy
with the other rebalancing policies. Finally, Section 7 concludes this paper and
presents some directions for future work.

2. Related Work

We review some related literature in this section.

2.1. Rebalancing Methods

Devising rebalancing policies for TNCs has received significant attention over
the past decade, where the demand is usually spatiotemporal stochastic [2] and
time-constrained [3]. We classify them into two parts based on methodologies:
state-independent policy and state-dependent policy.

The state-independent policies balance the vehicle distribution based on the
average dynamics of the system. Pavone et al. [4] utilized a fluid model for
vehicles and developed a rebalancing policy to ensure that the system reaches
equilibrium, at which there are empty vehicles at every station and no waiting
customers at each station. The rebalancing policy is posed as a solution to a Lin-
ear Programming (LP) problem that minimizes the cost of rebalancing. Smith
et al. [5] proposed a mobility-on-demand system mixing both customer-driven
vehicles and taxi service. With a requirement of bounded waiting customers,
both the number of vehicles and the number of drivers are minimized by solving
two minimum cost flow problems. Rossi et al. [6] also introduced a flow-based
network model that is congestion sensitive. The rebalancing policy is devised
by solving an LP that minimizes the total trip time and fleet size with the con-
gestion entering the constraints of the LP. Sayarshad and Chow [7] proposed
a new queuing-based formulation for relocating idle vehicles and developed a
Lagrangian decomposition heuristic. Considering estimated future states, the
proposed algorithm outperforms the myopic policy when minimizing the op-
erational cost and queuing delay. The queuing network model is also used in
Zhang et al. [8] and Iglesias et al. [9]. The rebalancing process in [8] is treated
as a virtual arrival process, which is independent of the real arrival process.
The rebalancing policy is an LP that minimizes the total rebalancing cost while
keeping an identical demand-supply ratio (also known as utilization) among all
the stations. The rebalancing policy developed in [9] also considers the conges-
tion on the road as a constraint in the LP formulation of the rebalancing cost
optimization problem.
However, the above policies are based on the steady-state analysis since they use either a fluid model or a queuing model. The steady-state analysis can only provide asymptotic optimality and fails to react to the current state and the dynamical evolution of the system. It also leads to a state-independent randomized rebalancing control policy. Moreover, the policies in [8, 9] also require impatient passengers who quit the system if not matched to a vehicle immediately, since it simplifies the system dynamics and allows us to carry out the steady-state analysis.

State-dependent policies, where rebalancing control is dependent on the current queue lengths of empty vehicles and waiting passengers, can achieve significantly better performance. Banerjee et al. [10] introduce Scaled MaxWeight rebalancing policy, which is a state-dependent rebalancing policy. This paper shows that no state-independent policy can achieve exponential decay of demand dropping probability with respect to the fleet size, while naive state-dependent policy (MaxWeight) achieves exponential decay without knowing demand arrival rates. Similarly, Kanoria and Qian [11] propose Mirror Backpressure policy based on an increasing function, called score function, of queue length, allowing for more general demand arrival rates. These two policies assume that the vehicles can move from one station to another within one timestep, which is inappropriate in a city with large and asymmetric trip times. This modeling assumption is satisfied in a restricted downtown area, where the distance between two locations is generally small.

2.2. Dynamic Programming and Reinforcement Learning

The usual approach for solving MDPs is value iteration or policy iteration [12]. However, if the state space is large or the value or policy functions cannot be stored in a compact form, then reinforcement learning (RL) has emerged as a remarkable tool for devising approximately optimal policies [13]. RL algorithms use data from a simulator or a real-world application to determine optimal decision policies in MDPs. Since in MoD systems, the state and action spaces are large and the value and policy functions are difficult to store, RL with an appropriate function approximator is a suitable tool for devising approximately optimal rebalancing policies.

An et al. [14] model the rebalancing problem in the car-sharing system as a Markov Decision Process and set picking bonus and parking bonus for users as action. The state is defined as the number of remaining vehicles at each station. Deep Deterministic Policy Gradient algorithm is used to approximate the value function as the action space is continuous.

Wen, Zhao, and Jaillet [15] utilize the deep Q-network for rebalancing the shared mobility-on-demand systems. The state space is defined as the distribution of idle vehicles, in-service vehicles, and the predicted demand (passenger arrival rate). Each vehicle either stays put or moves to the highest value neighborhood (computed appropriately). One case study of London shows the deep Q-network performs effectively by reducing the waiting time of customers. However, future demand is difficult to estimate in practice, and relying on this information makes it less realistic to be implemented.
Gueriau and Dusparic [16] propose a Q-learning-based decentralized approach to vehicle relocation as well as ride request assignment in shared mobility-on-demand systems. The full agent state consists of (i) vehicle state (empty, one passenger or full), (ii) presence of currently active requests in agent’s zone (yes, no), and (iii) presence of currently active requests in neighboring zones. Three basic actions an agent can execute are defined as pickup, rebalance, and doNothing. Both ridehailing (one request by one vehicle) and ride-sharing (multiple requests by one vehicle) cases are evaluated through the demand generated from the NYC taxi dataset. However, the rebalancing strategies are limited in their setting, where agents can only choose one of four existing heuristic policies and may not execute the optimal rebalancing action.

Motivated by these studies, we design a more general rebalancing strategy, which requires only the passenger queue and vehicle queue length, without the need for demand prediction. Also, the control policy directly specifies the number of empty vehicles to be dispatched and the destination of these vehicles. This allows us to explore a wider class of policies, and as simulation suggests, gives rise to a better rebalancing strategy. We design our total cost as a convex combination of vehicle rebalancing miles and passenger waiting time with an adjustable parameter that controls the relative loss associated with each component. These two components represent the major trade-off in the system: the operational cost of EVMT v.s. quality of service. By combining these two, service providers can fine-tune the algorithm to fit their specific preferences.

3. System Model

The city is divided into $n$ regions. We model the city as a complete graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N} = \{1, \ldots, n\}$ is the set of stations and $\mathcal{E} = \{(i, j) | i, j \in \mathcal{N}\}$ is the road connecting the two stations $i, j$. Station $i \in \mathcal{N}$ is the centroid of the corresponding region. We let $t_{ij}$ denote the deterministic travel time between origin station $i$ and destination station $j$. If $t_{ij}$ is not an integer, we replace it with $\lceil t_{ij} \rceil$, where $\lceil x \rceil = \min\{m \in \mathbb{Z} | m \geq x\}$ is the ceiling function.

Each passenger enters the system with a starting station $i \in \mathcal{N}$ and destination station $j \in \mathcal{N}$. Passenger arrival for each origin and destination pair $(i, j)$ is assumed to follow a Poisson process with the arrival rate denoted by $\lambda_{ij}$. We assume that $\lambda_{ii} = 0$. Such an assumption on the arrival process is cross-validated by the New York Taxi data, and the details are presented in Section 5.1 and Appendix A. The interarrival time between two passengers is assumed to be independent of all other randomnesses in the system. At each station $i$, passengers who originated at this station form a First-Come-First-Serve (FCFS) queue regardless of their destinations.

The city is a closed system with respect to $m$ vehicles, that is, a fleet of $m$ vehicles are serving the demand. At every time step, each vehicle can have only two statuses: either be traveling between two stations or wait at a station with no passengers. If the vehicle is traveling between the two stations, then it can transport a passenger or be rebalanced. Each vehicle can carry one
passenger at most, and we assume that there is no ride-sharing between two
distinct passengers.

3.1. System Dynamics

We now introduce the notations to capture the system dynamics. We
define that $t$ is a discrete time index. Let $p_i[t]$ and $v_i[t]$ denote the number of
passengers and vehicles at station $i$ by time step $t$ before the platform matches
the passengers and vehicles. Let $p_{ij}[t]$ be the number of passengers who are
waiting to reach destination $j$. Let $b_{ij}[t]$ denote the number of new passengers
arriving at station $i$ with destination $j$ during time interval $[t - 1, t)$. Further,
we have $p_i[t] = \sum_{j \neq i} p_{ij}[t]$ and $b_i[t] = \sum_{j \neq i} b_{ij}[t]$. Let $x_{ij}[t]$ and $y_{ij}[t]$ denote,
respectively, the numbers of non-empty vehicles and empty vehicles dispatched
from $i$ to $j$ at time step $t$, respectively. The number of passengers leaving station
$i$ is $x_i[t] = \sum_{j \neq i} x_{ij}[t]$.

It is further assumed that waiting passengers get matched as soon as there
are empty vehicles at the same station, i.e., the number of passengers departing
station $i$, $x_i[t]$ is the minimum of the current passenger queue length $p_i[t]$ and
the current number of vehicles available $v_i[t]$ at that station. This leads to the
following expression

$$x_i[t] = \min\{p_i[t], v_i[t]\} \quad \text{for all } i \in \mathcal{N}.$$  \hfill (1)

Since there are at most $v_i[t]$ vehicles available, the total number of vehicles
leaving a station regardless of transporting passengers or rebalancing is upper
bounded by

$$x_i[t] + y_i[t] \leq v_i[t] \quad \text{for all } i \in \mathcal{N}.$$  \hfill (2)

Combining (1) and (2), the constraint on the number of empty vehicles
dispatched from each station is

$$y_i[t] \triangleq \sum_{j \neq i} y_{ij}[t] \leq \max\{v_i[t] - p_i[t], 0\} \quad \text{for all } i \in \mathcal{N}.$$  \hfill (3)

The total number of vehicles arriving at station $i$ within the time interval
$[t - 1, t)$ is given by $\sum_{j \neq i} (x_{ji}[t - t_{ji}] + y_{ji}[t - t_{ji}])$. For each station $i$, $p_i[t]$, the
number of waiting passengers by time step $t$, can be updated recursively
based on passenger departures $x_i[t - 1]$ and newly arrivals $b_i[t]$ within one time
slot. Similarly, the number of vehicles available at station $i$ can be updated
recursively based on vehicle departures due to transporting passengers $x_i[t - 1]$ or rebalancing $y_i[t - 1]$, and vehicles arriving from other stations. Thus, the
update equation for each station is
\[ p_i[t] = p_i[t-1] - x_i[t-1] + b_i[t], \]  
\[ v_i[t] = v_i[t-1] - x_i[t-1] - y_i[t-1] + \sum_{j \neq i} (x_{ji}[t-t_{ji}] + y_{ji}[t-t_{ji}]). \] (5)

The goal is to find the set of rebalancing vehicles \( \{y_{ij}[t]\}_{i,j \in \mathcal{N}} \) such that the average passenger waiting time and total Vehicle Miles Traveled (VMT) due to rebalancing are optimized. The following subsection will address the optimization problem more precisely.

### 3.2. SMDP formulation

The optimization problem of rebalancing vehicles is modeled as a semi Markov Decision Process (SMDP), which is defined by a tuple \((\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)\). \( \mathcal{S} \) and \( \mathcal{A} \) represent the state and action space. \( \mathcal{P} \) is the transition probability matrix. \( R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \) is the reward function and \( \gamma \) is the discount factor. We explain the details as follows:

- **State Space \( \mathcal{S} \):**
  
  The state \( s[t] \in \mathcal{S} \) is defined as the information set including passenger queue length \( p[t] \triangleq \{p_i[t]\}_{i=1}^{n} \), vehicle queue length \( v[t] \triangleq \{v_i[t]\}_{i=1}^{n} \), and the history of vehicle departures \( (x^{t-1}, y^{t-1}) \) due to transporting passengers and rebalancing, up to the maximum traveling time \( t_{\text{max}} \) between any two of the stations, that is:

  \[ x^{t-1} \triangleq \{x_{ij}[\tau] | i, j \in \mathcal{N}; t - t_{\text{max}} \leq \tau \leq t - 1 \}, \]
  \[ y^{t-1} \triangleq \{y_{ij}[\tau] | i, j \in \mathcal{N}; t - t_{\text{max}} \leq \tau \leq t - 1 \}, \]
  \[ s[t] \triangleq \{p[t], v[t], x^{t-1}, y^{t-1}\} \in \mathbb{N}^{2(n+n^2t_{\text{max}})}. \]

- **Action Space \( \mathcal{A} \):**
  
  The action space of the platform is defined as follows: If there is a station with empty vehicles, then the platform dispatches a fraction of these empty vehicles to one of the \( k \) nearest neighbors of that station. If a station has no empty vehicle, then no vehicle is dispatched to any of the other stations. Denote the set of \( k \) nearest stations to station \( i \) as \( \text{Neighbor}(i) \), where \( i \notin \text{Neighbor}(i) \). Define the set \( \mathcal{M} = \{0, \frac{1}{M}, \frac{2}{M}, \ldots, 1\} \), which is the set of all possible fractions of empty vehicles that can be dispatched from any one station to another.\(^1\)

\(^1\)The action space could be taken to be closed interval \([0, 1]\); however, this leads to slower convergence, so we discretized the action space to accelerate convergence.
Let $a_i[t] = [a_{ij_1}[t], a_{ij_2}[t], \ldots, a_{ij_k}[t]]^T \in \mathcal{M}^{k+1}$ denote the action at station $i$. Here, $\{j_1, \ldots, j_k\} := \text{Neighbor}(i)$. The variable $a_{ij_k}[t] \in \mathcal{M}$ denotes the fraction of vehicles that is dispatched from station $i$ to station $j_k$. We further assume that, the action $a_i[t]$, $\forall t \geq 0$ satisfies the following three conditions:

1. there exists only one $j^* \in \text{Neighbor}(i)$, s.t. $a_{ij^*}[t] > 0$,
2. $a_{ij}[t] = 0$, if $j \in \text{Neighbor}(i) \setminus \{j^*\}$,
3. $\sum_{j \in \text{Neighbor}(i) \cup \{i\}} a_{ij}[t] = 1$.

The first and second conditions mean that if, at station $i$ the number of vehicles $v_i[t]$ is greater than the number of waiting passengers $p_i[t]$, then only one station $j^* \in \text{Neighbor}(i)$ is picked and $[a_{ij^*}[t] \max(v_i[t] - p_i[t], 0)]$ number of empty vehicles are dispatched to station $j^*$. The second and third conditions imply that, $a_{ii}[t] = 1 - a_{ij^*}[t]$, which means that the platform will keep the remaining $\lceil (1 - a_{ij^*}[t]) \max(v_i[t] - p_i[t], 0) \rceil$ vehicles at station $i$. The platform follows the same procedure for all the stations with excess number of vehicles.

Let $a[t] := [a_1^T[t], \ldots, a_n^T[t]]^T$ denote the action of the platform. We thus define the action space $\mathcal{A}$ as

$$\mathcal{A} = \left\{ a[t] \in \mathcal{M}^{(k+1)n} \mid a_{ij}[t] \text{ satisfies } 1, 2, 3, \ \forall \ t \geq 0 \ \forall \ i \in \mathcal{N} \right\}.$$  

We apply the rebalancing action every few time steps. For our simulations, the time step of the SMDP is 1 second whereas the rebalancing action is applied every 100 seconds.

- **Transition Probability $\mathcal{P}$**: The transition probability is a map $\mathcal{P} : S \times \mathcal{A} \times S \rightarrow [0,1]$. Since the arrival of passengers follows a Poisson process (memoryless), the state $s[t+1]$ follows the distribution $\mathcal{P}(s[t+1] \mid s[t], a[t])$, which depends on the arrival rates $\lambda_{ij}$, travel time $t_{ij}$. This transition probability is induced by the state update equation and the actions introduced above.

In transportation systems, the quality of service for passengers in terms of their waiting time and the operational cost of rebalancing the fleet need to be traded off. As stated in [17], the customer receives a shorter waiting time with ride-hailing apps but ridehailing leads to approximately 83.5% more VMT than non-ridehailing cases. We explicitly address this trade-off by picking a cost function that is a convex combination of these two quantities.

- **Reward Function $R$**: According to Little’s Law [18], the passenger waiting time is proportional to the passenger queue length. Thus, passenger queue length $p_i[t]$ is used to compute passenger waiting time in the cost function. The reward function (negative of cost function) is written as a
convex combination of passenger queue length and the distance traveled by empty vehicles with a weight $\alpha$ where $0 \leq \alpha \leq 1$:

$$R(s[t], a[t]) = -(1 - \alpha) \sum_{i=1}^{n} p_i[t]$$

$$- \alpha \sum_{i=1}^{n} \sum_{j \in \text{Neighbor}(i)} \max(v_i[t] - p_i[t], 0) a_{ij}[t] t_{ij} v.$$  \hspace{1cm} (6)

The first term in the reward function uses $p_i[t]$ to represent the total waiting time for all passengers. The second term in the reward function calculates EVMT for rebalancing with $\alpha$ presents the preference of the service providers between the passenger waiting time and EVMT.

**Remark 1.** $\sum_{i=1}^{n} p_i[t]$ measures the total waiting time for all passengers and $\frac{\sum_{i=1}^{n} p_i[t]}{\sum_{i=1}^{n} \sum_{j \in \text{Neighbor}(i)}}$ measures the average waiting time per passenger. In case the platform wants to minimize the average waiting time per passenger, then $\sum_{i=1}^{n} p_i[t]$ in the reward function can be replaced with $\frac{\sum_{i=1}^{n} p_i[t]}{\sum_{i=1}^{n} \sum_{j \in \text{Neighbor}(i)}}$.

In the second term of reward function, $\max(v_i[t] - p_i[t], 0)$ is the number of vehicles available at station $i$ after serving the local demand at station $i$. The rebalancing distance for vehicles traveling from station $i$ to $j$ is then determined by $a_{ij}[t] t_{ij} v$ where $t_{ij} v$ is the distance between station $i$ and $j$, which is calculated by multiplying the fixed trip time $t_{ij}$ and the average speed $v$ of the vehicles. Summing over $i \in \mathcal{N}$ and $j \in \text{Neighbor}(i)$ gives the total empty vehicle miles traveled.

- **Discount factor $\gamma$:** The discount factor $\gamma$ is set to be 0.99 with respect to the time step $t = 1$ second for a infinite horizon MDP Problem.

Given a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, the corresponding objective function of the optimization problem can be then written as

$$J(\pi) \triangleq \mathbb{E}_{s_0}^{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R(s[100t], a[100t]) \right], \hspace{1cm} (7)$$

where $s_0$ is the initial state, $a[100t] \sim \pi(\cdot | s[100t])$ and $a[q] = 0$ for $q \neq 100t$, and $s[t+1] \sim \mathbb{P}(\cdot | s[t], a[t])$. The optimal dispatch policy $\pi^*$ maximizes the discounted reward $J(\pi)$. We again emphasize that the rebalancing action is applied every 100 seconds in our simulator. Hence, this is an SMDP problem.

As we can observe, the objective function and the state update equation is very complicated, due to which solving this analytically is seemingly difficult.
Table 1: The notations used throughout the paper.

| Notations | Meaning |
|-----------|---------|
| $\mathcal{N}$ | station set |
| $\mathcal{E}$ | topology set between stations |
| $\mathcal{S}$ | state space |
| $\mathcal{A}$ | action space |
| $\mathcal{P}$ | state transition probability |
| $\gamma$ | discount factor |
| $\pi_\theta(a|s)$, $\pi_\theta$ | parameterized policy with parameter $\theta$ |
| $R(s, a)$ | reward given state $s$ and action $a$ |
| $V_\pi(s)$ | state value function under $\pi$ |
| $Q_\pi(s, a)$ | action value function under $\pi$ |
| $\eta(\pi)$ | expected reward under $\pi$ |
| $A_\pi(s, a)$ | advantage function under $\pi$ |
| $L_\pi(\tilde{\pi})$ | expected advantage function under $\pi$ |
| $M_\pi(\tilde{\pi})$ | surrogate objective |
| $D_{KL}(\pi, \tilde{\pi})$ | Kullback–Leibler divergence |

As a result, we adopt an RL approach to devise an approximately optimal rebalancing policy. We next review the theory of proximal policy optimization with a neural network as a function approximator, which is the primary RL tool used to solve this problem in our paper. There are, of course, other RL algorithms with other function approximators that can be used to solve this problem.

4. Solution Approach: Proximal Policy Optimization

In this section, we discuss the proximal policy optimization algorithm (PPO) adopted for the taxi rebalancing problem. PPO is the state-of-the-art policy optimization algorithm that has gained impressive success in continuous state space control problems [19]. In general, policy optimization algorithms use a parameterized stochastic policy function $\pi_\theta(a|s)$ (the probability of choosing action $a$ given state $s$ and parameter $\theta$) and formulates a general nonlinear optimization problem of the expected discounted reward $J(\theta) := J(\pi_\theta)$ (we have slightly abused the notation here). In practice, the parameterized policy $\pi_\theta(a|s)$ is often represented in the form of a neural network. Therefore, it is restricted to a small class of functions $\Pi = \{\pi(a|s, \theta) : \theta \in \Theta\}$ where $\Theta$ is the neural network parameter space. We use $\pi_\theta$ to denote $\pi_\theta(a|s)$, and the goal of our problem becomes:

$$\max_{\theta \in \Theta} J(\theta) = \max_{\theta \in \Theta} \mathbb{E}_{s^0}^{\pi_\theta} \left[ \sum_{t=0}^{\infty} \gamma^t R(s[100t], a[100t]) \right].$$
The PPO algorithm consists of two main components: the surrogate objective for expected policy reward; and the policy neural network and value function neural network with different or shared parameters, which are introduced in the following two subsections.

4.1. Surrogate Objective

Rather than directly optimizing \( J(\theta) \), PPO considers the problem of maximizing policy improvement \( J(\theta') - J(\theta) \) as follows:

\[
\max_{\theta' \in \Theta} [J(\theta') - J(\theta)]
\]  

Schulman et al. [20] provide an explicit lower bound on the expected reward of a new policy \( J(\theta') \) relative to the expected reward of the old policy \( J(\theta) \). To introduce this bound, we need some notations. Let

\[
A_\theta(s,a) := Q_\theta(s,a) - V_\theta(s),
\]

which is called the advantage of action \( a \) over other actions under state \( s \).

Let \( D_{KL}(\cdot || \cdot) \) denote the KL-divergence between two probability distributions. Then, [20] showed that the difference \( J(\theta') - J(\theta) \) is bounded by

\[
J(\theta') - J(\theta) \geq L_\theta(\theta') - CD_{KL}^{\max}(\theta, \theta').
\]  

where, \( C = \frac{4\gamma}{(1-\gamma)^2}, \epsilon = \max_{s,a} |A_\theta(s,a)|, \)

\[
L_\theta(\theta') \triangleq \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta} \left[ \frac{\pi_\theta'(a|s)}{\pi_\theta(a|s)} A_\theta(s,a) \right],
\]

\[
D_{KL}^{\max}(\theta, \theta') = \max_{s \in S} D_{KL}(\pi_\theta(\cdot|s)||\pi_{\theta'}(\cdot|s)).
\]

Given the current policy \( \pi_\theta \) at each iteration, PPO algorithm obtains the updated policy \( \pi_{\theta'} \) that maximizes the surrogate objective \( L_\theta(\theta') - CD_{KL}^{\max}(\theta, \theta') \).

Due to [9], the PPO algorithm is guaranteed to generate a monotonically improving sequence of policies. Accordingly, the original policy improvement maximization be converted into an equivalent problem of maximizing its lower bound as follows:

\[
\max_{\theta' \in \Theta} [L_\theta(\theta') - CD_{KL}^{\max}(\theta', \theta)].
\]

Trust region constraint

This subsection shows how to solve (10) by rewriting it to its equivalent trust region formulation. This transformation is used mainly for two practical reasons [20]: 1) if the formulation in (10) and the KL-divergence coefficient \( C \) takes the value given in [9], then the step sizes would be very small; 2) the \( \delta \) parameter in (12) is easier to tune as it directly represents the maximum allowed change from \( \theta \) to \( \theta' \).
maximize \( L_\theta(\theta') \)
\[ (11) \]
s.t. \( D_{KL}^{\max}(\theta, \theta') \leq \delta. \)  
\[ (12) \]

However, [21] proposed a third formulation, the clipped surrogate objective \( L_{CLIP} \), which is much simpler to implement and tune. Since the maximization in Equation (11) contains a probability ratio between \( \pi_{\theta'}(a|s) \) and \( \pi_{\theta}(a|s) \) which reflects the size of the policy update, we can restrict the probability ratio to be within some neighborhood around 1 and clip away the large updates. i.e. \( \frac{\pi_{\theta'}(a|s)}{\pi_{\theta}(a|s)} \in [1-\epsilon, 1+\epsilon] \). The objective function in line 9 of Algorithm 1 uses the clipped surrogate objective. The details can be found in [21].

### 4.2. The Complete Algorithm

So far, we have discussed the surrogate objective and how it forms the clipped objective \( L_{CLIP} \) used in the algorithm. Now, we can construct the complete algorithm as shown in Algorithm 1. \( \theta_1, \theta_2 \in \Theta \) are the parameters for policy network and value function network respectively. They are initialized arbitrarily as well as the initial state \( s_0 \). For each iteration, the algorithm samples the environment up to \( T \) timesteps using the policy from the previous iteration. The advantage \( \hat{A}_t \) at timestep \( t \) is then estimated using the samples and the approximated value function \( V(s, \theta_2) \) [22]. After that, \( K \) gradient updates (epochs) are performed by drawing minibatches of samples. The gradient of \( L_{CLIP} \) w.r.t \( \theta_1 \) in line 9 and the gradient of value function loss w.r.t \( \theta_2 \) in line 10 are obtained using the automatic differentiation engine in the Tensorflow framework.

### 5. Experiment Setup

In this section, we discuss the details of our simulation setup. We first discuss the procedure adapted for data processing of New York Taxi data to yield arrival rates of passengers. Thereafter, we present the details of the simulator written in Python to carry out the RL implementation. Finally, we review various rebalancing algorithms that have been proposed in the recent literature for benchmarking the performance of the trained policy.

#### 5.1. Data Processing

We obtain the data from New York City Taxi and Limousine Commission website [23]. The data is analyzed to extract information on the passenger arrival process. We specifically pick the Manhattan borough yellow cab trip records between 8 am to 9 am on weekdays in December 2019. This time window is chosen because we observed that the demand for taxis is high in this time interval and the demand is highly asymmetric. The imbalanced rush hour scenario poses a greater challenge to the rebalancing policy. From our data analysis, we found that the real demand follows Poisson arrival since the
Algorithm 1: Proximal Policy Optimization \cite{21}

1. Initialize: $\theta_1, \theta_2, \pi_0, s_0$
2. $a_0 \sim \pi_0(a|s_0)$
3. For iteration: $i=1, 2, \ldots$ do
   4. Run policy $\pi_{i-1}$ for $T$ timesteps
   5. Collect samples $S_i = \{(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_{T-1}, a_{T-1}, r_{T-1}), (s_T, a_T, r_T)\}$
   6. Compute advantage estimates $\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_{T-1}$
   7. For epoch: $k=1, 2, \ldots, K$ do
      8. Draw a minibatch $B_k \subset S_i$ from collected samples in this iteration
      9. Compute gradient $g_1 \leftarrow \nabla_{\theta_1} L_{CLIP}^{CLIP}$
     10. Compute gradient $g_2 \leftarrow \nabla_{\theta_2} (V(s_t, \theta_2^i) - r_t - V(s_2, \theta_2^{i-1}))^2$
     11. $\theta_1 \leftarrow \theta_1 + \alpha g_1$
     12. $\theta_2 \leftarrow \theta_2 + \alpha g_2$
   13. end
   14. end
15. Output: $\theta_1, \theta_2$

interarrival time distribution has a good fit with the exponential distribution.

For more details of data processing, we refer the reader to Appendix A.

5.2. Simulator

Our simulator has a stationary passenger arrival process simulated by a Poisson process with arrival rates estimated from the selected data set. The topology of the network is modeled after the geographic taxi zones in the Manhattan borough minus some isolated zones that have no ground connection with others. The complete Manhattan taxi zone map can be found in Figure A.5. The simulator has 63 zones in total, and one can configure it to use any subset of the zones. Figure 1 shows the topology based on a 20 stations network around Midtown Manhattan. Our RL simulations are conducted for the demand across these 20 stations.

The entire process of the simulator can be summarized as four steps: 1) passenger generation; 2) matching vehicles and passengers; 3) dispatch empty vehicles; 4) vehicle arrival. Passengers accumulate at the stations if there are not enough vehicles to pick them up. When a vehicle is at a station with no passenger waiting, the taxi can either choose to wait at the station or be dispatched (rebalanced) to the other stations with higher demands. The simulator provides performance statistics for comparison between rebalancing policies, which includes the average passenger waiting time, number of rebalancing trips, and rebalancing miles traveled.

Here, we list the general simulator settings that common for both the PPO algorithm and the benchmarks. The simulator updates the passenger queues,
Table 2: Distances between every pair of stations in the 20 stations network in miles. We assume that the average velocity is 10 miles per hour on the road. Thus, $t_{\text{max}} = 22$ mins.

| Origin | Destination | 48  | 68  | 100 | 107 | 140 | 141 | 142 | 143 | 161 | 162 | 170 | 186 | 229 | 234 | 236 | 237 | 238 | 262 | 263 |
|--------|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 48     | 0.00 | 0.74 | 0.63 | 2.43 | 2.09 | 0.61 | 0.30 | 0.84 | 1.21 | 0.83 | 0.32 | 1.71 | 0.42 | 2.29 | 1.88 | 1.29 | 0.88 | 3.01 | 2.70 |
| 68     | 0.74 | 0.00 | 0.78 | 1.12 | 2.14 | 0.81 | 1.60 | 1.55 | 1.91 | 1.48 | 0.52 | 2.41 | 0.67 | 3.03 | 2.40 | 2.03 | 1.62 | 3.75 | 3.43 |
| 100    | 0.63 | 2.09 | 0.00 | 0.46 | 2.36 | 2.03 | 0.63 | 0.44 | 0.77 | 1.14 | 0.72 | 0.27 | 1.63 | 0.26 | 2.25 | 1.63 | 1.31 | 0.91 | 2.65 |
| 107    | 0.31 | 0.28 | 2.09 | 0.00 | 2.14 | 1.78 | 0.72 | 0.79 | 0.60 | 0.89 | 0.44 | 0.62 | 1.36 | 0.45 | 2.05 | 1.41 | 1.29 | 0.97 | 2.70 |
| 140    | 0.29 | 0.84 | 0.18 | 0.63 | 0.00 | 0.34 | 1.86 | 1.52 | 0.00 | 0.42 | 0.73 | 0.54 | 0.89 | 1.38 | 0.68 | 1.70 | 1.10 | 0.68 | 2.31 |
| 141    | 0.68 | 1.21 | 0.74 | 1.18 | 2.29 | 0.80 | 0.69 | 0.92 | 1.40 | 0.46 | 0.00 | 1.89 | 0.21 | 2.52 | 1.89 | 1.57 | 1.17 | 3.32 | 2.92 |
| 142    | 0.83 | 1.68 | 0.30 | 0.73 | 1.02 | 2.62 | 2.02 | 0.69 | 0.62 | 1.63 | 1.40 | 0.86 | 0.00 | 1.89 | 0.21 | 2.52 | 1.89 | 1.57 | 1.17 |
| 143    | 0.83 | 1.68 | 0.42 | 0.74 | 1.18 | 1.60 | 0.87 | 0.50 | 0.94 | 1.89 | 0.00 | 1.76 | 0.72 | 0.23 | 0.86 | 1.07 | 1.34 | 1.06 |
| 161    | 0.32 | 0.74 | 0.74 | 1.18 | 1.60 | 0.87 | 0.50 | 0.94 | 1.89 | 0.00 | 1.76 | 0.72 | 0.23 | 0.86 | 1.07 | 1.34 | 1.06 |
| 162    | 0.42 | 0.67 | 0.28 | 0.45 | 2.51 | 1.29 | 0.88 | 0.71 | 0.94 | 1.29 | 0.84 | 0.21 | 1.78 | 0.00 | 2.43 | 1.80 | 1.55 | 1.17 | 2.82 |
| 170    | 0.29 | 1.03 | 1.25 | 2.05 | 2.05 | 0.33 | 0.31 | 1.70 | 2.12 | 1.49 | 1.15 | 1.61 | 2.52 | 0.72 | 2.48 | 0.00 | 0.64 | 1.13 | 1.50 | 0.75 | 0.42 |
| 186    | 1.68 | 2.49 | 1.63 | 1.41 | 0.76 | 0.42 | 1.10 | 1.53 | 0.86 | 0.52 | 0.97 | 1.89 | 0.23 | 1.80 | 0.64 | 0.00 | 0.68 | 0.94 | 1.33 | 1.03 |
| 229    | 1.29 | 2.03 | 1.31 | 1.29 | 1.36 | 1.64 | 0.66 | 0.65 | 0.72 | 0.67 | 0.92 | 1.57 | 0.86 | 1.55 | 1.53 | 0.68 | 0.00 | 0.41 | 1.86 | 1.54 |
| 234    | 0.88 | 1.52 | 0.91 | 0.97 | 2.49 | 1.63 | 0.29 | 0.50 | 0.68 | 0.68 | 0.17 | 0.77 | 1.07 | 1.07 | 1.17 | 1.50 | 0.94 | 0.00 | 2.22 | 1.91 |
| 236    | 3.01 | 3.73 | 2.95 | 2.70 | 0.60 | 0.92 | 2.42 | 2.84 | 2.18 | 1.62 | 2.27 | 3.22 | 1.34 | 1.11 | 0.73 | 1.33 | 1.96 | 2.22 | 0.00 | 0.52 |
| 237    | 2.70 | 3.43 | 2.65 | 2.42 | 0.30 | 0.64 | 2.11 | 2.53 | 1.89 | 1.53 | 1.99 | 2.92 | 1.06 | 2.52 | 0.42 | 1.01 | 1.54 | 1.91 | 0.32 | 0.00 |

5.3. PPO algorithm settings

We use neural networks to approximate state-value function $V(s)$ and policy function $\pi(a|s)$. A summary of the configurations for the PPO algorithm is shown in Table 3. We set the discount factor $\gamma = 0.99$. The neural network takes as input the current state of the system, which is a vector in $\mathbb{N}^{2(n+n^2 \times t_{\text{max}})} = \mathbb{N}^{2(20+400 \times 22 \times 60)}$ — this is over a million dimensional input. Both state-value function approximator $V(s)$ and policy function approximator $\pi(a|s)$ share two
hidden layers, each with 256 units and tanh nonlinearity, in the neural network. The output of the state-value network $V(s)$ is a scalar in $\mathbb{R}$ and we use a linear function of the results in the second hidden layer to compute it. In the policy network $\pi(a|s)$, the output is a vector in $[0, 1]^{(M+1)(k+1)n} = [0, 1]^{6\times6\times20}$, where we have taken $M = 5$ and $k = 5$. The rebalancing algorithm will pick the action $a$ according to the probability $\pi(a|s)$.

The initial weights for the neural network are drawn from a uniform distribution. The optimizer of choice is the Adam algorithm [24]. The training process takes 2000 iterations. Within one iteration, the PPO algorithm uses a fixed policy to sample state, action, and reward data from the simulator. Each iteration consists of 4000 timesteps, and one timestep equals one rebalancing interval mentioned in the above section which is 100 seconds in simulation time. The optimizer performs 30 gradient ascent steps on the samples collected in each iteration. The sample sets are randomly picked to form a minibatch of 128 samples on which the gradient estimation is performed. The training process is executed using Ray [25], a distributed training framework, which allows us to run multiple sampling agent simultaneously. After training, the trained policy is applied to 10 test episodes. The average performance metrics of the test episodes are used for comparison with the benchmarks.

| Parameter                  | Value   |
|----------------------------|---------|
| optimizer                  | Adam    |
| learning rate              | $3 \times 10^{-4}$ |
| discount $(\gamma)$        | 0.99    |
| training batch size        | 4000    |
| SGD minibatch size         | 128     |
| number of hidden layers    | 2       |
| activation function        | tanh    |
| number of hidden units per layer | 256  |
| number of samples per minibatch | 256  |
| gradient steps             | 30      |

Table 3: PPO Hyperparameters used in the training.

5.4. Benchmarks

The rebalancing policy we obtained through the application of PPO algorithm is a state-dependent rebalancing policy. Recall that in our problem, the number of vehicles remain the same throughout – which leads to a closed queueing system. There are only a few state-dependent rebalancing policies for ride-hailing problems that are modeled as closed queueing system, which we discuss next. We compare our rebalancing policy with these rebalancing policies in the sequel and show that our rebalancing policy strictly outperforms these rebalancing policies (for the reward function chosen above). In order to maintain a fair comparison, the following benchmarks are performed every 100 seconds, the same frequency as in our algorithm.
5.4.1. MaxWeight

MaxWeight policy is an online scheduling algorithm and has been used extensively across a wide range of applications. For the ridehailing system, the properties of MaxWeight policy are studied in [10]. The policy works as follows: If a passenger arrives at station $i \in N$ and the number of vehicles at that station is zero $v_i[t] = 0$, then the algorithm will dispatch an empty vehicle from a nearby station $j^*$ given by

$$j^*_i[t] = \text{argmax}_{j \in \text{Neighbor}(i)} v_j[t].$$

The platform matches an empty vehicle at $j^*$ to the passenger at station $i$. The dispatch continues until all passengers get matched. It is assumed that if the passenger is not matched immediately (because there are no empty vehicles at any of the neighbors of $i$), then the passengers leave the system. In our situation, the passengers queue up to receive the service. Accordingly, in our implementation, the passenger at station $i$ waits until an empty vehicle appears at station $i$ or in the neighborhood of station $i$.

5.4.2. Mirror BackPressure

This policy was designed in a recent work [11]. The policy assigns a score to each pair of dispatching origin $j$ and destination $i$, which depends on the number of empty vehicles $v_j[t]$ and the distance between two stations $d_{ji}$. It will only dispatch a vehicle from origin to destination if the score $\text{score}_{ji}[t]$ is positive. Let the score defined as $\text{score}_{ji}[t] = f(v_j[t]) - d_{ji}$ and the set of score: $\text{score}_i[t] = \{\text{score}_{1i}[t], \ldots, \text{score}_{ni}[t]\}$ the algorithm will dispatch an empty vehicle to station $i$ from station $j^*$, which is defined as

$$j^*_i[t] = \begin{cases} 
\text{argmax}_{j \in \text{Neighbor}(i)} \text{score}_j[t] & \text{max}_{j \in \text{Neighbor}(i)} \text{score}_i[t] > 0 \\
0 & \text{max}_{j \in \text{Neighbor}(i)} \text{score}_i[t] \leq 0
\end{cases}.$$ 

5.4.3. Proportional

This heuristic will dispatch vehicles to stations in its neighborhood proportional to the passenger queue length $p_i[t]$ at that station. No vehicle is dispatched when the sum of passenger queue length is zero around the neighborhood. If there are extra empty vehicles at station $j$, that is, $v_j[t] - p_j[t] > 0$, then the number of empty vehicles dispatched from station $j$ to station $i$ is as follows:

$$y_{ji}[t] = \frac{(v_j[t] - p_j[t])p_i[t]}{\sum_{i \in \text{Neighbor}(j)} p_i[t]}.$$
5.4.4. CostSensitive

This real-time rebalancing policy is proposed in [8], where the number of empty vehicles dispatched every time is decided by solving a linear program. Let $M$ be the total number of vehicles in the system and the number of excess vehicle is defined as $M - \sum_{j=1}^{n} \max\{p_i[t] - v_i[t], 0\}$. Through rebalancing, this algorithm wishes to distribute excess vehicles evenly into $n$ stations, i.e. $v_i^d[t] = \lfloor (M - \sum_{j=1}^{n} \max\{p_i[t] - v_i[t], 0\})/n \rfloor$. Accordingly, the number of rebalancing vehicles from station $i$ to $j$, $y_{ij}[t]$, can be found through following linear integer programming, where $t_{ij}$ is the travel time.

\[
\begin{align*}
\text{minimize} & \quad \sum_{i,j} t_{ij} y_{ij} \\
\text{subject to} & \quad v_i[t] + \sum_{j \neq i} (y_{ji} - y_{ij}) \geq v_i^d[t], \quad \forall i \in \mathcal{N}
\end{align*}
\]

We next show through numerical simulation that the rebalancing policy obtained through PPO outperforms the rebalancing algorithms reviewed above.

![Figure 2: Policy Loss (Left), Value Function Loss (Right), and accumulated Reward per episode (Below)](image)
6. Results and Analysis

In this section, we present the results of our algorithm and compare them to the benchmarks mentioned in the last section.

6.1. Training loss

The policy loss is defined as the negative surrogate objective [9], defined in Section 4. The policy loss should approach zero from the bottom as policy gets close to the approximately optimal policy within the policy approximating class. The value function loss is the mean square error between the neural network output and the target value. As shown in Figure 2, the policy converges to a stable policy as the improvement is diminishing, and the value function neural network converges as the loss goes to zero. In the meantime, the reward function (defined in Equation 6) increases as training progresses and converges to a value with a small negative value, which indicates negative residual cost when the policy is near-optimal.

6.2. Performance Comparison

We conduct experiments on three scenarios with 600, 800 and 1000 fleet sizes. All the vehicles are equally distributed at the beginning of the simulation. We set the speed to 10 mph for all vehicles to reflect the real average traffic speed in Manhattan as suggested in [26]. Here, we introduce the metrics used to compare RL based rebalancing policy against other benchmarks. For the test case, the number of total passengers is 46377 within 10 hours when the initial random seed is set to 0, and fixed for all the test cases. The average waiting time is the period between the arrival of a passenger and his/her departure. The cost of waiting time is the product of the average waiting time and the total number of passengers. The Empty Vehicles Miles Traveled (EVMT) representing the rebalancing cost is calculated as the sum of the distance traveled by empty vehicles, which is the product of rebalancing trips and EVMT per trip. We provide relative value for the cost of waiting time and EVMT by dividing these values by those of the MaxWeight rebalancing policy.

Table 4 shows the test performance of each algorithm under 1000 vehicles in the system, where \( \alpha \) is the preference parameter defined in Equation 7. The total waiting time of MaxWeight is 25971 minutes and EVMT is 15272, which is set as the baseline for comparison. PPO(0.6) outperforms MaxWeight, BackPressure, and Proportional with both less cost of waiting and less EVMT. Compared with CostSensitive, PPO(0.2) has a better performance on both the cost of waiting and the EVMT. Therefore, our method is able to beat all the benchmarks when both metrics are considered by appropriately tuning the parameter \( \alpha \).

As shown in Figure 3, we can observe that \( \alpha \) has the ability to switch the focus of the rebalancing policy between the quality of service (waiting time) and the operational cost (EVMT). With the decrease of \( \alpha \), the algorithm achieves less waiting time at the expense of higher EVMT. This means the service provider can customize the parameter \( \alpha \) and tweak the learning objective based on their specific requirements.
Table 4: Performance comparison of various rebalancing algorithms with a fleet size of 1000 vehicles. The passenger arrival process is the same in all algorithms (since they are generated from the same initial random seed). The number of passenger arrivals is 46377 over 10 hours period. The total waiting time of MaxWeight is 25971 minutes and EVMT is 15272, which is set as the baseline for relative comparison.

| Algorithm (α) | Avg. wait time (mins) | Rebalance trips | EVMT per trip | Rel. cost wait time | Relative EVMT |
|---------------|------------------------|----------------|---------------|---------------------|---------------|
| PPO(0)        | 0.05                   | 56327          | 0.55          | 0.09                | 2.02          |
| PPO(0.2)      | 0.12                   | 51366          | 0.54          | 0.21                | 1.81          |
| PPO(0.4)      | 0.35                   | 30252          | 0.68          | 0.62                | 1.34          |
| **PPO(0.6)**  | **0.36**               | **23029**      | **0.64**      | **0.64**            | **0.96**      |
| PPO(0.8)      | 2.70                   | 21795          | 0.6           | 4.80                | 0.85          |
| PPO(1)        | 189.63                 | 919            | 0.65          | 338.62              | 0.04          |
| MaxWeight     | 0.56                   | 12417          | 1.23          | 1.00                | 1.00          |
| BackPressure  | 0.75                   | 35362          | 0.55          | 1.34                | 1.27          |
| Proportional  | 1.96                   | 36947          | 1.37          | 3.50                | 3.31          |
| CostSensitive | 0.25                   | 37197          | 1.4           | 0.45                | 3.41          |

Figure 3: A plot of relative waiting time (in log scale) and relative EVMT versus the weight α for the fleet size of 1000 vehicles. This plot is obtained through the Monte-Carlo simulation to obtain the relative waiting time and relative EVMT. The data is presented in Table 4.

Table 5 shows the performance of RL based rebalancing policies with a smaller fleet size of 800 vehicles. The number of passenger arrivals is the same with 1000 fleet case and is 46377 over 10 hours period. The total waiting time of MaxWeight is 34782 minutes and EVMT is 22807, which is set as the baseline for relative comparison. PPO(0), PPO(0.2), and PPO(0.4) outperform Maxweight, Proportional, and CostSensitive in both waiting time and EVMT. PPO(0.6) achieves better performance compared with BackPressure, considering waiting time cost and EVMT.

Table 6 shows the performance of RL based rebalancing policies with a
| Algorithm (α) | Avg. wait time (mins) | Rebalance trips | EVMT per trip | Rel. cost wait time | Relative EVMT |
|--------------|-----------------------|----------------|--------------|---------------------|--------------|
| PPO(0)      | 0.18                  | 45059          | 0.48         | 0.24                | 0.95         |
| PPO(0.2)    | 0.30                  | 37526          | 0.53         | 0.40                | 0.92         |
| **PPO(0.4)**| **0.35**              | **31768**      | **0.57**     | **0.53**            | **0.79**     |
| PPO(0.6)    | 1.62                  | 22533          | 0.67         | 2.16                | 0.66         |
| PPO(0.8)    | 2.98                  | 20318          | 0.60         | 3.97                | 0.53         |
| PPO(1)      | 190.89                | 217            | 0.49         | 253.33              | 0.00         |
| MaxWeight   | 0.75                  | 17680          | 1.29         | 1.00                | 1.00         |
| BackPressure| 2.06                  | 27860          | 0.56         | 2.74                | 0.68         |
| Proportional| 2.11                  | 26299          | 1.25         | 2.81                | 1.44         |
| CostSensitive| 1.04                 | 26413          | 1.26         | 1.39                | 1.46         |

Table 5: Performance comparison of various rebalancing algorithms with a fleet size of 800 vehicles. The passenger arrival process is the same in all algorithms (since they are generated from the same initial random seed). The number of passenger arrivals is 46377 over 10 hours period. The total waiting time of MaxWeight is 34782 minutes and EVMT is 22807, which is set as the baseline for relative comparison.

| Algorithm (α) | Avg. wait time (mins) | Rebalance trips | EVMT per trip | Rel. cost wait time | Relative EVMT |
|--------------|-----------------------|----------------|--------------|---------------------|--------------|
| PPO(0)      | 2.00                  | 17724          | 0.79         | 0.46                | 0.89         |
| PPO(0.2)    | 2.27                  | 18413          | 0.76         | 0.53                | 0.89         |
| **PPO(0.4)**| **2.20**              | **18174**      | **0.76**     | **0.51**            | **0.88**     |
| PPO(0.6)    | 2.59                  | 17255          | 0.80         | 0.60                | 0.87         |
| PPO(0.8)    | 3.19                  | 17209          | 0.79         | 0.74                | 0.86         |
| PPO(1)      | 204.30                | 257            | 0.53         | 47.62               | 0.01         |
| MaxWeight   | 4.29                  | 12989          | 1.21         | 1.00                | 1.00         |
| BackPressure| 19.92                 | 19418          | 0.53         | 4.64                | 0.65         |
| Proportional| 3.52                  | 12854          | 1.21         | 0.82                | 0.99         |
| CostSensitive| 154.06               | 2266           | 1.10         | 35.91               | 0.16         |

Table 6: Performance comparison of various rebalancing algorithms with the fleet size of 600 vehicles. The number of passenger arrivals is the same with 1000 fleet case and is 46377 over 10 hours period. The total waiting time of MaxWeight is 198957 minutes and EVMT is 15716, which is set as the baseline for relative comparison.

smaller fleet size of 600 vehicles. The number of passenger arrivals is the same with 1000 fleet case and is 46377 over 10 hours period. The total waiting time of MaxWeight is 198957 minutes and EVMT is 15716, which is set as the baseline for relative comparison. PPO(0), PPO(0.2), PPO(0.4), PPO(0.6) and PPO(0.8) outperform MaxWeight and Proportional considering both waiting time cost and EVMT. Although PPO(0.4) has 40% more rebalancing trips than MaxWeight, it reduces the average empty vehicle miles per rebalancing trip by selecting nearby stations as a rebalancing destination, which leads to 12% less EVMT compared to MaxWeight. The PPO algorithm selects the nearby stations because of the EVMT item in the reward function. With the nature...
of maximizing the reward function, the PPO algorithm will choose actions that can reduce EVMT. Therefore, nearby stations are selected since it leads to less EVMT. Moreover, the relationship between $\alpha$ and EVMT can be found in Figure 3, where a smaller $\alpha$ leads to a larger EVMT.

With the decrease of fleet size of 1000, 800 to 600, it shows that more PPO cases outperform Maxweight. Only one case (PPO(0.6)) in 1000 fleet size beats Maxweight, three cases (PPO(0), PPO(0.2) and PPO(0.4)) in 800 fleet size outperform Maxweight, five out of six cases (PPO(0), PPO(0.2), PPO(0.4), PPO(0.6) and PPO(0.8) in 600 fleet size beat Maxweight. This result shows the effectiveness of our approach, especially when the fleet sizes are small and precise rebalancing is needed.

7. Conclusion and Future Work

In this paper, we developed an RL-based vehicle rebalancing algorithm for a large-scale ridehailing system. We model the problem within a semi Markov decision problem (SMDP) framework, where passengers queue up at every station until picked up by an empty vehicle. We considered a convex combination of passenger waiting time and empty vehicle miles traveled as the objective function of the SMDP. An intelligent agent makes state-dependent empty vehicle dispatching decisions to maximize the long-term reward, which converges to an approximately optimal solution of the SMDP. The performance of the RL algorithm is compared with other rebalancing algorithms, and we found that RL outperforms other rebalancing algorithms such as MaxWeight and backpressure based rebalancing schemes. The results are encouraging especially when the fleet sizes are small.

Our future research will address the problem of determining approximately optimal rebalancing schemes through reinforcement learning to also minimize passenger drop probability when passengers are impatient and determining dynamic pricing schemes to curtail demand at stations with a high arrival rate of passengers.

References

[1] Uber Technologies, Inc., Uber announces results for fourth quarter and full year 2019, https://investor.uber.com/news-events/news/press-release-details/2020/Uber-Announces-Results-for-Fourth-Quarter-and-Full-Year-2019/ (2019).

[2] W. Yang, Q. Su, S. H. Huang, Q. Wang, Y. Zhu, M. Zhou, Simulation modeling and optimization for ambulance allocation considering spatiotemporal stochastic demand, Journal of Management Science and Engineering 4 (4) (2019) 252–265.
[3] H. Li, M. Bai, Y. Zhao, C. Dai, Vehicle flow formulation for two-echelon time-constrained vehicle routing problem, Journal of Management Science and Engineering 4 (2) (2019) 75–90.

[4] M. Pavone, S. L. Smith, E. Frazzoli, D. Rus, Robotic load balancing for mobility-on-demand systems, The International Journal of Robotics Research 31 (7) (2012) 839–854.

[5] S. L. Smith, M. Pavone, M. Schwager, E. Frazzoli, D. Rus, Rebalancing the Rebalancers: Optimally Routing Vehicles and Drivers in Mobility-on-Demand Systems, 2013 American Control Conference (2013) 2362–2367.

[6] F. Rossi, R. Zhang, Y. Hindy, M. Pavone, Routing autonomous vehicles in congested transportation networks: Structural properties and coordination algorithms, Autonomous Robots 42 (7) (2018) 1427–1442.

[7] H. R. Sayarshad, J. Y. Chow, Non-myopic relocation of idle mobility-on-demand vehicles as a dynamic location-allocation-queueing problem, Transportation Research Part E: Logistics and Transportation Review 106 (2017) 60–77.

[8] R. Zhang, M. Pavone, Control of robotic mobility-on-demand systems: a queueing-theoretical perspective, The International Journal of Robotics Research 35 (1-3) (2016) 186–203.

[9] R. Iglesias, F. Rossi, R. Zhang, M. Pavone, A BCMP network approach to modeling and controlling autonomous mobility-on-demand systems, The International Journal of Robotics Research 38 (2-3) (2019) 357–374.

[10] S. Banerjee, Y. Kanoria, P. Qian, State dependent control of closed queueing networks with application to ride-hailing, arXiv 1803.04959 (2018) 1–47.

[11] Y. Kanoria, P. Qian, Near optimal control of a ride-hailing platform via mirror backpressure, arXiv (2019) arXiv–1903.

[12] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, John Wiley & Sons, 2014.

[13] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[14] L. An, C. Ren, Z. Gu, Y. Wang, Y. Gao, Rebalancing the car-sharing system: A reinforcement learning method, in: 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC), IEEE, 2019, pp. 62–69.

[15] J. Wen, J. Zhao, P. Jaillet, Rebalancing shared mobility-on-demand systems: A reinforcement learning approach, in: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), IEEE, 2017, pp. 220–225.
[16] M. Guériau, I. Dusparic, Samod: Shared autonomous mobility-on-demand using decentralized reinforcement learning, in: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), IEEE, 2018, pp. 1558–1563.

[17] A. Henao, W. E. Marshall, The impact of ride-hailing on vehicle miles traveled, Transportation 46 (6) (2019) 2173–2194.

[18] J. D. C. Little, A proof for the queuing formula: \( L = \lambda w \), Operations Research 9 (3) (1961) 383–387. URL https://doi.org/10.1287/opre.9.3.38

[19] P. Hämäläinen, A. Babadi, X. Ma, J. Lehtinen, Ppo-cma: Proximal policy optimization with covariance matrix adaptation, arXiv preprint arXiv:1810.02541 (2018).

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International conference on machine learning, 2015, pp. 1889–1897.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).

[22] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International conference on machine learning, 2016, pp. 1928–1937.

[23] NYC Taxi & Limousine Commission, TLC Trip Record Data, https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page (2019).

[24] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[25] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, I. Stoica, Tune: A research platform for distributed model selection and training, arXiv preprint arXiv:1807.05118 (2018).

[26] NYC Department of Transportation, New York City Mobility Report, https://www.nyc.gov/html/dot/downloads/pdf/mobility-report-2018-screen-optimized.pdf (2018).
Appendix A. Data Processing

The data set used is the monthly records of yellow taxi trips in NYC. The trip records starting from 2009 to 2019 are available from the New York City Taxi and Limousine Commission website (https://www1.nyc.gov/site/tlc/index.page). To reflect the latest traffic condition in NYC, data collected from December 2019 is used as the primary data set to analyze. The total number of data points for December 2019 is roughly 7 million, where each one of the data points includes the following variables:

- **Vendor ID:** This variable indicates which vendor provided the trip data, 1 for Creative Mobile Tech., LLC and 2 for VeriFone Inc.
- **Pick-up/Drop-off time:** Time when the trip starts/ends.
- **Passenger:** Number of passengers in the trip enter by the driver, ranges from \{0, 1, ..., 9\}.
- **Trip Distance:** Miles one trip traveled.
- **Pick-up/Drop-off location ID:** The TLC taxi zone where the trip starts and ends, ranges from \{1, ..., 256\}.
- **Rate Code ID:** The type of rate for the trip, including 1 for standard rate, 2 for JFK, 3 for Newark, etc., ranges from \{1, ..., 6\}.
- **Payment Type:** Methods of payment, 1 for credit card, 2 for cash etc., ranges from \{1, ..., 6\}.
- **Fare Amount:** The time-and-distance fare calculated by the meter.
- **Total Amount:** The total amount including fare amount, tips, and other fees.

Using this primary data, the following section will cover the main steps used to filter and clean the data, before applying it to the simulator.

Appendix A.1. Data Filtering and Cleaning

Given that the coverage of the data is huge, traffic conditions could be quite different, and thus different models. The simulator considers a one hour window of morning rush time in the downtown area. More specifically, the selected trip starts within 8 am to 9 am, and has both pick-up and drop-off points within Manhattan only. The data count is reduced from 6896317 to 294597 points for a one-hour pick-up window and is further reduced to 255183 given trips within Manhattan. After carefully reading the description of each column of the original .csv file, the following criteria are developed to filter out any abnormal data.
• Trip time: 60-7200 seconds, calculated from stating and ending time provided. The trip time between 1 minute and 2 hours is required based on the size of the Manhattan area, and estimated driving time of some extreme cases tried from Google Map. The starting and ending time columns will then be deleted.

• Trip distance: 0.1-20 miles, existing variable. This range is selected using a similar approach as trip time. Some extremely long trip distance as well as negative trip distance appeared in the dataset and is considered not valid.

• Fare amount, total amount: positive value, existing variable. Zero and negative values were observed.

• Rate Code ID: 1, existing variable. Trips with standard rates are accepted, other types of trips such as voided or disputed trips are excluded.

• Passenger count, 1-6, existing variable. Exclude trips with 0 or NA passenger count.

• MTA tax, congestion surcharge, improvement surcharge: a fixed amount, existing variable. Extras are listed on the NYC website. Trip with values other than the mandated will be dropped.

• Day of the week: 0-6, calculated based on the trip dates, new variable as a factor.

• Latitude and longitude of zone center, calculated based on the zone border using gCentroid in rgeos package. Zone border file can be downloaded from the NYC website and further process can be done using rgdal package. In the original data set, only zone IDs are provided to locate pick-up and drop-off points, and the zone center GPS location is used instead in this project to replace zone IDs. Zone IDs are defined in a mysterious order and are not useful to represent geometric information, see Figure A.5 for more details.

The data count is then reduced to 237727 or a 93% pass rate. Using this cleaned data set, the number of trips versus date is shown in Figure A.4.
weekly pattern is observed for the first three weeks (Dec 1 is Sunday). The
number of trips is reduced significantly during the weekend. The pattern for
the holiday season is quite different. In general, the number of trips could be an
indicator of traffic condition, and the trip time during the holiday season could
be significantly away from the weekly pattern found in the first three weeks.
As a result, all trips after December 22 are discarded and the factor variable
indicating the day of a week is added to the data set. This brings the total data
to count down to 200761.

Appendix A.2. Passenger Interarrival Time Distribution

Another important part of the data analysis is to check passenger interarrival
distribution. The key observation from the data set is that higher demand
usually implies a better exponential distribution fit for the data. Data from
the workdays of the first three weeks in December 2019 is used for the analysis.
Demands for each zone is calculated by the total number of pick-ups from the
corrected data set, and are listed below in Table A.7 in decreasing order.

| ID  | 236 | 237 | 186 | 170 | 141 | 162 | 140 | 238 | 142 | 229 | 239 | 48  | 161 | 107 | 263 | 262 | 234 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Demand | 13243 | 11026 | 7296 | 7142 | 6762 | 6534 | 6350 | 6090 | 5669 | 5576 | 5426 | 5221 | 5173 | 5126 | 4775 | 4467 | 4106 |
| ID  | 68  | 106 | 141 | 230 | 43  | 233 | 79  | 111 | 231 | 164 | 163 | 249 | 137 | 90  | 13  | 246 | 151 |
| Demand | 3960 | 3656 | 3517 | 3657 | 3388 | 3367 | 3357 | 3208 | 3256 | 3224 | 3367 | 3044 | 2946 | 2893 | 2800 | 2639 | 2565 |
| ID  | 75  | 50  | 158 | 166 | 87  | 114 | 41  | 74  | 135 | 24  | 148 | 144 | 211 | 263 | 42  | 88  | 116 |
| Demand | 2322 | 2047 | 2020 | 1405 | 1336 | 1236 | 1231 | 1128 | 1058 | 974 | 848 | 794 | 749 | 575 | 501 | 458 | 414 |
| ID  | 234 | 4   | 244 | 209 | 152 | 232 | 45  |     |     |     |     |     |     |     |     |     |     |

Table A.7: Zone ID and the number of passengers taking a taxi between 8-9 AM on weekdays
for three weeks of December 2019.

The fitness of exponential distribution is then checked using a quantile-
quantile plot (qq-plot). To see the relationship between fitness and demand,
location ID 236, 231, 211 are chosen to represent high, medium, and low de-
mand. The qq-plots for these three locations are shown in Figures A.6 and A.7.
From the plots, we see that the qq-plots with ID 236 (high demand)
have a relatively better fit for exponential distribution except for a few points
with large values, which are mostly present in the data collected on December
20th. The mean interarrival time also has a relatively small deviation across
the weekdays of the first three weeks of December. For the medium demand
case (ID 231), the qq-plot looks similar, but the mean interarrival time for each
day has a larger deviation compared with the high demand case. For the low
demand case (ID 211), we notice that both exponential distribution and the
constant mean assumptions are violated. To simplify the simulation model, we
assume that the top 20 locations with the highest demands follow an exponen-
tial distribution with a constant rate (a lookup matrix) under the considered
scenario.

Appendix A.3. Data for Simulation

Based on the corrected data, we output two matrices to the simulator. Table
A.8 is the average interarrival matrix. The row of the matrix is the location for
pick up and the column of the matrix is the location for dropoff. The element in the ith row and jth column of the matrix is the mean interarrival time of passengers who want to travel from location i to j, in seconds. The second matrix is the average trip matrix which has the same dimension as the first one but the element in the ith row and jth column of the matrix is the mean trip time of passengers who want to have a trip from location i to j, also in seconds.
Zone IDs adopted in original data set

Zone center calculated in R

Figure A.5: Taxi zones of Manhattan, New York area.

Figure A.6: Quantile-Quantile Plot for Zone ID 236 (High demand).
Figure A.7: Quantile-Quantile Plot for Zone ID 231 (Medium demand).

Figure A.8: Quantile-Quantile Plot for Zone ID 211 (Low demand).