REAL-TIME IDLE VEHICLE RELOCATION AND DYNAMIC PRICING IN CENTRALIZED RIDE-HAILING SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Enpeng Yuan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Machine Learning in the
School of Industrial and Systems Engineering

Georgia Institute of Technology

December 2022

REAL-TIME IDLE VEHICLE RELOCATION AND DYNAMIC PRICING IN CENTRALIZED RIDE-HAILING SYSTEMS

Going beyond your limitations is far more important than staying within the limitations of what you like.

Sadhguru

For my parents Chengwu Yuan and Yonghua Lu

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Pascal Van Hentenryck, for his support and guidance throughout my graduate studies. Without him, I would not have been able to complete this work, build my strengths and, more importantly, hone my weaknesses in the process.

I would also like to express sincere gratitude to my friends and colleagues – Connor Riley, who patiently helped me get started with research in the early years, Wenbo Chen, who discussed and contributed many wonderful ideas and taught me so much along the way, Guanlin Li, who inspired my curiosity in research and who encouraged and guided me through difficult times, Kevin Dalmeijer, who honed my research and presentation skills, and many others who listened, cared, and shared my journey.

Special thanks to my parents, Yonghua Lu and Chengwu Yuan, without whom I would not have been able to pursue my dream or finish the journey. If going through a Ph.D. is my destiny, your love and support is also part of it.

The work in this thesis is partly supported by General Motors Company, NSF Award 2112533 and NSF Award 1854684.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Dynamic pricing and idle vehicle relocation are important tools for addressing demand-supply imbalance that frequently arises in the ride-hailing markets. Although interrelated, pricing and relocation have largely been studied independently in the literature. Moreover, the current mainstream methodologies, optimization and reinforcement learning (RL), suffer from significant computational limitations. The optimization needs to be solved in real-time and often trades off model fidelity (hence solution quality) for computational efficiency. Reinforcement learning requires a large number of samples to be trained offline, and often struggles to achieve full coordination among the fleet. This thesis expands the research horizon and addresses the limitations of existing approaches.

Chapter 4 designs an optimization model for computing both pricing and relocation decisions. The model ensures reasonable waiting time for the riders by reducing or postponing the demand that is beyond the service capacity. The postponement is by giving out discounts to riders who are willing to wait longer in the system, thus leveling off the peak without pricing out riders. Experiments show that the model ensures short waiting time for the riders without compromising the benefits (revenue and total rides served) of the platform. The postponement helps serve more riders during mild imbalances when there are enough vehicles to serve postponed riders after the peak.

Chapter 5 presents a machine learning framework to tackle the computational complexity of optimization-based approaches. Specifically, it replaces the optimization with an optimization-proxy: a machine learning model which predicts its optimal solutions. To tackle sparsity and high-dimensionality, the proxy first predicts the optimal solutions on the aggregated level and disaggregates the predictions via a polynomial-time transportation optimization. As a consequence, the typical NP-Hard optimization is reduced to a polynomial-time procedure of prediction and disaggregation. This allows the optimization model to be considered at higher fidelity since it can be solved and learned offline. Experiments show that the learning + optimization approach is computationally efficient and outperforms the original optimization due to its higher fidelity.

Chapter 6 extends one step further from Chapter 5, refining the optimization-proxy by reinforcement learning (RL). Specifically, RL starts from the optimization-proxy and improves its performance by interacting with the system dynamics and capturing long-term effects that are beyond the capabilities of the optimization approach. In addition, RL becomes far easier to train starting from a good initial policy. This hybrid approach is computationally efficient in both online deployment and offline training stages, and outperforms optimization and RL by combining the strengths of both approaches. It is the first Reinforcement Learning from Expert Demonstration (RLED) framework applied to the pricing and relocation problems and one of the few RL models with a fully-centralized policy.

CHAPTER 1
INTRODUCTION

The rapid growth of ride-hailing markets has transformed urban mobility, offering on-demand mobility services via mobile applications. While major ride-hailing platforms such as Uber and Didi leverage centralized dispatching algorithms to find good matching between drivers and riders, operational challenges persist due to imbalance between demand and supply. Consider morning rush hours as an example: most trips originate from residential areas to business districts where a large number of vehicles accumulate and remain idle. Relocating these vehicles back to the demand area is thus crucial to maintaining quality of service and income for the drivers. In the event that demand significantly exceeds supply, dynamic pricing is needed to ensure that the system does not admit more demand than its service capacity.

The impact of demand-supply imbalance on a ride-hailing market is highlighted by Hall et al. using Uber data in 2015 [1]. When surge pricing was deactivated during New Year's Eve, rider waiting time and driver enroute time (the time between a driver departs to pick up a request and the time the request is picked up) increased significantly and trip completion rate dropped dramatically (Figure 1.1). This phenomenon was further studied by Castillo et al. who modeled the ride-hailing system as a steady-state queuing network [2]. The authors demonstrated that, when demand exceeds supply capacity, drivers from distant areas are dispatched, resulting in long enroute time and low utilization rates (aka the Wild Goose Chase phenomenon). Several solutions have been introduced to partially mitigate the problem. Ride-sharing services (UberPool, Lyft Shared rides, etc.) enable drivers to serve multiple requests simultaneously [3, 4], increasing the service capacity of the fleet. However, it is not always sufficient to address severe imbalances. Adding more vehicles before or during the imbalance is another possibility. Bringing in new vehicles, however,

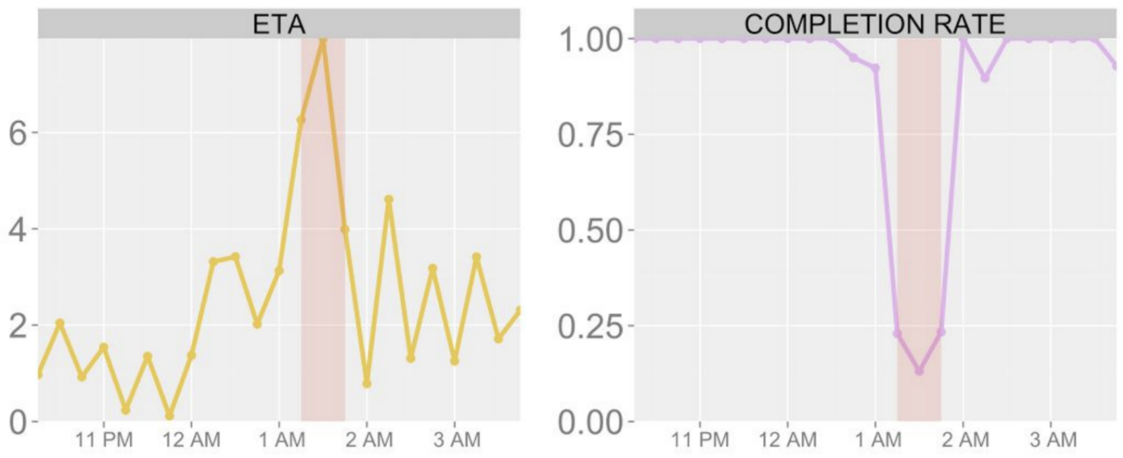


Figure 1.1: The Impact of Demand Surge on Estimated Time of Arrival (ETA) and Trip Completion Rate (Proportion of Requests Fulfilled). The Left Denotes ETA in Minutes. The Shaded Area Represents the Period When Surge Pricing Was Deactivated on New Year's Eve [1].

takes time and increases operational costs; hence it is not always feasible. Predicting the imbalance is also challenging as real-time demand depends on various factors including weather, traffic conditions, special events (concerts, sports games, conferences), and availability of other travel modes. To address the imbalance in its entirety, the ability to control the demand by pricing is indispensable.

This thesis is dedicated to developing systematic approaches to address demand-supply imbalance using idle vehicle relocation and dynamic pricing. Relocation and pricing are studied together due to their inter-dependency - where the vehicles relocate depends on demand which is shaped by the pricing decisions. They are also decided at a similar frequency in real-time (typically every 5 - 20 minutes), in contrast to matching which is decided at a much higher frequency (typically every 10 - 60 seconds).

Although there has been abundant research on pricing and relocation, there are still significant research gaps to be addressed. First, the existing literature has largely modeled pricing and relocation independently, although considering them together is far more interesting and realistic as the ride-hailing market is a dynamic interplay between demand and supply shaped by relocation and pricing. The joint modeling however, makes the problem exponentially more complex, thus requiring new methodological and computational contributions. Second, the current mainstream methodologies for pricing and relocation suffer from significant computational limitations. Existing approaches fit broadly into two categories: model-based and model-free approaches. Model-based approaches, e.g., Model Predictive Control (MPC), repeatedly solve an optimization problem over a future horizon to derive the best control decisions. Model-free approaches (predominantly Reinforcement Learning (RL)) train a state-based decision policy by interacting with the environment and observing the rewards. While both approaches have demonstrated promising performance in simulation and (in some cases) real-world deployment [5], they have evident drawbacks: the optimization needs to be solved in real time and often trades off model fidelity (hence solution quality) for computational efficiency. Reinforcement learning does not have a model but needs a large number of samples to be trained offline. Consequently, most RL models simplify the problem (e.g., by restricting relocations to nearby regions and/or limiting coordination among the fleet) to reduce computational complexity.

This thesis aims to expand the research horizon and tackle the drawbacks of these existing approaches. Chapter 4 presents an optimization model for computing both pricing and relocation decisions. The model ensures reasonable waiting time for the riders by reducing or postponing the demand that is beyond the service capacity. To resolve the computational issues of the existing approaches, Chapter 5 and Chapter 6 design a Reinforcement Learning from Expert Demonstration (RLED) framework. Chapter 5 applies machine learning (ML) to approximate a pricing and relocation optimization model (expert), and Chapter 6 refines the ML policy in Chapter 5 by a policy gradient method (RL). As a consequence, the final policy is computationally efficient, runs in polynomial-time, and requires much fewer samples to train than traditional RL. Moreover, The policy achieves better performance than optimization and RL alone by combining the strengths of both approaches.

To limit the scope of the study, this thesis focuses on a centralized ride-hailing system where drivers follow the platform's instructions exactly. Specifically, this means that

1. The drivers follow the platform's routing and relocation instructions exactly.

2. The drivers' willingness to work is not affected by the trip fares or the routing and relocation decisions.

These assumptions hold for a fleet of autonomous vehicles or human drivers who are contracted to follow the platform's decisions (they are paid a fixed rate per time, for example). Although most existing ride-hailing services employ human drivers who can choose to decline the platform's instructions, the study of a centralized system still brings tremendous insights to the system operators and may become a reality in the future with the rapid development of autonomous driving technologies.

The thesis is organized as follows. Chapter 2 covers the background material. Chapter 3 reviews the related work and summarizes the research gap. Chapter 4 presents the pricing and relocation optimization model. Chapter 5 designs a machine learning framework to imitate a pricing and relocation optimization model. Chapter 6 combines reinforcement learning with the machine learning framework in Chapter 5 to refine the imitated policy. Chapter 7 concludes the thesis and outlines directions for future work.

Part of the thesis has been published in various journals/venues. Chapter 4 has been published in [6]. Chapter 5 and part of Chapter 6 have been published in [7].

CHAPTER 2
BACKGROUND

2.1 Real-Time Ride-Hailing System

A real-time ride-hailing system has three key components: vehicle routing, idle vehicle relocation, and dynamic pricing (Figure 2.1). The vehicle routing algorithm matches requests to vehicles and chooses the vehicle routes. It operates at the individual request level with high frequency (e.g., every $10-60$ seconds). Because of the tight time constraints and the large number of requests, the routing algorithm is usually myopic, taking only the current demand and supply into account. Idle vehicle relocation and dynamic pricing, on the other hand, are forward-looking in nature. Idle vehicle relocation repositions the vehicles preemptively to anticipate demand, and dynamic pricing balances expected demand and supply in a future horizon. Vehicle relocation and pricing also take place at a lower frequency (e.g., every $5-20$ minutes).

The three components are interdependent. Take vehicle relocation as an example: where the vehicles should relocate to depends on future demand as well as how the requests will be served, which are determined by the vehicle routing and pricing algorithms. Since vehicle relocation and dynamic pricing are both forward-looking and take place at lower frequency than vehicle routing, this thesis studies them together and abstracts away the routing component.

2.2 Model Predictive Control

MPC is an online control procedure that repeatedly solves an optimization problem over a moving time window to find the best control actions. System dynamics, i.e., the interplay between demand and supply, are explicitly modeled as mathematical constraints.

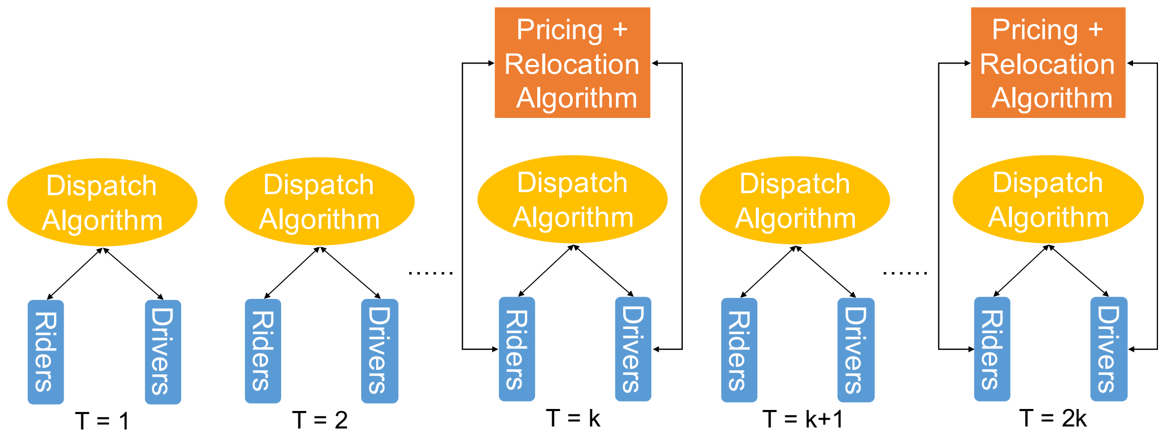


Figure 2.1: The Real-Time Ride-Hailing Operations.

Specifically, time is discretized into epochs of equal length and, during each epoch, the MPC performs three tasks: (1) it predicts state of the system for the next T epochs; (2) it optimizes decisions over these epochs; and (3) it implements the decisions of the first epoch. As time reaches the next epoch, it repeats the same procedure using information that has become available in the latest epoch (Figure 2.2). Due to real-time computational constraints, almost all the MPC models in the literature work at discrete spatial-temporal scale (dispatch area partitioned into zones, time into epochs) and use a relatively coarse granularity (see Chapter 3 for an overview of the literature).

2.3 Reinforcement Learning

Reinforcement learning, on the contrary, does not explicitly model the system dynamics. It considers the problem as an Markov decision process (MDP) which is characterized by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $r(s, a)$, a probability transition function $P(s'|s, a)$, and a discount factor $\gamma \in [0, 1]$. At each time step t in the decision horizon, the agent observes the environment state s_t, takes an action a_t, receives immediate reward $r(s_t, a_t)$, and transitions to the next state s_{t+1} according to the (unknown) transition function $P(\cdot|s_t, a_t)$ (illustrated in Figure 2.3). The goal is to find a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ or a stochastic policy $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ denotes the space of

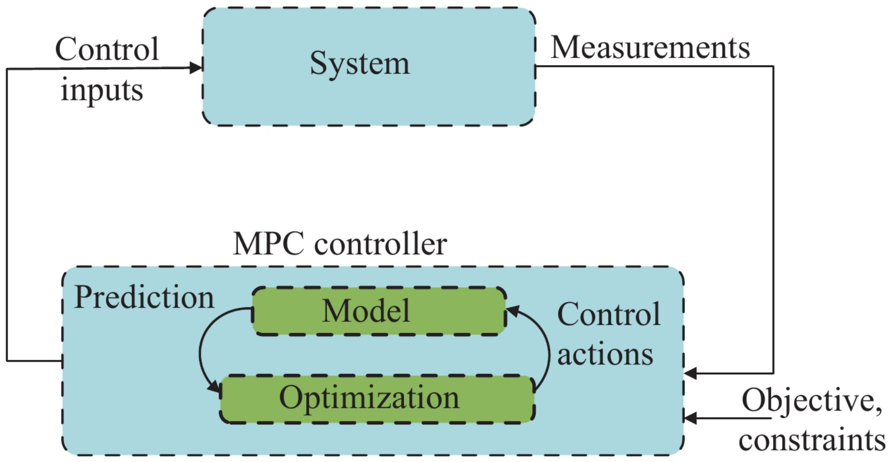


Figure 2.2: Model Predictive Control.

probability distributions over $\mathcal{A}$, to maximize total expected reward

$$J(\pi) = \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

where $\gamma \in [0,1]$ is a discount factor. Value function approximation and policy gradient are two main methods for finding π.

2.3.1 Value Function Approximation

Let action-value function

$$Q^\pi(s,a) = \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a\right]$$

denote the expected total reward starting from state s and action a under policy π. Let Q^* denote the action-value function corresponding to the optimal policy π^*. By Bellman optimality condition [8], the optimal policy satisfies

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_P\left[\max_{a' \in \mathcal{A}} Q^*(s', a')\right], \quad \forall s, s' \in \mathcal{S}, a, a' \in \mathcal{A}. \tag{2.1}$$

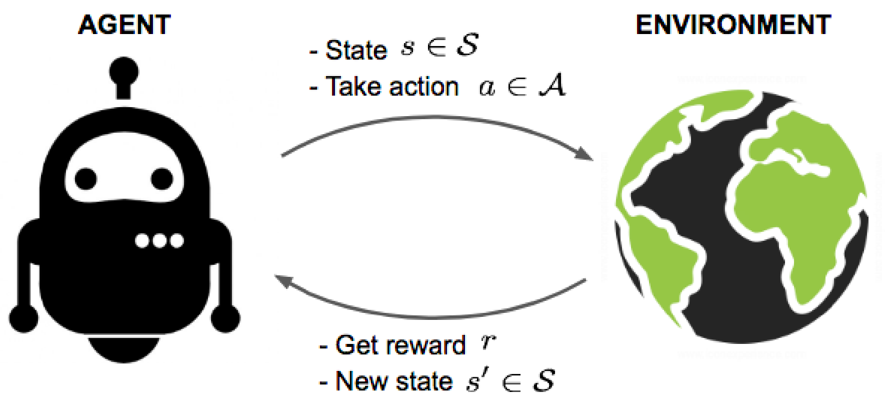

AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Figure 2.3: Reinforcement Learning.

where s' is the next state following (s, a). In practice, computing the expectation is intractable since the transition probability function P is unknown. The Q-values can instead be learned iteratively by stochastic approximation (Temporal Difference [9], SARSA [8], etc.), which is guaranteed to converge to the optimal Q-values if the state and action spaces are finite and each (s, a) is visited infinitely many times. In practice, the Q-value is often parametrized by a differentiable function (e.g., by a deep neural network (DQN)). Once the optimal Q-values are found, the optimal policy is given by $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$. Similarly, the state-value function $V^\pi(s) := \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s\right]$ can be learned to derive the optimal policy.

2.3.2 Policy Gradient

Alternatively, a policy can be found directly by policy gradient method. Let $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ be a stochastic decision policy parametrized by θ and let π be differentiable with respect to θ. By Policy Gradient Theorem [8],

$$\nabla_\theta J(\pi) = \mathbb{E}_{P,\pi}\left[G(\tau) \sum_{t=0}^{T} \nabla_\theta \log P_\pi(a_t | s_t)\right] \tag{2.2}$$

where

$$G(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t) \tag{2.3}$$

is the total (discounted) reward of the trajectory $\tau = (s_0, a_0, r(s_0, a_0)..., s_T, a_T, r(s_T, a_T))$ and $P_\pi(s_t, a_t)$ is the probability of taking a_t in s_t under policy π. In reality, computing the expectation in (2.2) is intractable since P is unknown. Instead, it can be approximated by *Monte Carlo Sampling*

$$\nabla_\theta J(\pi) \approx \frac{1}{N} \sum_{i=1}^{N} G(\tau_i) \sum_{t=0}^{T} \nabla_\theta \log P_\pi(a_{t,i}|s_{t,i}) \tag{2.4}$$

where $\tau_1, \tau_2, ..., \tau_N$ are trajectories sampled by applying π in the environment. The policy can be updated by gradient ascent

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi) \tag{2.5}$$

for a chosen stepsize α.

2.3.3 Paradigms of RL

RL can be divided into three paradigms: single-agent RL, decentralized multi-agent RL, and centralized multi-agent RL. The single-agent framework maximizes the reward of an individual agent, while the multi-agent frameworks maximize the collective rewards when there are multiple agents in the environment. In decentralized multi-agent RL, each agent follows its own policy with some extent of state/objective sharing. In centralized multi-agent RL, all the agents share their state information and plan their actions together to optimize group-level benefits.

A main challenge of RL is training complexity since the state and action spaces are typically high-dimensional (often infinite-dimensional) due to the complex system dynamics. Sampling in high-dimensional spaces makes the training computationally expensive and unstable. Another challenge is promoting coordination among a large number of agents (vehicles). Both the single-agent framework and decentralized multi-agent framework consider group-level benefits only in a limited fashion. Centralized multi-agent framework could potentially lead to full cooperation but further increases the dimensionality of the state-action space. Consequently, most work in the literature follows the single-agent or decentralized multi-agent paradigms and makes simplifying assumptions to reduce the training complexity (see Section 3.2 and Section 3.3 for details).

2.4 Multi-layer Perceptron

Multi-layer Perceptron (MLP) is the most basic form of artificial neural networks (ANN). It is composed of three types of layers: input layer, hidden layer, and output layer (Figure 2.4). The input layer contains the input data. Each hidden layer takes the previous layer's output h, performs a linear transformation Ah, and applies an element-wise nonlinear activation function σ to produce its output $\sigma(Ah)$. Commonly-used activation functions include the Sigmoid function, Rectified Linear Unit (ReLU), and hyperbolic tangent (tanh) function. The output layer takes the last hidden layer's output, performs a linear transformation, and applies a linear/nonlinear activation function to produce the final output.

The weights of the MLP (linear operators) are trained by backward propagation. Given a data input, MLP first calculates the prediction by going through the input, hidden, and output layers (a forward pass), then computes the prediction error and derives its gradient with respect to each layer's weights in a backward fashion. The weights are then updated by taking a gradient step toward the direction that reduces the error. Typically the MLP is updated based on the gradient of one or a few data points in the data set at a time (stochastic gradient descent/ batched stochastic gradient descent [10]).

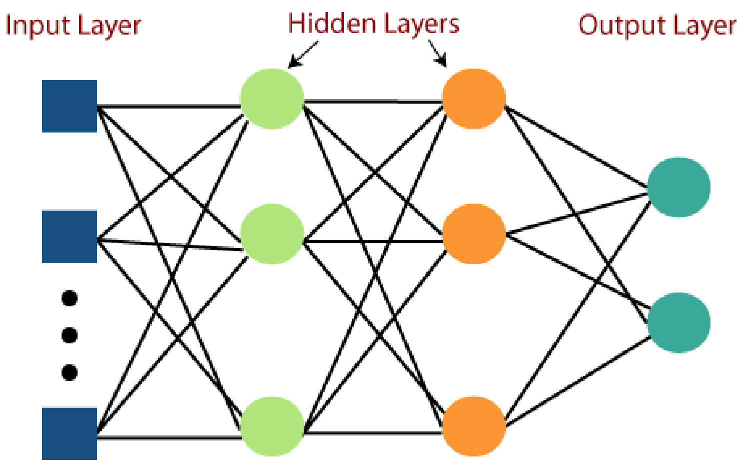


Figure 2.4: Multi-Layer Perceptron.

2.5 The Vehicle Routing Algorithm

The simulation experiments in the thesis use the routing algorithm in [4]. The algorithm batches requests into a time window and optimizes every 30 seconds. Its objective is to minimize a weighted sum of passenger waiting times and penalties for unserved requests. Each time a request is not scheduled by the routing optimization, its penalty is increased in the next time window giving the request a higher priority. The routing algorithm is solved by column generation: it iterates between solving a restricted master problem (RMP), which assigns a route (sequence of pickups and dropoffs) to each vehicle, and a pricing subproblem, which generates feasible routes for the vehicles. The RMP is depicted below.

$$\begin{aligned}
\min \quad & \sum_{r \in R} c_r y_r + \sum_{i \in P} p_i z_i && && \text{(2.6a)}\\
\text{s.t.} \quad & \left(\sum_{r \in R} y_r a_i^r \right) + z_i = 1 && \forall i \in P && \text{(2.6b)}\\
& \sum_{r \in R_v} y_r = 1 && \forall v \in V && \text{(2.6c)}\\
& z_i \in \mathbb{N} && \forall i \in P && \text{(2.6d)}\\
& y_r \in \{0,1\} && \forall r \in R && \text{(2.6e)}
\end{aligned}$$

R denotes the set of routes. V the set of vehicles, and P the set of passengers. R_v denotes the subset of feasible routes for vehicle v. A route is feasible for a vehicle if it does not exceed the vehicle capacity and does not incur too much of a detour for its passengers due to ride-sharing. c_r represents the wait times incurred by all customers served by route r. p_i is the penalty of not scheduling request i, and $a_i^r = 1$ iff request i is served by route r. Decision variable $y_r \in [0, 1]$ is 1 iff route r is selected and $z_i \in [0, 1]$ is 1 iff request i is not served by any of the selected routes. The objective function minimizes the waiting times of the served customers and the penalties for the unserved customers. Constraint (2.6b) ensures that z_i is set to 1 if request i is not served by any of the selected routes and constraint (2.6c) ensures that only one route is selected per vehicle. The column generation process terminates when the pricing subproblem cannot generate new routes to improve the solution of the RMP or the solution time limit is met.

CHAPTER 3
RELATED WORK

Prior work on real-time idle vehicle relocation and pricing problems fits broadly into two categories: *model predictive control (MPC)* and *reinforcement learning (RL).*

3.1 MPC for Relocation

The MPC optimization for the relocation problem can be formulated on either the individual vehicle/driver level or the zone level. [11] optimized the staff(driver) activities in a car-sharing network to balance demand-supply of vehicles. [12] modeled each electric vehicle's dispatching and relocation decisions while considering the battery usage. The drawback of the vehicle-level MPC is scalability since the model size increases with the number of vehicles in the system. On the other hand, MPC can be formulated on the zone level: the dispatch area is partitioned into a set of zones and relocation decisions are made on the zone-to-zone level (number of vehicles to relocate between each zone). Hence the model size only depends on the spatial partition and not the fleet size. [13] formulated a zone-level model to balance the supply-demand ratio of each zone. [14] investigated the stochastic demand setting and optimized for the worst-case scenario. [15] designed a model to minimize passenger waiting time and relocation costs and [16] extended it to the stochastic setting and solved it by sample average approximation. [17] extended [15] to the ride-sharing setting by introducing a ride-share ratio.

Although different in underlying assumptions, the zone-level MPC models follow similar formulations and the model in [15] is presented here as an illustration. Let $\mathrm{T} = \{1, 2, .., T\}$ denote the MPC model's planning horizon and N the set of zones. Decision variables x^r_{ijt} denote number of vehicles relocating from zone i to zone j at time t. Auxiliary variables x^p_{ijt} denote number of vehicles carrying passengers from zone i to zone j at time t. Auxiliary variables w_{ijt} denote number of outstanding passengers waiting to travel from zone i to zone j at time t. Auxiliary variables d_{ijt} denote the expected demand of passengers wanting to travel from i to j departing at time t that will remain unsatisfied. Input $\hat{\lambda}_{ij}$ denotes the expected number of passengers traveling from i to j at time t. τ_{ij} denote number of time periods to travel from i to j. s_{it} denote expected number of idle vehicles in zone i at time t.

$$\min \quad \sum_{i,j,t} c^r_{ijt} x^r_{ijt} + c^w_{ijt} w_{ijt} + c^d_{ijt} d_{ijt} \tag{3.1}$$

$$\text{s.t.} \quad x^p_{ijt} + d_{ijt} - w_{ijt} = \hat{\lambda}_{ij}, \qquad \forall i, j \in \mathrm{N}, t \in \mathrm{T} \tag{3.2}$$

$$\sum_{j \in \mathrm{N}} x^p_{ijt} + x^r_{ijt} = s_{it} + \sum_{j \in \mathrm{N}} x^p_{jit-\tau_{ji}} + x^r_{jit-\tau_{ji}}, \qquad \forall i \in \mathrm{N}, t \in \mathrm{T} \tag{3.3}$$

$$\sum_{t \in \mathrm{T}} w_{ijt} = \lambda_{ij0}, \qquad \forall i, j \in \mathrm{N} \tag{3.4}$$

$$x^p_{ijt}, x^r_{ijt}, w_{ijt}, d_{ijt} \in \mathbb{N}, \qquad \forall i, j \in \mathrm{N}, t \in \mathrm{T} \tag{3.5}$$

The objective minimizes relocation costs and penalty for leaving customers unserved or waiting. Constraint (3.2) and Constraint (3.3) are the passenger and vehicle continuity constraints. Constraint (3.4) ensures that all outstanding passengers at the beginning of the planning horizon are served. Similar to this model, most zone-level MPC models are mixed integer linear programs (MILP) and are difficult to solve at high spatio-temporal granularity. Consequently, they are forced to use a small number of zones or a short time horizon [13, 14, 15, 16, 17].

3.2 RL for Relocation

There have been numerous applications of RL to the vehicle relocation problem. Each vehicle is considered an agent in the environment. The state space of a vehicle typically includes the global contextual supply-demand information (number of orders and vehicles in each zone) as well as the vehicle's location and time. To reduce the dimension of the state-action space, most prior work makes the following assumptions:

1. The vehicles are considered homogeneous and share the same value function and decision policy in the single-agent and decentralized multi-agent paradigms [18, 19, 20, 21, 22, 23, 5].
2. The vehicle's relocation destination is restricted to neighboring zones that can be reached before the next decision epoch [18, 19, 20, 21, 22, 5].

3.2.1 Single-Agent RL

[18] trained a Deep-Q Network (DQN) based on the trajectory of an individual vehicle (agent). The reward function considers reduction in rider wait time (compared to no relocation) as well as relocation cost. Once the DQN is trained, it is used to guide the relocation movements of all the vehicles. Consequently, there is no coordination among the fleet since each vehicle is optimizing its own performance. Simulation studies show that this approach reduces rider waiting time compared to no relocation, but the performance is inferior to an optimization baseline.

3.2.2 Decentralized Multi-Agent RL

The decentralized multi-agent approach trains a shared decision policy for the vehicles using trajectories of the whole fleet. [20] proposed a scheme where vehicles make decisions sequentially ordered by the time they become idle, and each vehicle incorporates the previous vehicles' decisions into its state information. This prompts coordination among the agents but in a limited fashion. In [22], vehicles take actions simultaneously. The reward is computed as the average profit of all the vehicles arriving at the same relocation destination to promote coordination. This approach outperforms the single agent approach in the experiments. [23] applied centralized planning on top of the decentralized training. To find the best action for each vehicle, a centralized linear program involving the Q-values of

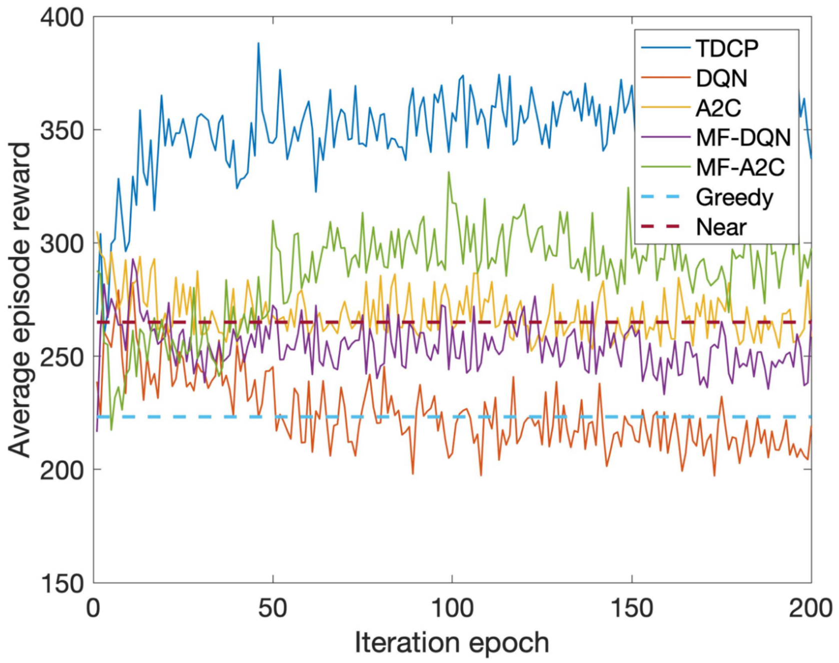


Figure 3.1: The Comparison of Different RL Paradigms. TDCP is the Decentralized Approach with Centralized Planning. DQN and A2C are Single-Agent Approaches. MF-DQN and MF-A2C are Decentralized Multi-Agent Approaches. Greedy and Near are Heuristics. The Figure is from [23].

individual vehicles is formulated and solved via value function decomposition. Due to the centralized action planning, this approach achieves better performance than the decentralized frameworks as well as the single-agent framework, as highlighted in Figure 3.1.

3.2.3 Centralized Multi-Agent RL

[24] is the only paper using a fully centralized formulation. It models each zone instead of a vehicle as an agent and employs the policy gradient method (A2C) to learn the *zone-to-zone level* dispatching and relocation decisions. The state contains supply and demand information in each zone and the action decides number of vehicles to relocate between each zone. Due to the high dimensionality of the state and action spaces, the trained algorithm was demonstrated in a simplified setting with a small number of zones.

3.3 Pricing

In contrast to the extensive literature on real-time vehicle relocation, most literature on pricing focuses on long-term market equilibrium and is not developed for a real-time setting [25, 26, 27, 28, 29, 30]. To the best of the author's knowledge, the few papers on real-time pricing are all *decentralized RL* approaches. [31] considered a profit-maximizing firm offering single and shared trip services. The company posts a price for each mode whenever a new request arises, and the riders choose one of the modes or reject the ride. The problem is solved by policy parameterisation. [32] modeled both the dispatching and pricing decisions of the platform. The dispatching solves a bipartite matching problem using state-value functions of the vehicles while the pricing selects a surge multiplier whenever a new request arrives. The state-value functions are trained by temporal-difference (TD) learning and the pricing is modeled as a contextual bandit. The two components are trained in a mutual bootstrapping manner. [33] modeled the interplay between path-based pricing and customer's choice (whether to use the service and if so, which path to take) as a Stackelberg leader-follower game. The original multi-period bilevel program was recast as an MDP problem via KKT conditions and solved by value function approximation. Experiments show that the model earns more revenue than a baseline model that does not change the price. *Of the three papers, only [33] considered both the pricing and relocation decisions, and none used a centralized RL formulation.*

3.4 Research Gap

In spite of the vast amount of existing literature, there is still significant research gap that needs to be explored. First, most work has focused on vehicle relocation and few has focused on pricing or the combination of the two, even though they are interrelated. Second, the two mainstream methodologies, optimization and RL, suffer from obvious computational drawbacks. Optimization models need to be solved in real-time and often trades off model fidelity (hence quality of solutions) for computational efficiency. RL suffers from curse of dimensionality and is difficult to achieve full coordination among the fleet. Third, the developments of optimization and RL methods have largely been disjoint. Although the two methodologies seem fundamentally different - one is model-based and the other model-free; one is data-independent and the other data-dependent - combining the two could yield immense benefits as they complement each other. This thesis address the research gap by exploring these directions.

CHAPTER 4
REAL-TIME PRICING AND RELOCATION OPTIMIZATION FOR QUALITY OF SERVICE

Transportation Network Companies (TNCs) like Uber and Lyft have fundamentally transformed mobility in many cities, providing new mobility options for a range of customers. Meanwhile, they are also facing many operational challenges. This work considers one of these critical challenges: *how to address an imbalance between demand and supply due to a surge in the number of requests*. When such an imbalance is present, riders often experience long waiting times. This hurts both the platform and riders: potential riders incur an opportunity cost by waiting and the platform receives poor customer reviews. Idle vehicle relocation (e.g., [15, 34, 35, 36, 17]) is one way to alleviate this issue: empty vehicles are relocated preemptively to places where they will be most needed to reduce waiting times. Another way to tackle the imbalance is to build mobility systems that utilize ride-sharing systematically. A study by Alonso-Mora et al. showed that systematic ride-sharing may significantly reduce the number of vehicles needed to serve requests [3]. Their results indicate that 98% of the historic demand for taxi services in NYC could be served with a much smaller taxi fleet while maintaining short wait times. However, vehicle relocation and ride-sharing on their own are not always sufficient to address the imbalance. Consider a state-of-the-art ride-sharing framework A-RTRS when applied to a 90-minute Yellow Taxi instance in New York City [17, 37]. A-RTRS routes and dispatches vehicles every 30 seconds, and it uses a model-predictive control algorithm to perform vehicle relocation every 5 minutes. Table 4.1 reports the average waiting times when a 30-minute peak with 17%, 24%, 31%, and 38% more requests is inserted into the instance. The results show that average waiting times rise dramatically as the peak becomes stronger.

Pricing is also commonly used to address demand and supply imbalances. For instance,

Table 4.1: Waiting Times in Minutes Under Various Demand Peaks

	Peak Demand Percentage			
	17%	24%	31%	38%
Avg Waiting Time	7.86	10.70	13.93	15.34
Standard Deviation	3.75	6.30	8.24	8.45

Uber's surge pricing and Lyft's Prime time pricing both raise prices to curtail excessive demand. While this may upset some customers, empirical evidence shows that it is effective at restoring demand-supply balance in the market [1]. It is not clear, however, how these pricing schemes affect average waiting times and whether they introduce unfairness in quality of service for various regions.

This work is a first step in understanding the relationship between demand and supply imbalances, pricing, average waiting times, and geographical quality of service. It extends A-RTRS with a real-time spatio-temporal pricing mechanism to restore service quality during peak times. The goal of the proposed framework, called AP-RTRS, is to decrease or postpone the demand to ensure that each rider is served within a reasonable amount of time and, ideally, that there are no significant regional differences in quality of service. More specifically, AP-RTRS features a novel optimization model for its Model Predictive Control (MPC) component that jointly optimizes price and relocation over time. The AP-RTRS framework makes three key contributions. First, while extensive work explores the effect of pricing in the long term (e.g., at market equilibrium), very few studies examine its real-time consequences when demand fluctuates both spatially and temporally. This work thus provides a detailed analysis of the real-time effects of pricing when integrated into an end-to-end ride-sharing framework. Second, while many papers study trip throughput rate, revenue, and buyer/seller surplus, to our knowledge, AP-RTRS is the first pricing framework that focuses on controlling waiting times and request completion rate. Third, AP-RTRS includes a novel dynamic pricing mechanism that also encourages riders to use the service at a later time through discounting. Discounting is appealing as it enables AP-

RTRS to level small peaks off without pricing out customers. These benefits of AP-RTRS are demonstrated on large-scale NYC taxi instances.

4.1 Problem Definition

Operating a real-time ride-sharing system requires the solving of large-scale dial-a-ride and pricing problems. Each request corresponds to a trip for a number of riders from an origin to a destination that must take place after a specified pickup time. In addition, riders are only willing to wait for a certain period of time after which they seek another mode of transportation. The goal of AP-RTRS is to regulate demand with pricing so that all riders choosing to use the service are picked up in a given time span. The platforms studied in this work either use a *fixed* fleet of autonomous vehicles or their own pool of drivers who follow the platform's instructions exactly. The system can thus relocate the vehicles at will in order to anticipate demand. It is assumed that significant historical data is available and can be used to forecast demand.

4.2 Prior Work

Among the vast literature on dynamic pricing, many papers study optimality of different pricing policies and their impact on revenue and trip throughput rates. However, most of the discussions are restricted to a simplified setting where demand is homogeneous over time (e.g., [25, 27, 28, 26, 29]), or both time and space (e.g., [2, 30]). In comparison, relatively few papers focus on a real-time setting where demand fluctuates both spatially and temporally (reviewed in Chapter 3). Although these few papers are developed in a real-time setting, there is no explicit control of the waiting times or the request completion rate. *The MPC optimization proposed in this work guarantees service quality, distinguishing it from prior work.* The most related work is the A-RTRS framework of [17] where an MPC model is developed to relocate idle vehicles in real-time. AP-RTRS replaces the MPC component of A-RTRS with a new optimization model that jointly decides idle vehicle relocation and pricing for each region over the MPC time horizon. Because of pricing, the demand becomes a variable in AP-RTRS. Moreover, the MPC model in AP-RTRS keeps track of waiting times to capture the behaviour of riders and discounting.

4.3 Pricing and Relocation Optimization

The MPC for pricing and relocation operates over a rolling time horizon and optimizes the decisions over a fixed time window for every epoch. Specifically, time is discretized into epochs of equal length and, during each epoch, the MPC performs three tasks: (1) it predicts the demand and supply for the next T epochs; (2) it optimizes decisions over these epochs; and (3) it implements the decisions of the first epoch. Due to potentially large number of vehicles and riders in real-time, deriving pricing and relocation decisions on the individual level is computationally challenging. The MPC operates at a coarser temporal and spatial granularity: it partitions the geographical area into zones (not necessarily of equal size or shape) and considers pricing/relocation decisions on the zone-to-zone level. The resulting model is scale-invariant with respect to the number of individual riders and vehicles.

Terminology and Notations The time when a rider connects to the platform and observes the price is called the *emerging time*. The time selected by a rider to use the service is called the *realization time*. It is either the emerging time or a later time selected by a rider due to discounting. The realization time is also when the platform starts to schedule the request. The *service time* is the time when a rider is picked up. The difference between realization time and service time is the *waiting time*. A rider not served after s epochs from her realization time is called a *dropout* (she will cancel the request and seek alternative modes of transportation). The length of one epoch is denoted by l and the set of epochs in the MPC time horizon $\mathcal{T} = \{1, ..., T\}$. The set of zones is denoted by $\mathcal{Z}$.

Price Offers and Price/Demand Feedback At each epoch $t \in \mathcal{T}$, AP-RTRS determines the *price offers* $(\bar{p}_{ijtt}, \bar{p}_{ijt(t+1)}, ..., \bar{p}_{ijtT})$ of requests between zone i and zone j for all epochs in $[t, T]$. A rider emerging at epoch t (i.e., between $[(t-1)l, tl]$) observes the price offer, and decides *whether* and *when* to use the service. If the rider decides not to use the service, she is assumed to exit the market and does not return. Otherwise, she reveals the realization time to the platform which commits to the price. For instance, a rider traveling from i to j, emerging at t, and deciding to use the service at τ will be charged $\bar{p}_{ijt\tau}$, even though the price for τ may vary between epoch $[t, \tau]$. Note that τ is the epoch when the rider starts to be scheduled and not necessarily the epoch in which she is picked up. Given a price offer $\vec{p}_{ijt} = (\bar{p}_{ijtt}, \bar{p}_{ijt(t+1)}, ..., \bar{p}_{ijtT})$, the platform can estimate the corresponding *demand pattern*, i.e., the expected demand $\vec{D}_{ijt} = (D_{ijtt}, D_{ijt(t+1)}, ..., D_{ijtT})$ between epoch $[t, \tau]$. The focus of this work is not on how to estimate the demand pattern from historical data; rather the work describes how to select the optimal demand pattern in the MPC.

Service Constraints In the MPC, vehicles only pick up riders in the same zone. Once a vehicle starts to serve riders or relocate, it must finish the trip before taking another assignment. These assumptions are chosen so that the MPC *approximates* how the underlying routing/dispatch algorithm works, but the dispatch algorithm does not necessarily obey these constraints.

4.3.1 The MPC Model Formulation

The overarching goal of the MPC model is to regulate demand such that all riders who choose to use the mobility system are served in the given waiting time. One way to meet this goal is to constrain all such riders to be served in the MPC time horizon. However, this might be impossible since those arriving near the end would have little flexibility. For this reason, the MPC focuses on serving those riders emerging in the first $T - s + 1$ epochs: these riders have at least s epochs available to be served if not postponed. *The goal of the*

MPC model is thus to choose a demand pattern for each zone and epoch to meet these service guarantees. When there are multiple (combination of) demand patterns to achieve this goal, the MPC model selects the combination serving the most people. This is the fundamental design philosophy behind the MPC model.

The optimization model is presented in Figure 4.1. In the model, i, j denote zones and t, τ, ρ epochs. The model inputs are as follows. V_{it} is the number of vehicles that will become idle in zone i during epoch t: those vehicles may be busy now but will become available in t. $\{\vec{D}_{ijt}^k\}_{k=1}^{N_{ij}}$ is the set of available demand patterns for O-D pair (i, j) at t, where N_{ij} is the number of demand patterns that can be selected. W_{ij} is ride-sharing coefficient: it represents the average number of passengers traveling from i to j that a vehicle typically carries accounting for the fact that a request may have multiple passengers and that a vehicle may pick up multiple requests. η_{ij} is the travel time from i to j in seconds, and λ_{ij} is the same travel time but in epochs. Inputs η_{ij} and λ_{ij} depend on traffic conditions and can be estimated in real-time. In addition, the optimization model uses $q^p(t, \tau, \rho)$ to weight a customer served at ρ who emerges at t and decides to use the service at τ. This weight is chosen to drive the model to serve people as early as possible - q^p should be decreasing in ρ. Riders who emerge and realize early should carry larger rewards since uncertainty about the future grows over time. $q^r(t)$ is the per-second relocation penalty for epoch t and should be decreasing in t for the same reason.

Decision variable p_{ijt}^k captures the pricing decision: it is a binary variable indicating whether demand pattern k, $\vec{D}_{ijt}^k$, of O-D pair (i, j) and epoch t is selected. Decision variable x_{ijt}^r captures the other important decision: it denotes the number of vehicles starting to relocate from i to j during epoch t. Auxiliary variable $x_{ijt\tau\rho}^p$ denotes the number of vehicles that start to serve riders from i to j in ρ who emerge at t and decide to use the service in τ. Auxiliary variable $v_{ijt\tau}$ denotes the number of vehicles needed to serve all expected passengers traveling from i to j who emerge in t and decide to use the service in τ. The model only implements the first epoch's decisions p_{ij1} and x_{ij1}^r; the other variables serve to

$$
\begin{aligned}
\max \quad & \sum_{i,j}\sum_{t,\tau,\rho} q^p(t,\tau,\rho) W_{ij} x^p_{ijt\tau\rho} - \sum_{i,j}\sum_{t} q^r(t)\eta_{ij} x^r_{ijt} && \\
\text{s.t.} \quad & \sum_{k=1}^{N_{ij}} p^k_{ijt} = 1 && \forall i,j,t && \text{(4.1a)} \\
& v_{ijt\tau} = \sum_{k=1}^{N_{ij}} D^k_{ijt\tau} p^k_{ijt} && \forall i,j,t,\tau && \text{(4.1b)} \\
& \sum_{\rho} x^p_{ijt\tau\rho} = v_{ijt\tau} && \forall i,j,t \leq T-s+1,\tau && \text{(4.1c)} \\
& \sum_{\rho} x^p_{ijt\tau\rho} \leq v_{ijt\tau} && \forall i,j,t > T-s+1,\tau && \text{(4.1d)} \\
& \sum_{j,t_e,\tau} x^p_{ijt_e\tau t} + \sum_{j} x^r_{ijt} = \sum_{j,t_e,\tau} x^p_{jit_e\tau(t-\lambda_{ji})} + \sum_{j} x^r_{ji(t-\lambda_{ji})} + V_{it} && \forall i,t && \text{(4.1e)} \\
& x^p_{ijt\tau\rho}, x^r_{ijt}, v_{ijt\tau} \in \mathbb{Z}_+ && \forall i,j,t,\tau,\rho && \text{(4.1f)} \\
& p^k_{ijt} \in \{0,1\} && \forall i,j,t && \text{(4.1g)}
\end{aligned}
$$

Figure 4.1: The MPC Optimization with Pricing and Relocation.

provide an approximation of the future.

It is important to mention that the variables are only defined for a subset of the subscripts given that riders drop out if not served in reasonable time. In particular, the valid subscripts for variables $x^p_{ijt\tau\rho}$ must satisfy the constraint $1 \leq t \leq \tau \leq \rho \leq \min(T, \tau + s - 1)$. Similar considerations apply to $v_{ijt\tau}$. These conditions are implicit in the model for simplicity.

The model is a mixed integer linear program (MILP). Its objective maximizes the weighted sum of customers served and minimizes the relocation cost. Constraint (4.1a) ensures that the model selects exactly one price offer (and hence one demand pattern) for each O-D pair and epoch. Constraint (4.1b) derives the number of vehicles needed to serve the demand from i to j emerging at t and realized at τ as a function of the price selected (captured by variable p^k_{ijt}). Constraint (4.1c) enforces the service guarantees: it makes sure that passengers emerging in the first $(T - s + 1)$ epochs are served in the time horizon, regardless of their realization time. Constraint (4.1d) makes sure that served demand does not exceed the true demand. Constraint (4.1e) is the flow balance constraint for each zone and epoch: it makes sure that, at every epoch t, the number of outgoing vehicles in zone i is equal to the number of incoming vehicles plus the number of idle vehicles. Note that vehicles need to depart from zone j in $t - \lambda_{ji}$ to arrive at zone i in t. Constraint (4.1f) and Constraint (4.1g) specify range of the variables. The model is always feasible when demand $v_{ijt\tau}$ can be reduced to 0.

Discussion The formulation assumes that the platform can postpone customers to any epoch within the time horizon. If the horizon is long, the platform may choose to postpone only a few epochs by restricting the range of τ to $[t, t+u]$ in the variables $x^{p}_{ijt\tau\rho}$ and $u_{ijt\tau\rho}$, where u is the maximum number of epochs a rider can be postponed. The case $u = 0$ corresponds to surge pricing which only assigns a price to the current epoch. The learning frameworks in Chapter 5 and Chapter 6 work with the surge pricing setting. Another important consideration is model complexity. In theory, longer time window and more demand patterns enable finer control and yield better results. However, they also increase the size of the model which needs to be solved in real-time. The user thus needs to strike a balance between model fidelity and computational efficiency in practice.

4.4 Simulation Study

The performance of AP-RTRS is evaluated using Yellow Taxi trip data in Manhattan, New York City [37]. The Manhattan area is partitioned into a grid of cells of 200 squared meter, and each cell represents a pickup/dropoff location. Travel times between the cells are queried from [38]. The fleet is fixed to be 1500 vehicles with capacity 4, distributed randomly among the cells at the beginning of the simulation. The test data is generated based on Yellow Taxi trip data on $12/30/2015$, 7:00am to 8:00am, scaled up proportionally to the number of requests between each Origin-Destination (O-D) cells to contain on average

1400 requests per 5 minutes. Peaks of various kinds are inserted into the instance as discussed later on, in order to test the AP-RTRS's ability to enforce the service guarantees and serve riders in reasonable time.

Simulation Environment. The end-to-end simulation to evaluate AP-RTRS is based on [17]. It has two modules: the vehicle routing algorithm reviewed in Section 2.5 and an MPC module for pricing and relocation. The vehicle routing algorithm batches riders into a time window and optimizes every 30 seconds. The MPC module has two components: the pricing/relocation model and a vehicle assignment model. The pricing/relocation model (Figure 4.1) is run every 5 minutes and given 30 seconds of solving time (it needs to output relocation decisions before the next dispatch). The pricing decisions are implemented in terms of percentages instead of absolute magnitude. For example, if the model decides to postpone 20 expected customers out of 50, the simulation will randomly select 40% *observed* requests in the next epoch to be postponed. The reason is that the model works with predicted demand and there may not be 20 customers in the next epoch. The relocation decisions are implemented by the vehicle assignment optimization, which is run immediately after the pricing/relocation model. It determines which actual vehicles to relocate by minimizing total traveling distances [17]. Each request in the simulation is given a maximum scheduling time of 5 minutes and a maximum waiting time of 15 minutes. A request for which one of these deadlines is not met is considered a *dropout* and removed from the simulation.

Configuration of The Pricing Model. The Manhattan area is partitioned into $|\mathcal{Z}| = 73$ zones and each cell is assigned to the closest zone. The travel times λ_{ij} (in epochs) between the zones are computed by averaging travel times between all cell pairs in the two zones. Time is discretized into epochs of $l = 5$ minutes. The pricing model is run every 5 minutes and has a time horizon of $T = 4$ epochs. Demand predictions for each O-D pair in each epoch is generated by adding white noise to the true demand. The white noise is normally distributed with zero mean and a standard deviation equal to 2.5% of the true demand. The forecasting module is agnostic about the peak until the second epoch of the peak and agnostic about the end of the peak until the second epoch after the peak. This captures the fact that peaks cannot always be anticipated. The number of idle vehicles in each epoch is estimated by the simulator based on current route of each individual vehicle and the travel times. Ride-sharing ratio is set to be $W_{ij} = 1.4$ for all $i, j \in \mathcal{Z}$. Service weight and relocation penalty functions are as follows: $q^p(t, \tau, \rho) = 0.5^t 0.75^{\tau - t} 0.67^{\rho - \tau}$, and $q^r(t) = 0.001 * 0.5^t$. The model is solved (optimally or near optimally) by Gurobi 9.0 in 30 seconds with 32 cores of 2.1 GHz Intel Skylake Xeon CPU [39].

Tested Models. The experiments compare three MPC models: RELOCATION, SURGE, and SURGE+POSTPONE. RELOCATION is the baseline and has no pricing component: it implements the formulation in Figure 4.1 with demand fixed to the predicted demand and without requiring that the demand emerging in the first $T - s + 1$ epochs be served. SURGE is the MPC optimization where price is only determined for one epoch. This is similar to the strategy adopted by TNCs in practice where the platform increases the price for the current epoch and customers either take the ride or leave the platform. Five demand patterns $\{D^0_{ijt}, 0.9D^0_{ijt}, 0.8D^0_{ijt}, 0.5D^0_{ijt}, 0\}$ are available for each O-D pair (i, j) and epoch t, where D^0_{ijt} is the predicted demand between i and j in epoch t under the base price. The factors $\{1.0, 0.9, 0.8, 0.5, 0.0\}$ are demand multipliers decided at the zone level, i.e., they act on the demand for O-D pairs with the same origin and emerging epoch. In other words, the demand multiplier is based on the trip origin only and does not discriminate against destinations. SURGE+POSTPONE adds to these demand patterns the option of discounting prices in future epochs to postpone riders. The experiments assume that $20\%/30\%/40\%$ of the riders can be postponed for 2 or 3 epochs for each O-D pair (i, j) and epoch t. The demand pattern is also decided at the zone level.

Table 4.2: Quality of Service Statistics for Long Peaks (No Ridesharing).

Peak Demand	**Passengers Served**			**Dropout Percentage**		
Percentage	Reloc	Surge	Post	Reloc	Surge	Post
0%	25664	25153	25196	0.0	0.0	0.0
17%	25237	25484	25470	11.1	0.0	0.0
24%	25176	25297	25346	13.8	0.0	0.0
31%	25640	25078	25188	15.4	0.0	0.0
38%	26172	25094	25154	17.0	0.0	0.0

Table 4.3: Quality of Service Statistics for Long Peaks Cont. (No Ridesharing).

Peak Demand	**Wait Time Avg (mins)**			**Relocation**		
Percentage	Reloc	Surge	Post	Reloc	Surge	Post
0%	4.1	3.5	3.5	2510	2852	2863
17%	5.8	4.3	4.3	1670	2117	2152
24%	6.1	4.2	4.3	1638	2264	2158
31%	5.5	4.1	4.3	1860	2363	2270
38%	6.3	4.2	4.2	1800	2514	2513

4.4.1 Long Peaks

Consider the case where a 30-minute peak is inserted into a single zone to simulate a demand surge after a special event such as a sports game or a concert. The experiments consider instances with four different peaks that contain 17%/24%/31%/38% more requests respectively. Table 4.2 and Table 4.3 report the dropout rate, the number of riders served, the waiting times, and the number of relocations. RELOCATION sees an increasing number of dropouts as the peaks become stronger, with 17% dropping out in the largest instance. SURGE and SURGE+POSTPONE, on the other hand, exhibit a zero dropout rate, while serving approximately the same number of riders. The pricing models also achieve lower waiting time. They perform more relocations than RELOCATION, most likely because there are fewer requests waiting to be scheduled, giving more opportunities for vehicles to relocate. Table 4.4 shows that postponed riders are served quickly, meeting the quality of service goals of the platform. *Overall, these results show that the pricing MPCs provide service quality guarantees.*

Table 4.4: Quality of Service for Postponed Riders.

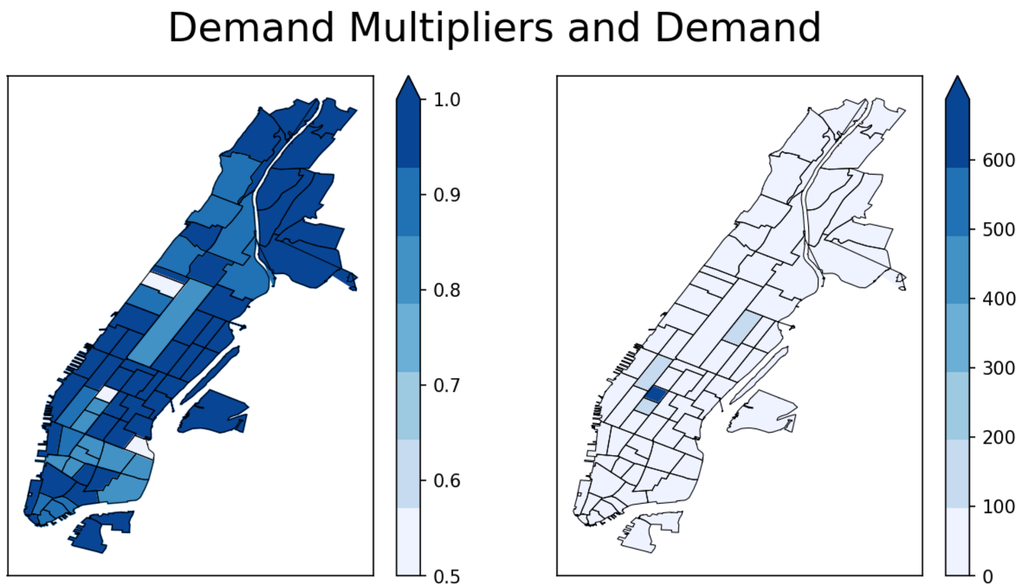
Statistics	**Peak Demand Percentage**			
	17%	24%	31%	38%
Percentage Served (%)	100	100	100	100
Wait Time Avg	5.4	5.3	5.1	5.2
Wait Time Std	1.3	1.1	1.4	1.2

Table 4.5: Revenues for Long Peaks (No Ridesharing).

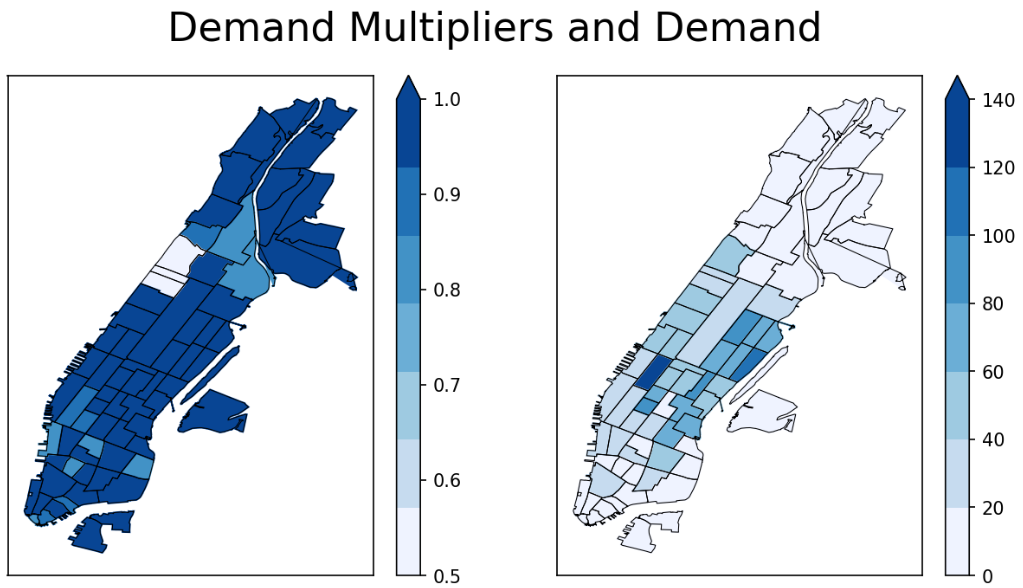
Peak Demand	**Revenue**				
Percentage	Reloc	Surge (-0.5)	Surge (-1.0)	Surge (-2.0)	Post
0%	74972	75808	74564	73942	73200
17%	73994	83245	78807	76587	75648
24%	74123	84890	79599	76953	76712
31%	74875	83360	78306	75779	75057
38%	76021	82039	77261	74871	74394

It is important to compare the revenues of the MPCs, which are computed based on price elasticity of demand. The price p^0_{ijt} that corresponds to D^0_{ijt} is computed in a standard way using travel times [40]. The prices for the other demand patterns are derived from the price elasticity of demand $\epsilon = \frac{\Delta D\%}{\Delta p\%} = \frac{(D^k_{ijt} - D^0_{ijt})/D^0_{ijt}}{(p^k_{ijt} - p^0_{ijt})/p^0_{ijt}}$. The model evaluation considers three elasticity levels: -0.5, -1.0, and -2.0 for SURGE. SURGE+POSTPONE assumes an elasticity of -1.0 for its surge component and offers an $x\%$ discount in order to postpone $x\%$ of the demand, for $x \in \{20, 30, 40\}$. Table 4.5 presents the revenue results: the revenues are noticeably higher for SURGE than for RELOCATION when $\epsilon \in \{-0.5 - 1.0\}$ and about the same when $\epsilon = -2.0$. The SURGE+POSTPONE's revenues are about the same as RELOCATION's and not as high as SURGE's revenues, since giving a discount to some riders is typically not as profitable as charging everyone a higher price, although this depends on the exact price-demand relationships. *Overall, the results show that the pricing MPC provides service guarantees without sacrificing revenues.*

To understand which riders are priced out, Figure 4.2 displays the proportions of demand kept by SURGE (demand multipliers) and the original demand for two epochs: one



(a) During Peak



(b) After Peak

Figure 4.2: Demand Multipliers and Demand.

during the peak and one after the peak. In both cases, the regions where people are priced out are spread out and are not solely concentrated in the low-demand or remote regions. *This shows that the pricing decisions are not biased with regard to demand or geographical locations.*

The above experiments were conducted without ridesharing but similar results were

Table 4.6: Number of Passengers Priced Out.

Peak Demand Percentage	**Model**			
	Rideshare		**No Rideshare**	
	Surge	Post	Surge	Post
17%	827	675	1423	1205
24%	1161	839	1836	1761
31%	1704	1721	2078	2101
38%	1789	1835	2235	2200

obtained when the platform used ridesharing. The main difference is an increase in the number of served riders and revenues. For SURGE, ridesharing increases the number of riders by 2%, 3%, 4%, and 3% and the revenues by 1%, 1%, 4%, and 4% for the peak instances.

4.4.2 Short Peaks

The benefits of postponing customers did not materialize during long peaks, since the model had no ability to postpone customers for very long periods. Instead, consider short peaks of 10-minutes with $17\%/24\%/31\%/38\%$ surges in demand and assume that the forecasting algorithm can predict the start and end of the peak. The demand forecast is obtained by adding white noise with a 2% standard deviation of the true demand. Table 4.6 reports the number of riders priced out by SURGE and SURGE+POSTPONE with and without ridesharing. While both models achieve a zero dropout rate on all instances, SURGE+POSTPONE prices out fewer riders, especially during small peaks in the ridesharing setting. A possible explanation is that, when the peak is not too intense, more vehicles are available in the future to serve postponed demand. When the peak is strong or ridesharing is not available, fewer vehicles can be used to serve the postponed demand. *These observations imply that postponing is most helpful when the surge is relatively small.*

4.5 Conclusions

This chapter proposes AP-RTRS, a real-time framework for ride sharing that features a dynamic pricing scheme. To our knowledge, AP-RTRS is the first framework that provides service guarantees and ensures short waiting times for ride-sharing platforms during peak times. This is achieved by prompting people to use the service at a later time or admitting fewer customers. AP-RTRS combines a real-time dial-a-ride optimization to dispatch and route vehicles with an MPC component for pricing and relocation. Experimental results on the Yellow Taxi instances in New York City with peaks of increasing intensities demonstrate that AP-RTRS meets its performance guarantee targets while not sacrificing revenues or creating any major geographical fairness issues. The results also show that discounting is effective for mild peaks where it increases the number of riders served significantly, especially when ride-sharing is considered. Future work can focus on generalizing AP-RTRS with fairness guarantees and taking into account the supply(driver) side's response to pricing.

CHAPTER 5
LEARNING OPTIMIZATION-PROXY FOR MPC MODELS

Large-scale ride-hailing systems often combine routing at the individual request level with dynamic pricing and vehicle relocation for anticipatory demand-supply control. Existing mainstream methodologies for pricing and relocation, MPC and RL, suffer from significant computational drawbacks in either online deployment or offline training. The MPC optimization needs to be solved online and often trades off fidelity (hence quality of solutions) for computational efficiency. Reinforcement learning requires a tremendous amount of data to explore high-dimensional state-action spaces and typically simplifies the problem to ensure efficient training. While a complex real-world system like ride-hailing may never admit a perfect solution, there are certainly possibilities for improvement.

This work presents a step to overcoming these computational challenges. *Its key idea is to replace the MPC optimization with a machine-learning model that serves as the optimization proxy and predicts the MPC's optimal solutions.* The proposed approach allows ride-hailing systems to consider the MPC at higher spatial or temporal fidelity since the optimizations can be solved and learned offline. The proposed learning framework can accommodate any MPC formulation as long as the pricing decisions are on the zone-level (demand/price in each zone) and the relocation decisions are on the zone-to-zone level (number of vehicle relocations between each zone).

Learning the MPC however, comes with several challenges. First, the pricing and relocation decisions are interdependent: where the vehicles should relocate depends on the demand which is governed by price. This imposes implicit correlation among the predictions, which are hard to enforce in classic regression models. Second, the relocation decisions are of high dimensions (e.g., the number of vehicles to relocate between pairs of zones) and sparse, as relocations typically occur only between a few low-demand and high-demand regions. Capturing such patterns is difficult even with large amount of data. Third, the predicted solutions may not be feasible, as most prediction models cannot enforce the physical constraints that the solutions need to satisfy.

To solve these challenges, we design a sequential prediction framework that first predicts the pricing decisions and then the relocation decisions based on the predicted prices. Furthermore, the framework utilizes an aggregation-disaggregation procedure that predicts the decisions at an aggregated level to overcome the high dimensionality and sparsity, and then converts them back to feasible solutions on the original granularity by a polynomial-time solvable transportation optimization. *As a consequence, during real-time operations, the typical NP-Hard and computationally demanding MPC optimization is replaced by a polynomial-time problem of sequential prediction and optimization.*

To the best of the authors' knowledge, the only work that has taken a similar approach is [41], which approximates the decisions of a relocation MPC model by a recurrent neural network (RNN) and shows that the RNN performs close to the original model. However, their model does not include pricing and considers only one epoch (10 mins). This work focuses on much more sophisticated MPC models incorporating both relocation and pricing decisions over the course of multiple epochs. As a consequence, the system dynamics becomes much more complex and the model is significantly harder to learn since the solution space is exponentially larger.

The proposed learning & optimization framework is evaluated on the Yellow Taxi data set in New York City and serves 6.7% more riders than the original optimization approach due to its higher fidelity. The results suggest that *a hybrid approach combining machine learning and tractable optimization may provide an appealing avenue for certain classes of real-time problems.*

5.1 The Imitated MPC Optimization

Without loss of generality, the learning and optimization framework is illustrated with a variation of the MPC model in Chapter 4. Note, however, that the framework can accommodate other pricing and relocation models. In particular, the new MPC model has the option to reduce but not postpone riders (same as the SURGE model in Section 4.4), since rider postponement is a specific design feature while the framework is intended for general model formulations. A demand multiplier $m_{it} \in [0, 100]$ is selected for each zone and epoch, which represents the percentage of incoming demand to admit. For example, $m_{it} = 80$ means to keep 80% demand originating from zone i during epoch t. The model is given a set of available demand multipliers $\{m_{it}^k\}_{k \in K}$ for each zone and epoch, and will choose the demand multiplier that maximizes the platform's benefits while maintaining short waiting times. The relocation decisions are determined on the zone-to-zone level, e.g., number of vehicles to relocate between each pair of zones. The model formulation is presented in Table 5.1 and Figure 5.1. In the formulation, i, j denote zones and t_0, t, ρ epochs. Compared with the original model formulation (Figure 4.1), the main difference is that decision variables $x_{ijt\tau\rho}^p$ and $v_{ijt\tau}$ become $x_{ijt\rho}^p$ and v_{ijt}, dropping the time index τ denoting the epoch that the request is postponed to. Since riders only wait for s epochs before dropping out, the valid subscripts for variables $x_{ijt\rho}^p$ must satisfy the constraint $1 \leq t \leq \rho \leq \min(T, t + s - 1)$. These conditions are implicit in the model for simplicity. Furthermore, let $\phi(t) = \{\rho \in \mathcal{T} : t \leq \rho \leq t + s - 1\}$ denote the set of valid pick-up epochs for riders placing their requests in epoch t.

The objective maximizes the weighted sum of customers served and minimizes the relocation cost. Constraint (5.1a) ensures that the model selects exactly one demand multiplier for each zone and epoch. Constraint (5.1b) derives the number of vehicles needed to serve the demand as a function of the demand multiplier selected (captured by variable p_{it}^k). Constraint (5.1c) enforces the service guarantees: it makes sure that riders with requests in the

Table 5.1: The Nomenclature for the MPC Optimization.

Model Input	
V_{it}	Number of vehicles that will become idle in i during t
$\{(m^k_{it}, D^k_{ijt})\}_{k\in K}$	Set of demand multipliers and their corresponding demand from i to j during t available to be selected
λ_{ij}	Number of epochs to travel from i to j
Model Parameters	
s	Number of epochs that a rider remains in the system
W_{ij}	Average number of riders from i to j that a vehicle carries
$q^p(t,\rho)$	Weight of a rider served at ρ whose request was placed at t
$q^r_{ij}(t)$	Relocation cost between i and j in t
Decision Variables	
$p^k_{it} \in \{0,1\}$	Whether the kth demand multiplier m^k_{it} is chosen for zone i and epoch t
$x^r_{ijt} \in \mathbb{Z}_+$	Number of vehicles starting to relocate from i to j during t
Auxiliary Variables	
$v_{ijt} \in \mathbb{Z}_+$	Number of vehicles needed to serve all expected riders from i to j whose requests are placed at t
$x^p_{ijt\rho} \in \mathbb{Z}_+$	Number of vehicles that start to serve at time ρ riders going from i to j whose requests were placed at t
$l_{it} \in \{0,1\}$	Whether there is unserved demand in i at the end of epoch t

first $(T-s+1)$ epochs are served in the time horizon since they have at least s epochs to be served. Constraint (5.1d) makes sure that the served demand does not exceed the true demand. Constraint (5.1e) is the flow balance constraint for each zone and epoch. Constraint (5.1f) and Constraint (5.1g) prevent vehicles from relocating unless all the demand in the zone has been served, approximating the behavior of the routing algorithm which favors scheduling vehicles to nearby requests. Constraint (5.1h) and Constraint (5.1i) specify the range of the variables. The model is always feasible when the demand can be reduced to 0 in all zones and epochs. The model is a mixed-integer linear program (MILP), which is difficult to solve at high fidelity when the number of zones/epochs/demand multipliers is large. This is the key motivation for replacing the optimization with a more efficient machine learning model.

$$
\begin{aligned}
\max \quad & \sum_{i,j}\sum_{t,\rho} q^p(t,\rho) W_{ij} x^p_{ijt\rho} - \sum_{i,j}\sum_{t} q^r_{ij}(t) x^r_{ijt} && && \\
\text{s.t.} \quad & \sum_{k \in K} p^k_{it} = 1 && \forall i,t && (5.1a) \\
& v_{ijt} = \sum_{k \in K} D^k_{ijt} p^k_{it} && \forall i,j,t && (5.1b) \\
& \sum_{\rho} x^p_{ijt\rho} = v_{ijt} && \forall i,j,t \le T-s+1 && (5.1c) \\
& \sum_{\rho} x^p_{ijt\rho} \le v_{ijt} && \forall i,j,t > T-s+1 && (5.1d) \\
& \sum_{j,t_0} x^p_{ijt_0t} + \sum_{j} x^r_{ijt} - V_{it} = \sum_{j,t_0} x^p_{jit_0(t-\lambda_{ji})} + \sum_{j} x^r_{ji(t-\lambda_{ji})} && \forall i,t && (5.1e) \\
& \sum_{j} x^r_{ijt} \le M l_{it} && \forall i,t && (5.1f) \\
& \sum_{j} \sum_{t_0 \in \phi(t)} \left(v_{ijt_0} - \sum_{\rho=t_0}^{t} x^p_{ijt_0\rho} \right) \le M(1-l_{it}) && \forall i,t && (5.1g) \\
& x^p_{ijt\rho}, x^r_{ijt}, v_{ijt} \in \mathbb{Z}_+ && && (5.1h) \\
& p^k_{it}, l_{it} \in \{0,1\} && && (5.1i)
\end{aligned}
$$

Figure 5.1: The Imitated MPC Optimization.

5.2 The Learning Framework

The learning framework trains an optimization-proxy that predicts the actionable decisions of a pricing and relocation optimization model $\Omega : \mathcal{I} \rightarrow \mathcal{W}$, where $\mathcal{I}$ is the model input and $\mathcal{W} = \mathcal{R} \cup \mathcal{M}$ is the model's pricing decisions $\mathcal{M}$ and the relocation decisions $\mathcal{R}$. The training data can be generated by running Ω on a set of problem instances and extracting its results. *Without loss of generality, the presentation illustrates the learning methodology based on the MPC model in Section 5.1, but the methodology applies to any MPC model whose pricing decisions are on the zone-level (demand/price in each zone) and relocation decisions are on the zone-to-zone level (number of relocations between each zone).* Hence

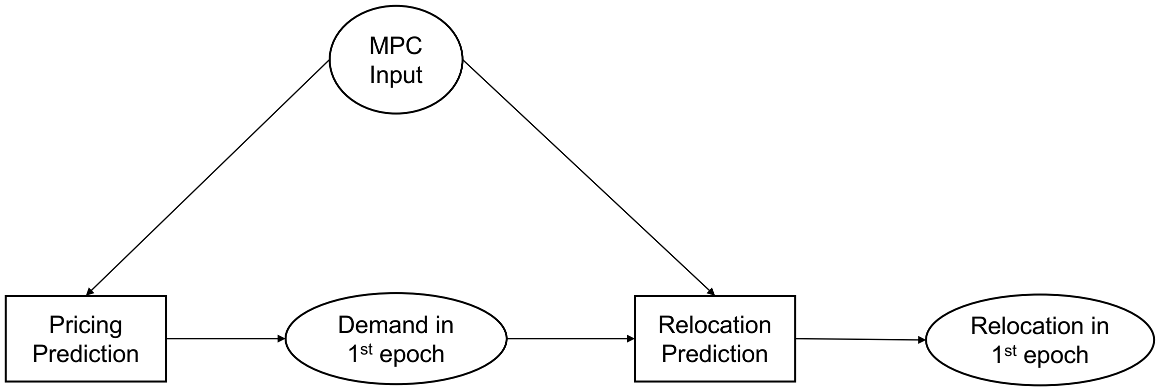


Figure 5.2: The Architecture of the Optimization-Proxy.

$|\mathcal{W}| = O(|\mathcal{Z}|^2)$, where $|\mathcal{Z}|$ is the number of zones in the dispatch area.

To account for the correlation between pricing and relocation decisions, the optimization-proxy is composed of two machine learning models - one to predict the pricing decisions and the other to predict the relocation decisions *based on the predicted demand (price)* (see Figure 5.2). The predictions of the relocation and pricing decisions are thus coupled via the sequential prediction structure.

5.2.1 The Pricing-Learning Model

The pricing-learning model $\hat{\mathcal{O}}_p : \mathcal{I} \rightarrow \mathcal{M}$ takes the MPC optimization's inputs $\mathcal{I}$ (listed in Table 5.1) and predicts the pricing decisions in the first epoch (only these decisions are actionable after each MPC execution). More precisely, the model predicts demand multipliers $\mathbf{m} = [m_{i1}]_{i \in Z}$. The predicted demand multipliers are rounded to the nearest demand multiplier to be the final pricing decisions. For example, if the set of demand multipliers is $\{m_{i1}^k\}_{k \in K} = \{100, 75, 50, 25, 0\}$ and the prediction is 80, the final prediction will be 75. The choice of the machine-learning model can be problem-dependent and the learning framework is not confined to any specific learning model.

5.2.2 The Relocation-Learning Model

The relocation-learning model $\hat{\mathcal{O}}_r : \mathcal{I}' \to \mathcal{X}$ takes as input the MPC model's input $\mathcal{I}$ as well as the predicted demand (derived from the demand multipliers $\mathbf{m}$) in the first epoch from $\hat{\mathcal{O}}_p$. The target is the first epoch's relocation decisions $\mathbf{r} = [x^r_{ij1}]_{i,j\in\mathcal{Z}}$. In reality, $\mathbf{r} \in \mathcal{R}$ is high-dimensional ($|\mathcal{R}| = |\mathcal{Z}|^2$) and sparse, since most vehicles relocate to a few high-demand zones. The high-dimensionality and sparsity impose great difficulty for learning. This chapter designs an aggregation-disaggregation procedure which predicts $\mathbf{r}$ at the aggregated (zone) level and then disaggregates the predictions via an efficient optimization procedure. More precisely, the zone-level relocation decision $\mathbf{x} \in \mathcal{X}$ is predicted and disaggregated to zone-to-zone level by an efficient optimization problem $\mathcal{TO} : \mathcal{X} \to \mathcal{R}$.

Aggregation and Prediction The zone-to-zone level decision $[x^r_{ij1}]_{i,j\in\mathcal{Z}}$ is first aggregated to, and predicted at the zone level. More specifically, two metrics are predicted for each zone i:

1. Number of vehicles relocating from i to other zones, i.e., $x^o_i := \sum_{j\in\mathcal{Z}, j\neq i} x^r_{ij1}$;
2. Number of vehicles relocating to i from other zones, i.e., $x^d_i := \sum_{j\in\mathcal{Z}, j\neq i} x^r_{ji1}$.

These two metrics can be both non-zero at the same time: an idle vehicle might be relocated from i to another zone for serving a request in the near future, and another vehicle could come to i to serve a later request. The aggregated decisions $\mathbf{x} = [x^d_i]_{i\in\mathcal{Z}} \oplus [x^d_i]_{i\in\mathcal{Z}}$, where $\oplus$ denotes vector concatenation, are then predicted by the chosen machine-learning model. This aggregation step reduces the label dimension from $|\mathcal{R}| = |\mathcal{Z}|^2$ to $|\mathcal{X}| = 2|\mathcal{Z}|$.

Disaggregation and Feasibility Restoration The predicted relocation decisions $\hat{\mathbf{x}}$ must be transformed to feasible solutions that are integer and obey flow balance constraints. This is performed in three steps:

1. $\hat{x}^o_i$ and $\hat{x}^d_i$ are rounded to their nearest non-negative integers;

2. To make sure that there are not more relocations than idle vehicles, take $\hat{x}_i^o = \min\{\hat{x}_i^o, V_{i1}\}$ where V_{i1} is expected number of idle vehicles in i in the first epoch;

3. $\hat{x}_i^o$ and $\hat{x}_i^d$ must satisfy the flow balance constraint, e.g., $\sum_{i \in \mathcal{Z}} \hat{x}_i^o = \sum_{i \in \mathcal{Z}} \hat{x}_i^d$: this is achieved by setting the two terms to be the minimum of the two, by randomly decreasing some non-zero elements of the larger term.

After a feasible relocation plan is constructed at the zone level, the disaggregation step reconstructs the zone-to-zone relocation via a transportation optimization $\mathcal{TO} : \mathcal{X} \to \mathcal{R}$. The model formulation is given below. Variable r_{ij} denotes the number of vehicles to relocate from zone i to zone j, and constant c_{ij} represents the corresponding relocation cost. The model minimizes the total relocation costs to consolidate the relocation plan. The solution r_{ij} will be implemented by the ride-hailing platform in the same way as x_{ij1}^r from the MPC. Note that r_{ii} should be 0 since $\hat{\mathbf{x}}$ denotes relocations into and out of each zone. However, the problem in that form may be infeasible. By allowing the r_{ii}'s to be positive and assigning a large value to the relocation costs c_{ii}, *the problem is always feasible, totally unimodular, and polynomial-time solvable* [42].

$$\mathcal{TO}(\hat{\mathbf{x}}) = \arg\min_{r} \quad \sum_{i,j \in \mathcal{Z}} c_{ij} r_{ij} \tag{5.2a}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{Z}} r_{ij} = \hat{x}_i^o, \qquad \forall i \in \mathcal{Z} \tag{5.2b}$$

$$\sum_{j \in \mathcal{Z}} r_{ji} = \hat{x}_i^d \qquad \forall i \in \mathcal{Z} \tag{5.2c}$$

$$r_{ij} \in \mathbb{Z}^+ \qquad \forall i, j \in \mathcal{Z} \tag{5.2d}$$

5.3 Simulation Study

The proposed learning framework is evaluated on Yellow Taxi Data in Manhattan, New York City [43]. The machine learning models, $\hat{\mathcal{O}}_r$ and $\hat{\mathcal{O}}_p$, are trained from 2017/01 to 2017/05 and tested in 2017/06. Section 5.3.1 reviews the simulation environment. Section

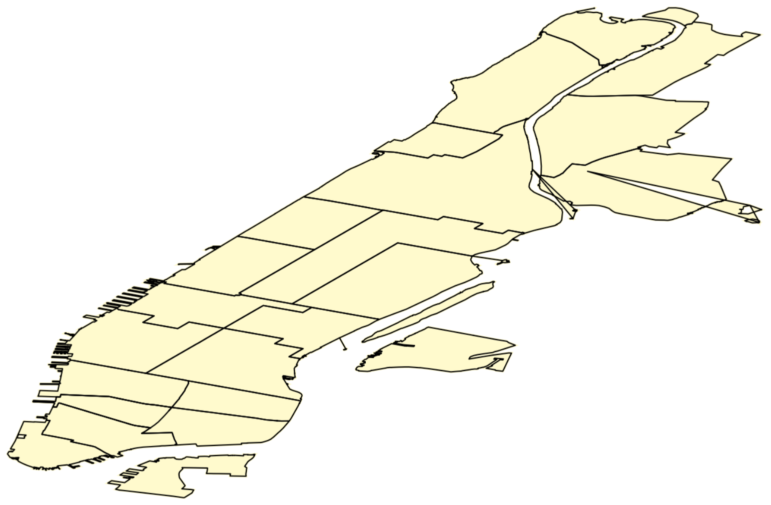


Figure 5.3: The Manhattan Area.

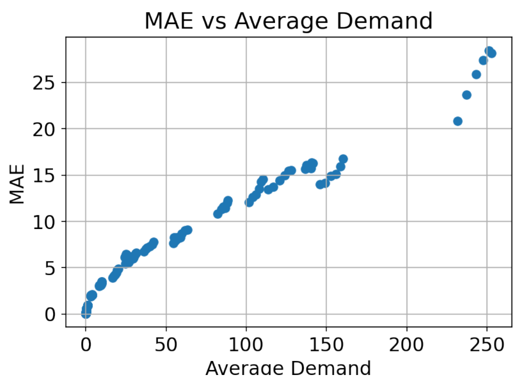
5.3.2 presents the learning results. Section 5.3.3 evaluates the performance of the machine learning policies.

5.3.1 Simulation Environment

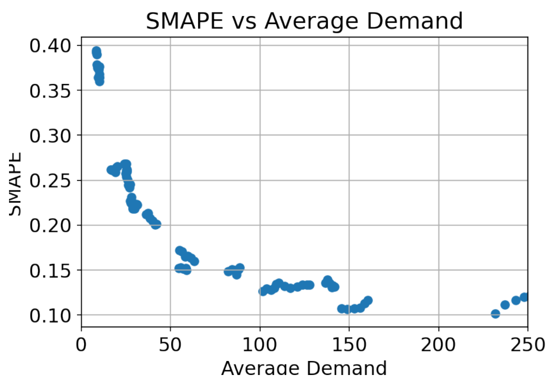
The end-to-end ride-hailing simulator in [17] is the basis of the experimental evaluation. The Manhattan area is partitioned into a grid of cells of 200 squared meter and each cell represents a pickup/dropoff location. Travel times between the cells are queried from OpenStreetMap [38]. The fleet is fixed to be 1600 vehicles with capacity 4, distributed randomly among the cells at the beginning of the simulation. Riders must be picked up in 10 minutes and matched to a vehicle in 5 minutes since their requests, after which they drop out. The routing algorithm (reviewed in Section 2.5) batches requests into a time window and optimizes every 30 seconds. The MPC component is executed every 5 minutes. It partitions the Manhattan area into 24 zones (Figure 5.3) and time into 5-minute epochs. The time window contains 6 epochs and riders can be served in 2 epochs following their requests. The number of idle vehicles in each epoch is estimated by the simulator based on the current route of each vehicle and the travel times. The ride-share ratio is $W_{ij} = 1.5$ for all $i, j \in Z$. Service weight and relocation penalty are $q^p(t,\rho) = 0.5^t 0.75^{\rho-t}$ and $q^r_{ij}(t) = 0.001 * 0.5^t \eta_{ij}$ where η_{ij} is travel time between zone i and zone j in seconds. Five demand multipliers $[100, 75, 50, 25, 0]$ are available for each zone and epoch.

The baseline demand $\mathbf{D} := [D^0_{ijt}]_{i,j \in Z, t \in \mathcal{T}}$ are forecasted and used to derive the demand under each demand multiplier - the demand from zone i to zone j at epoch t is $D^k_{ijt} = D^0_{ijt} * m^k_{it}\%$ under demand multiplier m^k_{it}. The design of demand forecasting techniques is beyond the scope of this work. We first forecast zone-level demand $D_{it} = \sum_{j \in \mathcal{Z}} D^0_{ijt}$ and then assign the destinations based on historical distribution. The reason for doing zone-level prediction is to reduce sparsity in D^0_{ijt}, since most trips travel between a few popular regions. The forecasting model is a 2-layer fully-connected neural network with $(256, 256)$ hidden units, RELU activation functions, and MSE loss with l_1-regularization. It is trained from $2017/01$ to $2017/05$ and tested in $2017/06$. The original time series data is augmented by injecting white noise at each time step to create more training data, where the white noise is sampled from a uniform distribution $U(-5, 5)$. To predict zone-level demand in the MPC horizon, the model uses the demand observed in the previous 4 epochs, as well as data observed a week ago during the same period to account for seasonality. For example, when forecasting demand from 8:00am to 8:30am (6 epochs) on $2017/06/08$, the model uses demand from 7:40am to 8:00am on $2017/06/08$ and demand from 7:40am to 8:30am on $2017/06/01$. After zone-level demand is predicted, it is assigned to zone-to-zone level based on the historical distribution of the trip's destination. For example, if μ_{ij} proportion of trips from zone i goes to zone j during the hour of the prediction and $\hat{D}_{it}$ is the demand prediction, the final zone-to-zone prediction is $\hat{D}^0_{ijt} = \hat{D}_{it} \times \mu_{ij}$ rounded to the nearest integer. The mean absolute error (MAE) and the symmetric mean absolute percentage error (SMAPE) of the zone-level forecast in $2017/06$ are displayed in Figure 5.4. The overall mean squared error of the zone-to-zone level forecast in $2017/06$ is 0.49.

The MPC's pricing decisions are implemented at the level of demand multipliers: if MPC decides to keep 50% demand in a zone, the simulation randomly keeps 50% requests



(a) MAE.



(b) SMAPE.

Figure 5.4: The Accuracy of the Demand Forecasting. Each Point Denotes the Average Demand and the MAE/SMAPE of a Zone on the Test Set.

in the current epoch and discards the rest. After the MPC decides zone-to-zone level relocations, a vehicle assignment optimization determines which individual vehicles to relocate by minimizing total traveling distances (see [17] for details).

Of the routing, MPC, and vehicle assignment models, the routing model is the most computationally intensive since it operates at the individual (driver and rider) level as opposed to the zone level. Since all three models must be executed within the 30 seconds batch window, the platform allocates 20 seconds to the routing optimization, 5 seconds to the MPC, and 5 seconds to the vehicle assignment. All the models are solved using Gurobi 9.0 with 6 cores of 2.1 GHz Intel Skylake Xeon CPU [39].

5.3.2 Training Results

The machine learning models $\hat{\mathcal{O}}_r$ and $\hat{\mathcal{O}}_p$ are trained on Yellow Taxi data between $2017/01$ and $2017/05$. Each daily instance between 7:00am and 9:00am are selected as training instances. The total number of riders in these instances ranges from 10,000 to 50,000, representing a wide variety of demand scenarios in Manhattan. The instances are perturbed by randomly adding/deleting a certain percentage of requests to generate more training instances, where the percentages are sampled from a uniform distribution $U(-5, 5)$. The instances are run by the simulator and the MPC results are extracted as training data.

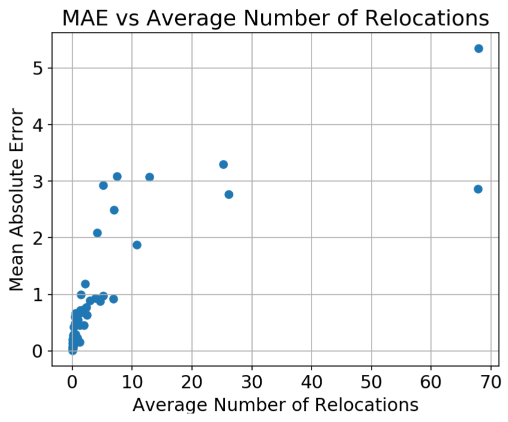
In addition to the expected demand and supply mentioned in Section 5.2.1, the pricing-learning model also uses the following input features: the difference between supply and demand in each zone during the first epoch under all demand multipliers, and the ratio between cumulative supply and cumulative baseline demand $[\tilde{V}_{it}/\tilde{D}^0_{it}]_{i\in Z, t\in \mathcal{T}}$ where $\tilde{V}_{it} = \sum_{\tau=1}^{t} V_{i\tau}$ and $\tilde{D}^0_{it} = \sum_{\tau=1}^{t}\sum_{j\in Z} D^0_{ij\tau}$. These additional features help improve the learning accuracy.

Four models were trained to learn the pricing and relocation decisions: random forest (RF), support vector regression (SVR), gradient boosting regression tree (GBRT), and deep neural network (DNN). The RF, SVR, and GBRT models were trained on 8000 data points and the DNN was trained on 35000 data points since fitting the DNN typically requires more data. All the models use the mean squared error (MSE) loss except SVR which uses epsilon-insensitive loss and l_2-regularization. The hyperparameters of each model were tuned through 5-fold cross-validation. The selected hyperparameters for the relocation models are: (kernel, regularization weight) = (radial basis function, 100), (max tree depth, number of trees) = (32, 200) for RF, (max tree depth, number of trees) = (64, 200) for GBRT, and two fully-connected hidden layers with (750, 1024) hidden units and hyperbolic tangent (tanh) activation functions for the DNN. The selected hyperparameters for the pricing models are: (kernel, regularization weight) = (radial basis function, 1000), (max tree depth, number of trees) = (64, 200) for RF, (max tree depth, number of trees) = (32, 100) for GBRT, and two fully-connected hidden layers with (750, 1024) hidden units and ReLU activation functions for DNN. The SVR, GBRT, and RF models were trained in scikit-learn package and the DNN model was trained in Pytorch by Adam optimizer with batch size 32 and learning rate 10^{-3} [44, 45, 46].

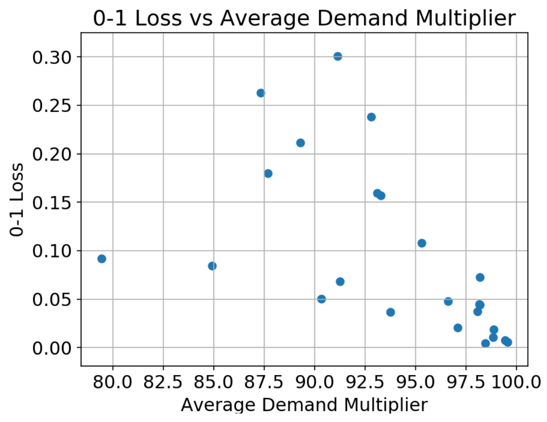
The trained models were evaluated on a held-out testing set. The predictions are rounded to feasible solutions by the procedures described in Section 5.2.1 and Section 5.2.2. The overall loss after rounding is reported in Table 5.2, where the relocation models report the mean squared error (MSE) loss and the pricing models report both the MSE and

Table 5.2: Testing Loss of the Models.

Model	Relocation (MSE)	Pricing (MSE)	Pricing (0-1 Loss(%))
SVR	21.98	100.49	13.17
RF	21.69	101.29	13.32
GBRT	28.37	88.59	12.28
DNN	**6.83**	**66.65**	**9.46**



(a) Relocation Predictions.



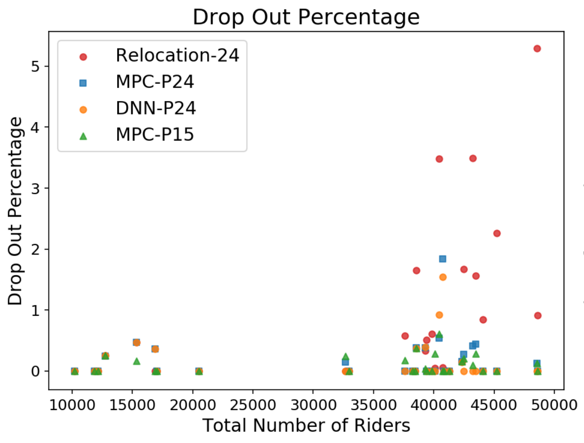
(b) Pricing Predictions.

Figure 5.5: MPC Decisions Predictions: Each Point on the Left Denotes the Average Number of Relocations and the MAE of Relocation Predictions for a Zone. Each Point on the Right Denotes the Average Demand Multiplier and the 0-1 Loss of the Pricing Prediction of a Zone.

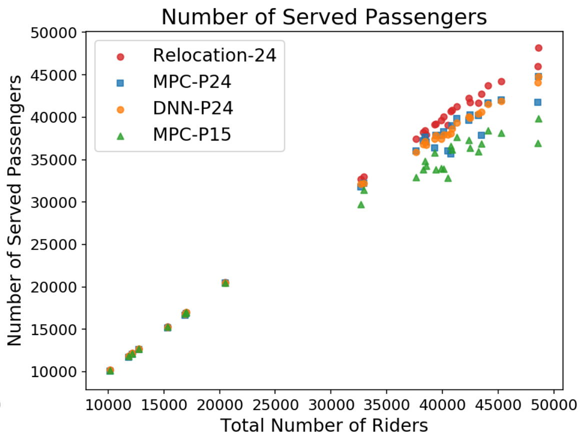
the 0-1 loss (percentage of time that the rounded predictions were wrong). Since the DNN models achieved the highest accuracy in both cases, they were selected as the final models. The error for each zone under the DNN model is given in Figure 5.5. The prediction errors for all zones are reasonable, although a few zones exhibit higher loss than others. Overall these results indicate that the models successfully learned the MPC decisions.

5.3.3 The Benefits of Learning the MPC Model

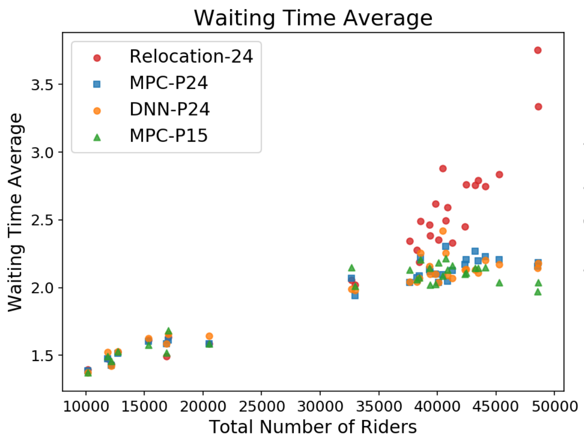
The trained machine learning models are evaluated on Yellow Taxi data in $2017/06$. The proposed methodology (DNN-P24) is compared with the original MPC model with 24 zones (MPC-P24), an MPC model at lower spatial fidelity with 15 clustered zones (MPC-P15), and a baseline that only performs relocation but not pricing (Relocation-24). *All*



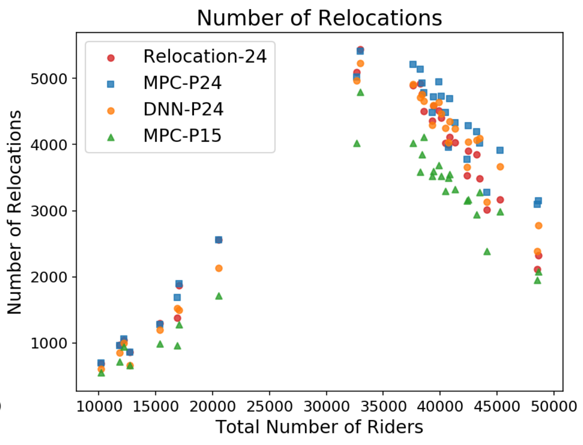
(a) Drop Out Percentage.



(b) Riders Served.



(c) Waiting Time Averages.



(d) Number of Relocation.

Figure 5.6: Evaluation Results of DNN-P24 and Its Comparison With Pure Optimization Approaches.

models but MPC-P24 are solved near optimally in 5 *seconds with 6 CPU cores as discussed in Section 5.3.1. MPC-P24, which cannot be solved near-optimally in* 5 *seconds, is given more time to solve and represents the ideal solution that cannot be achieved in real-time due to computational limits.* In particular, 2.4% MPC-P24 instances failed to find a solution within 20% optimality gap in 5 seconds. MPC-P15, on the other hand, can be solved in 5 seconds due to its lower granularity, therefore representing what can be achieved in real-time. The drop-out rate, number of riders served, rider waiting time averages, and number of relocations are reported in Figure 5.6. In all instances, DNN-P24 achieves similar performance as MPC-P24. Both approaches ensure a drop-out rate near zero, whereas

Table 5.3: The Transportation Model Run Times.

	Mean	Max
Solver Time (s)	0.004	0.093

Relocation-24 loses increasingly more riders as the instance becomes larger. DNN-P24 and MPC-P24 also achieve lower average waiting times and serve similar number of riders as Relocation-24. In addition, they serve more riders than MPC-P15: on large instances with more than 25,000 riders, DNN-P24 serves on average 6.7% more riders than MPC-P15 by pricing out fewer riders, demonstrating the benefits of higher model fidelity. DNN-P24 and MPC-P24 perform more relocations than MPC-P15 to serve more riders (finer spatial partition also unveils more opportunities for vehicle relocation). The solver times of the transportation optimization are reported in Table 5.3 - they never exceed 0.093 seconds. The prediction time is also within fraction of a second. *Overall, these promising results demonstrate that the proposed framework is capable of approximating the MPC model at high-fidelity efficiently, which leads to significant improvements in service quality.*

5.4 Conclusions

Large-scale ride-hailing systems often combine real-time routing at the individual request level with a macroscopic Model Predictive Control (MPC) optimization for dynamic pricing and vehicle relocation. The MPC operates over a longer time horizon to compensate for the myopic nature of the routing. However, the longer horizon increases computational complexity and forces the MPC to use coarser spatial-temporal granularity, degrading the quality of its decisions. This work addresses the computational challenge by imitating the MPC optimization by machine learning and a polynomial-time transportation optimization. The resulting learning & optimization approach serves as the optimization proxy, allowing the MPC to be considered at higher spatial and/or temporal fidelity since the optimizations can be solved and learned offline. Experimental results on the New York Taxi data set show that the proposed approach is computationally efficient and serves 6.7% more riders than the original optimization approach due to its higher fidelity.

CHAPTER 6

REINFORCEMENT LEARNING FROM OPTIMIZATION-PROXY

The previous chapter presents a learning & optimization framework to reduce the typical NP-Hard pricing and relocation optimization to a polynomial-time procedure. Reducing the computational cost however, does not tackle all the drawbacks of the model-based approach - the optimization model's performance heavily depends on how well it approximates the underlying ride-hailing dynamics (e.g., how the demand and supply interact over a period of time), which is challenging since the real-time dynamics is highly complex and volatile. For instance, anticipating how demand will be served is an immensely difficult task as the routing algorithm may be a complex mechanism itself. In addition, the model is confined to the time horizon that it considers and is not able to capture the long-term effects that may be present in the system.

Reinforcement learning, on the other hand, has the advantage of learning by interacting with the system and receiving feedback (reward) for its actions. As discussed in Chapter 3, RL can be divided into three streams: single-agent RL, decentralized multi-agent RL, and centralized multi-agent RL. The single-agent framework maximizes the reward of an individual agent, while the multi-agent framework maximizes system-level benefits. A main challenge of RL is training complexity since the state and action spaces are typically high-dimensional (often infinite-dimensional) due to the complex demand-supply dynamics. Sampling in high-dimensional spaces makes the training computationally expensive and unstable. Consequently, many papers simplify the problem by enforcing agents within the same region to follow the same policy [47, 22], or restricting relocations to only neighboring regions [18, 19, 20, 21, 22, 5]. Another challenge is promoting coordination among a large number of agents (vehicles). Single-agent RL focuses on a single vehicle and ignores group-level reward. Decentralized multi-agent RL considers group-level benefits only in a limited fashion since the state/action/reward of the individual agents are still modeled separately. Centralized multi-agent RL considers the state and action of the agents jointly and has the potential to achieve maximal cooperation. However, the joint state-action spaces are extremely high-dimensional and make the problem computationally prohibitive. Mao et al. proposed the only work using a fully-centralized formulation [24]. They modeled each dispatch zone instead of a vehicle as an agent to simplify the state-action space. Nevertheless, the approach was demonstrated in a simple setting with a small number of zones due to computational complexity. Their model also only makes relocation decisions but not pricing.

While MPC and RL seem drastically different - one is model-based and the other model-free; one is data-independent and the other data-dependent - they have different strengths and weaknesses and could complement each other. In this chapter, we propose a Reinforcement Learning from Optimization Proxy (RLOP) approach that combines optimization with reinforcement learning (Figure 6.1). The RLOP framework is a special case of the Reinforcement Learning from Expert Demonstration (RLED) framework where the expert is an optimization algorithm [48]. *To the best of our knowledge, this work is the first application of an RLED framework to vehicle relocation and dynamic pricing problems, and one of the few RL models with a fully-centralized policy.*
The RLOP approach consists of two main steps:

1. It first applies imitation learning to obtain an *optimization proxy* for a given optimization model, i.e., it trains a machine learning model to approximate the mapping between the inputs of the optimization and its actionable decisions;

2. It then seeds an RL component with the optimization proxy as the initial policy. The RL component further improves this policy by interacting with the environment, capturing the real system dynamics and long-term effects that are beyond the capabilities of the model-based approach.

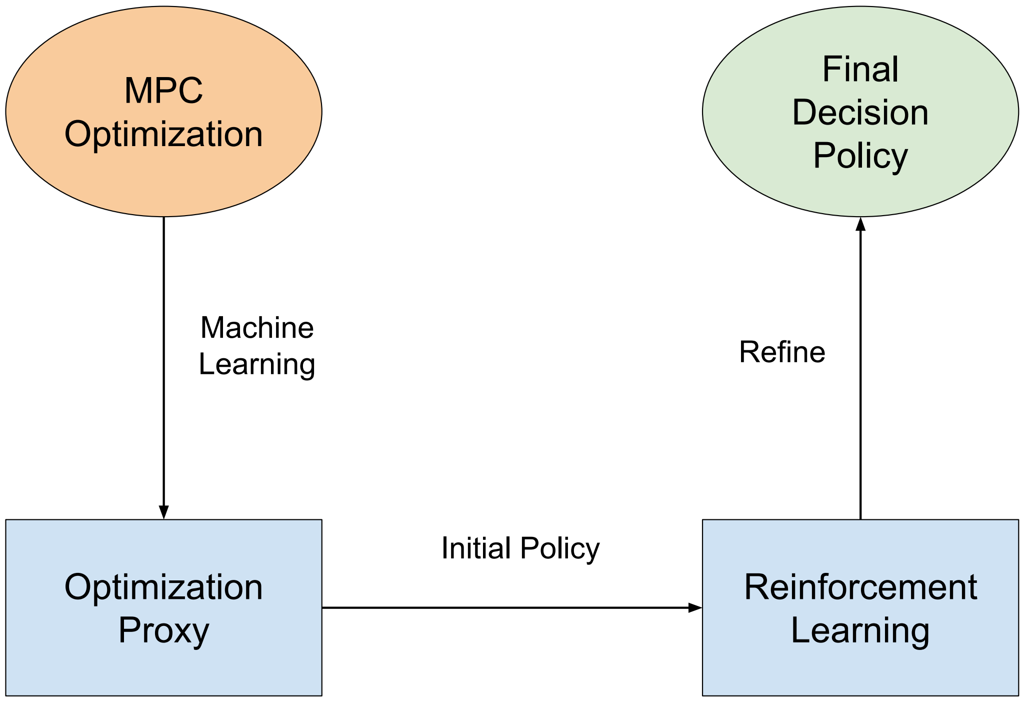


Figure 6.1: The RLOP Framework.

The RLOP framework has three important benefits. First, the optimization proxy approximates the model-based optimization with high fidelity and is order of magnitude faster. Second, the RL component improves the optimization proxy by capturing long-term effects and real system dynamics present in sample trajectories, which is beyond the capability of the original optimization. Third, The RL component is significantly easier to train since it starts with a high-quality policy. *Compared to [24], the only fully-centralized approach in the literature, RLOP is able to demonstrate the centralized approach on a much larger scale due to its training efficiency.*

The proposed RLOP framework is evaluated on the New York Taxi data set, using the optimization and simulation architecture presented by [17]. The experimental results reveal two interesting findings:

1. The optimization proxy learns the optimization model with high fidelity, producing similar objective values at a fraction of the optimization's computing time.

2. The RL component warm-started by the optimization proxy is computationally efficient, converging in a few iterations while pure centralized reinforcement learning is too expensive computationally to be applied.

3. The RL component further reduces the relocation costs by 5.9% compared to the optimization proxy.

6.1 RLOP Framework

The RLOP framework is composed of two stages: imitation learning and reinforcement learning. The imitation learning stage trains an optimization proxy, i.e., a machine-learning module that approximates the actionable decisions of a pricing and relocation optimization model. The reinforcement-learning stage takes the optimization proxy as the initial policy and refines it by a policy gradient method.

6.1.1 Imitation Learning

The imitation-learning stage trains an optimization proxy to predict the actionable decisions of a pricing and relocation optimization model $\Omega : \mathcal{I} \to \mathcal{W}$, where $\mathcal{I}$ is the model input and $\mathcal{W} := \mathcal{R} \cup \mathcal{M}$ is the model's pricing decisions $\mathcal{M}$ and the relocation decisions $\mathcal{R}$. $\mathcal{M}$ needs to be on the zone-level (price/demand in each zone) and $\mathcal{R}$ needs to be on the zone-to-zone level (number of relocations between each zone). The methodology for training the optimization proxy is presented in Chapter 5. Specifically, the optimization proxy has two machine learning models, $\hat{\mathcal{O}}_p : \mathcal{I} \to \mathcal{M}$ and $\hat{\mathcal{O}}_r : \mathcal{I}' \to \mathcal{X}$, which sequentially predict the pricing and relocation decisions. The zone-level relocation prediction $\mathcal{X}$ is disaggregated to the original zone-to-zone level via a transportation optimization $\mathcal{TO} : \mathcal{X} \to \mathcal{R}$ following $\hat{\mathcal{O}}_r$'s prediction. To make sure that the optimization proxy can be refined by the policy gradient method, $\hat{\mathcal{O}}_r$ and $\hat{\mathcal{O}}_p$ need to be **differentiable** with respect to their parameters. For instance, they can be artificial neural networks or linear regression models but not decision trees.

6.1.2 Reinforcement Learning

The RL algorithm starts from $\hat{\mathcal{O}}_r$ and $\hat{\mathcal{O}}_p$ and improves them by a policy gradient method. Specifically, the RL step models the underlying problem as a Markov Decision Process (MDP). MDP is characterized by a tuple $< \mathcal{S}, \mathcal{A}, R, P, \gamma >$, which consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $R(\mathbf{s}, \mathbf{a})$, a transition function $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, and a discount factor $\gamma \in [0, 1]$. At each decision epoch t in the planning horizon $\{0, 1, ..., T_e\}$, the agent observes the state of the system $\mathbf{s}_t \in \mathcal{S}$, takes an action $\mathbf{a}_t \in \mathcal{A}$, receives an immediate reward $R(\mathbf{s}_t, \mathbf{a}_t)$, and transitions to the next state $\mathbf{s}_{t+1}$ according to the transition probability $P(\cdot|\mathbf{s}_t, \mathbf{a}_t)$. For simplicity, we will use R_t in place of $R(\mathbf{s}_t, \mathbf{a}_t)$ throughout the discussion. The goal is to find a stochastic decision policy $\pi_\theta : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ parametrized by θ, i.e., a mapping from the state space to a probability distribution over the action space, to maximize the total expected discounted reward

$$J(\theta) = \mathbb{E}_{P,\pi_\theta}\left[\sum_{t=0}^{T_e} \gamma^t R_t\right] \tag{6.1}$$

The policy is trained iteratively based on the Policy Gradient Theorem [8]

$$\nabla_\theta J(\theta) = \mathbb{E}_{P,\pi_\theta}\left[\sum_{t=0}^{T_e} G_t \nabla_\theta \log P_{\pi_\theta}(\mathbf{a}_t|\mathbf{s}_t)\right] \tag{6.2}$$

where $G_t = \sum_{\tau=t}^{T_e} \gamma^{\tau-t} R_\tau$ is the total discounted reward since epoch t in the trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, R_0, ..., \mathbf{s}_{T_e}, \mathbf{a}_{T_e}, R_{T_e})$ and $P_{\pi_\theta}(\mathbf{a}_t|\mathbf{s}_t)$ is the probability of taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$ under the decision policy π_θ. In reality, computing the expectation in (6.2) is intractable since transition probability P does not have a closed-form expression. The gradient is approximated by Monte-Carlo sampling, i.e.,

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T_e} G_t^i \nabla_\theta \log P_{\pi_\theta}(\mathbf{a}_t^i|\mathbf{s}_t^i) \tag{6.3}$$

```
Algorithm 1: RLOP
  Input: A pricing and relocation optimization model $\mathcal{M}$.
1 Train differentiable machine learning models $\hat{\mathcal{O}}_r$ and $\hat{\mathcal{O}}_p$ to approximate $\mathcal{M}$;
2 Choose learning rates $\alpha_p$ and $\alpha_r$, parameters $\beta_p$ and $\beta_r$, discount factor $\gamma$, and
    covariances $\Sigma^p$ and $\Sigma^r$;
3 while not converged do
4     $\hat{\mathcal{O}}_r \leftarrow$ Apply Policy Gradient (Algorithm 2) with $(\alpha_r, \beta_r, \gamma, \hat{\mathcal{O}}_r, \Sigma^r)$;
5     $\hat{\mathcal{O}}_p \leftarrow$ Apply Policy Gradient (Algorithm 2) with $(\alpha_p, \beta_p, \gamma, \hat{\mathcal{O}}_p, \Sigma^p)$;
6 end
```

where $\{\tau_i\}_{i=1}^N = \{(\mathbf{s}_0^i, \mathbf{a}_0^i, R_0^i, ..., \mathbf{s}_{T_e}^i, \mathbf{a}_{T_e}^i, R_{T_e}^i)\}_{i=1}^N$ are trajectories generated by applying π_θ in the simulation environment.

Due to the complex inter-dependencies between pricing and relocation, training a single decision policy for both is challenging because of high sampling variance. The two decisions are also distinct in nature - pricing controls the demand and relocation controls the supply. This requires the design of different reward functions and training techniques. *The RLOP framework instead trains two separate policies, one for pricing and another for relocation, starting from the optimization proxy $\hat{\mathcal{O}}_p$ and $\hat{\mathcal{O}}_r$, respectively.* The two policies are trained iteratively with one policy fixed and the other policy updated by the policy gradient method until convergence. The iterative algorithm is summarized in Algorithm 1. Next we will present the policy gradient algorithm.

The underlying MDP of relocation reinforcement learning is defined as follows. The state and action spaces are the same as the input and output spaces of $\hat{\mathcal{O}}_r : \mathcal{I}' \to \mathcal{X}$ so that $\hat{\mathcal{O}}_r$ can be transformed into an initial policy for RL. The details of this transformation will be presented shortly. The reward function is $R(\mathbf{s}_t, \mathbf{a}_t) = -u_t - \beta_r v_t$, a weighted average of customer satisfaction and system cost, where u_t is the total waiting time of riders who emerge in epoch t and v_t is the expected time that vehicles will relocate due to action $\mathbf{a}_t$. For dropout riders, u_t can be assigned a big number as a penalty. Both u_t and v_t are in minutes. Parameter β_r denotes the relative importance of system cost compared to customer satisfaction, e.g., $\beta_r = 0.5$ implies that the platform is willing to relocate up to 2

Algorithm 2: Policy Gradient (Gaussian Policy)

```
Input: Learning rate $\alpha$, parameter $\beta$, discount factor $\gamma$, initial deterministic policy
       $f_\theta$, and covariance $\Sigma$.
Output: Trained policy $f_\theta$
1  for Episode $= 1, 2, ...$ do
2      for $i = 1, ..., N$ do
3          for $t = 0, 1, ...T_e$ do
4              Observe current state $\mathbf{s}_t^i$ and sample an action $\mathbf{a}_t^i$ from Gaussian policy
                 $\pi_\theta(\mathbf{s}_t^i) = \mathcal{N}(f_\theta(\mathbf{s}_t^i), \Sigma)$;
5              Restore $\mathbf{a}_t^i$ to a feasible solution, implement it in the simulator, and
                 compute reward $R_t^i$ based on $\beta$;
6          end
7      end
8      Compute total discounted reward $G_t^i = \sum_{\tau=t}^{T_e} \gamma^{\tau-t} R_\tau^i$ for all $i, t$;
9      Compute the policy gradient $\nabla_\theta J(\theta)$ by Equation (6.3);
10     $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
11 end
```

minutes for a 1 minute reduction in waiting time. β_r depends on the platform's underlying objective and is taken as an input. The transition function $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ depends on the underlying vehicle-routing algorithm, travel times, and the pricing policy, and does not have a closed-form expression.

The underlying MDP of the pricing reinforcement learning is defined as follows. The state and action spaces are the same as the input and output spaces of $\hat{\mathcal{O}}_p : \mathcal{I} \to \mathcal{M}$. The reward function is $R(\mathbf{s}_t, \mathbf{a}_t) = -n_t^s + \beta_p n_t^u$ where n_t^s is the number of served riders who emerge in epoch t and n_t^u is the number of dropout riders who emerge in epoch t. β_p controls the tradeoff between fulfilled rides and dropouts. $\beta_p = 10$ implies that the platform allows a rider to dropout only if it can serve at least 10 riders instead. β_p depends on the platform's underlying objective and is taken as an input. The transition function $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ depends on the underlying vehicle-routing algorithm, travel times, and the relocation policy, and does not have a closed-form expression.

It remains to specify how the optimization proxy can be turned into initial policies for RL. We will use $\hat{\mathcal{O}}_r$ as an example as $\hat{\mathcal{O}}_p$ will follow the same procedure. Recall that

$\hat{\mathcal{O}}_r : \mathcal{I}' \to \mathcal{X}$ is a deterministic mapping from the state space to the action space. In the RLOP framework, the policy gradient algorithm starts from Gaussian policy $\pi_\theta^0(\cdot) = \mathcal{N}(\hat{\mathcal{O}}_r(\cdot), \Sigma)$ centered around $\hat{\mathcal{O}}_r$ with covariance Σ. The covariance matrix Σ is a diagonal matrix whose diagonal entry Σ_{ii} is the (sampling) variance of an relocation action x_i (x_i is an entry of $\mathbf{x} \in \mathcal{X}$). Note that x_i is one of the prediction labels of $\hat{\mathcal{O}}_r$, so its empirical distribution can be estimated in the imitation-learning stage. Therefore, Σ_{ii} can be taken as a certain percentage of x_i's characteristic statistics such as its empirical mean or median. Prior knowledge of Σ is extremely valuable since a well-chosen sampling variance can lead to more efficient exploration during training.

The policy gradient algorithm is summarized in Algorithm 2. Note that after sampling an action $\mathbf{a}$ from π_θ, it should be restored to a feasible solution before being implemented - it should be rounded to the nearest demand multiplier if it is a pricing decision or restored to zone-to-zone level via the transportation optimization $\mathcal{TO}$ if it is a relocation decision. Note that the policy gradient algorithm is general and can incorporate any specific reinforcement-learning techniques (e.g., actor-critic, PPO, off-policy sampling, etc.) appropriate for the problem at hand.

6.2 Simulation Study

The RLOP framework is evaluated on Yellow Taxi Data in Manhattan, New York City [43]. It is trained from $2017/01$ to $2017/05$ and evaluated in $2017/06$ during morning rush hours on weekdays. Section 6.2.1 reviews the simulation environment. Section 6.2.2 presents the imitation-learning results. Section 6.2.3 presents the reinforcement-learning results. Section 6.2.4 evaluates the performance of the policy.

6.2.1 Simulation Environment

The experiments use the end-to-end simulation framework in [17]. The Manhattan area is partitioned into a grid of cells of 200 squared meters and each cell represents a pickup/dropoff location. Travel times between the cells are queried from [38]. The fleet is fixed to be 1600 vehicles with capacity 4, distributed randomly among the cells at the beginning of the simulation. Riders must be picked up in 10 minutes and matched to a vehicle in 5 minutes since their requests, after which they drop out.

The simulator has two main components: the ride-sharing routing algorithm reviewed in Section 2.5 and the MPC model in Chapter 5. The routing algorithm batches riders into a time window and optimizes every 30 seconds. The MPC model is executed every 5 minutes. It partitions the Manhattan area into 24 zones and time into 5-minute epochs (Figure 5.3). Its planning horizon contains 4 epochs and riders can be served in 2 epochs following their requests. The number of idle vehicles in each epoch is estimated by the simulator based on the current route of each vehicle and the travel times. The ride-share ratio is $W_{ij} = 1.5$ for all $i, j \in \mathcal{Z}$. Service weight and relocation penalty are $q^p(t, \rho) = 0.5^t 0.75^{\rho - t}$ and $q^r_{ij}(t) = 0.001 * 0.5^t \eta_{ij}$ where η_{ij} is travel time between zone i and zone j in seconds. Five demand multipliers $[100, 75, 50, 25, 0]$ are available for each zone and epoch. The zone-to-zone demand D_{ijt} is forecasted based on historical data. The design of demand forecasting techniques is beyond the scope of this work. We first forecast zone-level demand $D_{it} = \sum_{j \in \mathcal{Z}} D_{ijt}$ and then assign the destinations based on historical distribution. The reason for doing zone-level prediction is to reduce sparsity in D_{ijt}, since most trips travel between a few popular regions. The forecasting model is a 2-layer fully-connected neural network with $(256, 256)$ hidden units and RELU activation functions. The loss function is MSE loss with l_1-regularization. It is trained from $2017/01$ to $2017/05$ and tested in $2017/06$. The original time series data is augmented by injecting white noise sampled from a uniform distribution $U(-5, 5)$ to create more training data. To predict zone-level demand in the MPC horizon $\{D_{it}\}_{i \in \mathcal{Z}, t \in \mathcal{T}}$, the model uses the demand observed in the previous 3 epochs, as well as demand observed a week ago during the same period to account for seasonality. For example, when forecasting demand from 8:00am to 8:20am (4 epochs) on $2017/06/08$, the model uses demand from 7:45am to 8:00am on $2017/06/08$ and demand from 7:45am to 8:20am on $2017/06/01$. After zone-level demand is predicted, it is assigned to zone-to-zone level based on the historical distribution of the trip's destination. For example, if μ_{ij} proportion of trips from zone i goes to zone j during the hour of the prediction and $\hat{D}_{it}$ is the demand prediction for zone i, the final zone-to-zone prediction is $\hat{D}_{ijt} = \hat{D}_{it} \times \mu_{ij}$ rounded to the nearest integer. Overall, the mean squared error of the zone-to-zone level forecast in $2017/06$ is 0.86.

The MPC's pricing decisions are implemented at the level of demand multipliers: if MPC decides to keep 50% demand in a zone, the simulation randomly keeps 50% requests in the current epoch and discards the rest. After the MPC decides zone-to-zone level relocations, a vehicle assignment optimization determines which individual vehicles to relocate by minimizing total traveling distances [17]. Of the routing, relocation, and vehicle assignment models, the routing model is the most computationally intensive since it operates on the individual (driver and rider) level as opposed to the zone level. Since all three models must be executed in the 30 seconds batch window, the experiments allocate 15 seconds to the routing optimization, 10 seconds to the MPC, and 5 seconds to the vehicle assignment. All the models are solved using Gurobi 9.1 with 24 cores of 2.1 GHz Intel Skylake Xeon CPU [39].

6.2.2 Imitation Learning

The optimization proxy is trained from $2017/01$ to $2017/05$, 8:00am - 9:00am, Monday to Friday, when the demand is at its peak and the need for relocation and pricing the greatest. The number of riders in these instances ranges from 22,000 to 29,000, providing a wide variety of demand distribution. The weekends and non-busy hours see much less demand and should be considered separately. The experimental study focuses on the busy hour because it is the most interesting and necessary period for balancing demand and supply. Data augmentation is employed to generate more instances: each 1-hour instance is perturbed by randomly adding/deleting a certain percentage of requests where the percentages are

Table 6.1: Imitation Learning Testing Loss.

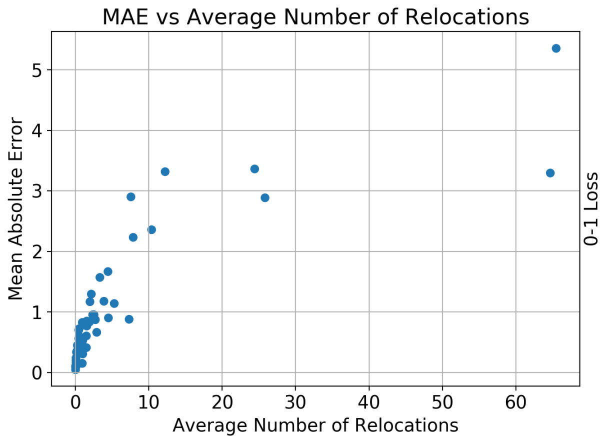
Relocation (MSE)	Pricing (MSE)	Pricing (0-1 Loss(%))
6.88	62.52	9.2

sampled from a uniform distribution $U(-5, 5)$. The instances are run by the simulator and the MPC model's inputs and outputs are extracted as training data. In total, $15,000$ data points are used in training and 2500 data points are used for testing.

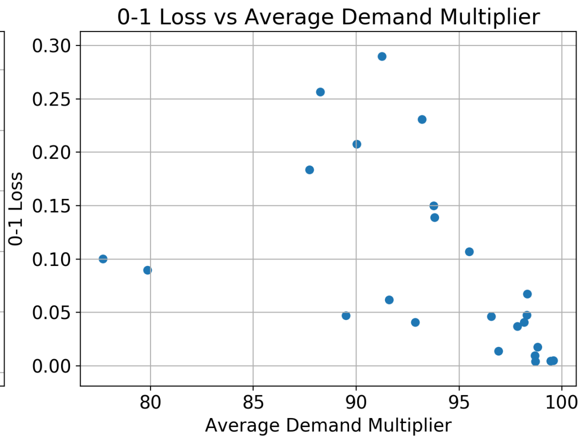
The two machine learning models $\hat{\mathcal{O}}_r$ and $\hat{\mathcal{O}}_p$ are both fully-connected neural networks with 2 hidden layers of (512, 256) units. The relocation-learning model uses the hyperbolic tangent (tanh) activation function and the pricing-learning model uses the RELU activation function for the hidden layers. The final output layer uses RELU in both cases. The two models both use mean-squared error (MSE) loss with l_1-regularization. The training is conducted in Pytorch by Adam optimizer with batch size 32 and learning rate 10^{-3} [45, 46]. The trained models are evaluated on the testing set where the predictions are rounded to feasible solutions by the procedures described in Chapter 5. The overall loss after rounding is reported in Table 6.1, where the relocation model reports the mean squared error (MSE) loss and the pricing model reports both the MSE and the 0-1 loss (percentage of time that the rounded predictions were wrong). The loss of each zone is reported in Figure 6.2. The errors for all the zones are reasonable, although a few zones exhibit higher loss than others. In addition, the optimization proxy achieves similar performance as the MPC in simulation: the detailed results are presented in Section 6.2.4. Overall, these results indicate that the optimization proxy successfully learned the MPC decisions.

6.2.3 Reinforcement Learning

The optimization proxy is refined by reinforcement learning in $2017/05$. Since the number of riders in most daily instances ranges from 22,000 to 29,000, four instances with [23960, 25768, 27117, 28312] riders are selected and the policy is trained on these representative instances. To stabilize training, it is common practice to subtract a baseline from the reward



(a) Relocation Predictions.



(b) Pricing Predictions.

Figure 6.2: MPC Decisions Predictions: Each Point on the Left Denotes the Average Number of Relocations and the MAE of Relocation Predictions for a Zone. Each Point on the Right Denotes the Average Demand Multiplier and the 0-1 Loss of the Pricing Prediction of a Zone.

to distinguish good and bad actions when computing the policy gradient:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T_e} (G_t^i - b_t^i) \nabla_\theta \log P_{\pi_\theta}(\mathbf{a}_t^i | \mathbf{s}_t^i) \tag{6.4}$$

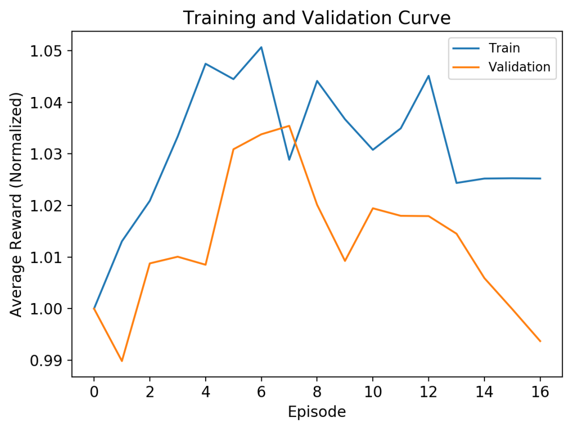
where b_t^i is the baseline representing the expected reward since t following the current policy. $(G_t^i - b_t^i)$ therefore measures the ”advantage” of this trajectory's decisions over the current policy. The baseline b_t^i can be estimated in many different ways [49]. This work employs the sample average method: it samples $K = 10$ trajectories for each training instance and takes the sample average as baseline, i.e., $b_t^i = \frac{1}{K}\sum_{k=1}^{K} G_t^{ik}$ if trajectories for instance i are indexed by $\{i_1, ...i_K\}$. Therefore each policy gradient update is based on $4K = 40$ sample trajectories.

Algorithm 1 is run with $(\alpha_r, \beta_r, \alpha_p, \beta_p, \gamma) = (0.005, 0.75, 0.001, 50, 0.75)$. $\Sigma_{ii}^r = 0.05x_i^{0.75}$ where $x_i^{0.75}$ is the 75th percentile of relocation action x_i in the imitation-learning data set (recall that x_i is a prediction label of $\hat{O}_r$). $\Sigma_{ii}^p = 0.1\sigma(m_{i0})$ where $\sigma(m_{i0})$ is the standard deviation of demand multiplier m_{i0} in the imitation-learning data set. To make sure that RL does not overfit on the selected representative instances, the policy is validated on other instances in $2017/05$ after each training episode in Algorithm 2. Algorithm 2 stops when the average reward on the validation set fails to improve upon the best average reward for 10 consecutive episodes. The overall training (Algorithm 1) stops when the average reward on the validation set fails to improve by at least 1% in two consecutive iterations (two executions of Algorithm 2).

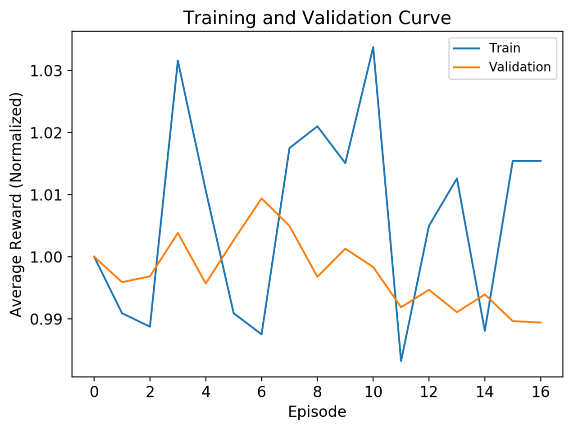
In the experiment, Algorithm 1 converges after 3 iterations. The normalized training and validation curves of each iteration are given in Figure 6.3. The three policy gradient iterations improve the reward on the validation set by 3%, 1%, and 0%, resp. Each iteration also converges in relatively few episodes, which is significantly more efficient than pure reinforcement learning algorithms which typically converge in tens of thousands of episodes.

6.2.4 Evaluation Results

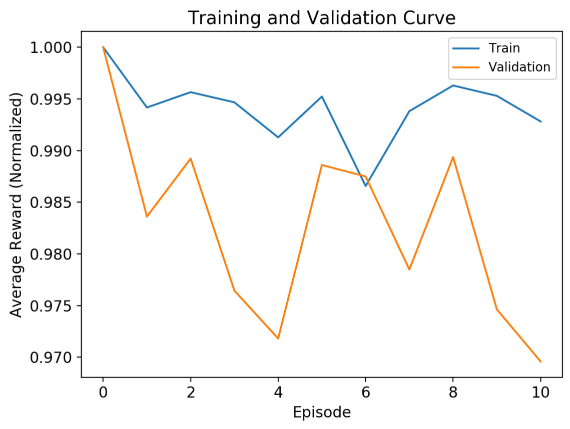
The trained policy is evaluated on weekdays in $2017/06$. The proposed RLOP approach is compared with the optimization proxy as well as the MPC optimization. Pure reinforcement learning without initial policies seeded with the optimization proxy (Algorithm 1 without the first step) fails to converge due to the high-dimensional state and action spaces: it is too expensive computationally to be applied in this setting. Figure 6.4 reports the number of riders served, the percentage of riders who drop out, the average rider waiting time (minutes), and the average vehicle relocation time (minutes) on each weekday in $2017/06$. Table 6.2 reports their monthly averages as well as the average model run times. The optimization proxy and the MPC optimization achieve similar performance on almost all daily instances. The optimization proxy sees slightly higher waiting time, but serves more riders with fewer dropouts. The optimization proxy outperforms the MPC on certain metrics because the MPC optimization is based on an approximation of the ride-sharing system - its decisions are optimal for the approximation but not necessarily for the real system. The RLOP's performance is similar to the other two models' performance in terms of riders



(a) Iteration 1.



(b) Iteration 2.



(c) Iteration 3.

Figure 6.3: Training and Validation Curve of Reinforcement Learning (Normalized).

served, waiting time, and drop-out percentage. Its relocation cost, on the other hand, is much lower. In particular, its relocation time is 5.4% lower than the MPC and 5.9% lower than the optimization proxy. The optimization proxy and the RLOP are also much faster than the MPC and are guaranteed to run in polynomial time. On average, the MPC instances take 2.6s to solve while the optimization proxy and the RLOP take only fractions of a second. The most computational cost of RLOP lies in the offline training stage where data for imitation learning and RL are generated through simulation. Nevertheless, RLOP is still more efficient than pure RL which requires a prohibitively large number of samples to train when starting from a random policy. *These promising results show that the RLOP is an efficient and effective approach for vehicle relocation and dynamic pricing in real-time*

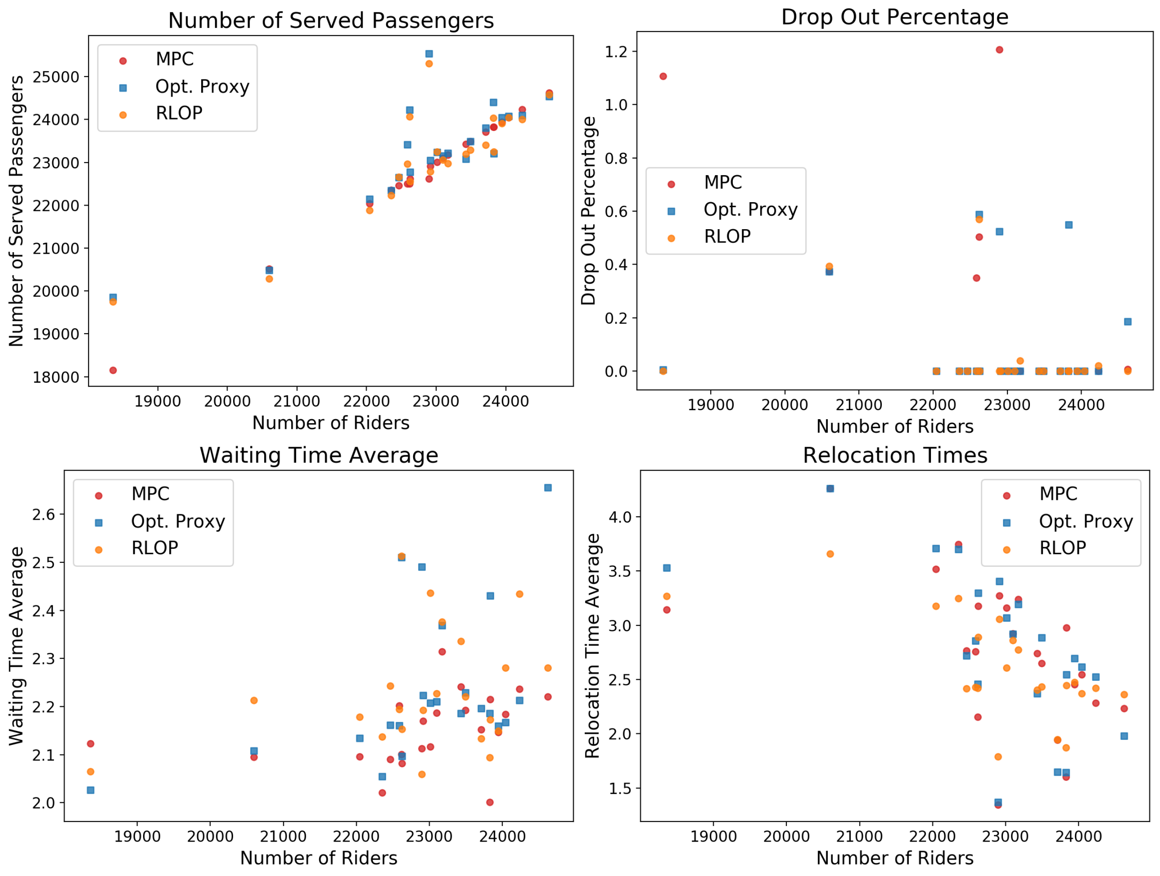


Figure 6.4: Evaluation Results of the RLOP, the Optimization Proxy, and the MPC Optimization. Each Point in the Plots Represents a Weekday in 2017/06.

settings.

6.3 Conclusions

Idle vehicle relocation and dynamic pricing are crucial for addressing demand-supply imbalances that frequently arise in the ride-hailing system. Current mainstream methodologies - optimization and reinforcement learning - suffer from computational complexity in either offline training or online deployment. This chapter proposes a reinforcement learning from Optimization Proxy (RLOP) approach to alleviate their computational burden and search for better policies. Specifically, RLOP trains two machine-learning models to approximate the pricing and relocation decisions of an optimization model, and then refines them iteratively by reinforcement learning. On the New York City dataset, the RLOP ap-

Table 6.2: Summary Statistics of Tested Models.

	Riders Served	Dropout Pct.	Wait Time	Reloc. Time	Run Time (s)
MPC	22867	0.16%	2.15	2.77	2.594
Opt. Proxy	23219	0.10%	2.24	2.79	0.028
RLOP	22997	0.11%	2.25	2.63	0.027

proach achieves significantly lower relocation costs and computation time compared to the optimization approach, while pure reinforcement learning is too expensive computationally for practical purposes.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

Idle vehicle relocation and dynamic pricing are important tools for addressing demand-supply imbalances that frequently arise in the ride-hailing market. Although interdependent, the two decisions have largely been studied independently in the literature due to modeling and computational complexity. The current mainstream methodologies, MPC and RL, also suffer from significant computational limitations in either online deployment or offline learning. To tackle these drawbacks and expand the research horizon, this thesis first proposes an optimization model computing both relocation and pricing decisions in Chapter 4. The model guarantees reasonable waiting time for the riders by reducing or postponing requests that are beyond the system's service capacity. Chapter 5 presents a learning & optimization framework to approximate the optimal solutions of a general pricing and relocation optimization, serving as an optimization-proxy. It reduces the typical NP-Hard optimization to a polynomial-time procedure of prediction and (efficient) optimization, allowing the optimization to be considered at higher fidelity and thus improving the quality of decisions. Chapter 6 takes one step further: it refines the optimization proxy in Chapter 5 by reinforcement learning, combining the strengths of both model-based and model-free approaches while overcoming their drawbacks. This hybrid approach is the very first Reinforcement Learning from Expert Demonstration (RLED) framework applied to relocation and pricing problems. It is not only computationally efficient but also demonstrates superior performance than MPC and RL alone.

7.2 Future Work

While this thesis expands the research horizon and tackles the computational limitations of the existing approaches, there are still many important research questions and new possibilities to explore.

The thesis has assumed that trip demand depends only on price. Nowadays, most ride-hailing services also provide estimated time of arrival (ETA) when the passenger searches for a ride, which factors into her decision-making. Therefore, using a more realistic demand model for dynamic pricing and vehicle relocation can be a further direction. Another key assumption in the thesis is that an accurate forecast of supply, demand, and travel time is available at hand. Both the optimization and the learning models make their decisions based on point estimates, which can be highly inaccurate as real-time dynamics is extremely volatile. Future research can therefore account for input uncertainty, developing stochastic optimization or RL models that leverage a set of uncertainty information (confidence intervals/distribution information) [14, 16]. The model can also incorporate robustness into the objective, optimizing not only the average case performance but also the worst case performance to control the risk and variance in real-time.

Furthermore, this thesis has taken number of rides and rider waiting time as the main performance metrics. In reality, more socio-economic factors need to be considered and a few are listed as an illustration. First, Low-income communities are more susceptible to surge pricing than high-income communities, leading to social fairness issues. Second, when the objective is to maximize number of rides or profit, the model may price out trips that are costly to serve: these trips may originate from remote areas that require long-distance pick-ups or go to areas where the trip demand is low. Discarding these trips is cost-effective for the platform but induces fairness issues. Third, when human drivers come into play, income equality becomes an important consideration. The platform needs to ensure similar income for the drivers to retain them in the long term. These various considerations are all essential to the ride-hailing operations but may have contradicting effects on the platform's decision-making. Research that balances multiple factors and explores their short-term and long-term socio-economic impact is greatly needed in the literature.

The learning frameworks in the thesis are trained entirely offline on historical data. During online deployment, however, the data distribution may deviate from the training data distribution, leading to potentially poor model performance. Future work can therefore focus on detecting distribution shift (change-point detection) and designing online learning methods to align the model with the latest distribution.

Future research could also extend from the centralized setting in this thesis to a non-centralized setting where drivers also respond to the pricing and relocation decisions of the platform, i.e., modeling the system as a two-sided market where both the demand and supply sides are affected by the platform's decisions. Estimating each driver's behaviors - such as when and where she wants to work, how long she is willing to work, and how much incentive she needs to perform certain actions (relocate/serve a trip at a certain price) - and incorporating them into the decision-making is an immensely complex and challenging task. Although some studies have considered driver's behavior [25, 26, 27, 28, 29, 30], they focus mostly on pricing's impact on the long-term market equilibrium. Real-time modeling of the two-sided market is much needed.

Finally, tackling real-time demand-supply imbalances could motivate new operational design. Part of the reason for frequent imbalances is that it is difficult to estimate demand and supply accurately beforehand. Having riders reserve their rides in advance, for example, cancels out some of the uncertainty and enables the platform to better plan ahead, a practice that ride-hailing companies such as Uber and Lyft have already adopted. Another possibility is carpooling where private car owners traveling for personal reasons (not for making profits from the trip) can find riders to share the ride on the platform (BlaBlaCar, Waze). Although carpooling services have existed for a long-time, they are largely operating as independent services and not combined with ride-hailing. Integrating them into the ride-hailing market can be another promising direction. Finally, the design of a multi-modal transportation system involving ride-hailing, public transit, and other mobility options can not only alleviate demand-supply imbalances but also reduce carbon emissions and decrease congestion.

In summary, on-demand mobility service is a complex system that involves various parties and is influenced by multiple factors. Its operational management requires contributions from both technical and non-technical sides. It is the author's wish that this thesis presents a small step forward and inspires innovative work in the future.

REFERENCES

[1] J. Hall, C. Kendrick, and C. Nosko, *The effects of uber's surge pricing: A case study*, https://economicsforlife.ca/wp-content/uploads/2015/10/effects_of_ubers_surge_pricing.pdf, 2015.

[2] J. C. Castillo, D. Knoepfle, and G. Weyl, "Surge pricing solves the wild goose chase," in *Proceedings of the 2017 ACM Conference on Economics and Computation*, ser. EC '17, Cambridge, Massachusetts, USA: Association for Computing Machinery, 2017, pp. 241–242, ISBN: 9781450345279.

[3] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017.

[4] C. Riley, A. Legrain, and P. V. Hentenryck, "Column generation for real-time ride-sharing operations," in *CPAIOR*, 2019.

[5] Y. Jiao *et al.*, "Real-world ride-hailing vehicle repositioning using deep reinforcement learning," *ArXiv*, vol. abs/2103.04555, 2021.

[6] E. Yuan and P. Van Hentenryck, "Real-time pricing optimization for ride-hailing quality of service," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, ijcai.org, 2021, pp. 3742–3748.

[7] E. Yuan, W. Chen, and P. V. Hentenryck, "Reinforcement learning from optimization proxy for ride-hailing vehicle relocation," *Journal of Artificial Intelligence Research*, 2022.

[8] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts.: The MIT Press., 2018.

[9] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, Aug. 1988.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[11] A. G. Kek, R. L. Cheu, Q. Meng, and C. H. Fung, "A decision support system for vehicle relocation operations in carsharing systems," *Transportation Research Part E: Logistics and Transportation Review*, vol. 45, no. 1, pp. 149–158, 2009.

[12] R. Zhang, F. Rossi, and M. Pavone, “Model predictive control of autonomous mobility-on-demand systems,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1382–1389.

[13] F. Miao *et al.*, “Taxi dispatch with real-time sensing data in metropolitan areas — a receding horizon control approach,” *IEEE Transactions on Automation Science and Engineering*, vol. 13, Apr. 2015.

[14] F. Miao, S. Han, A. M. Hendawi, M. E. Khalefa, J. A. Stankovic, and G. J. Pappas, “Data-driven distributionally robust vehicle balancing using dynamic region partitions,” in *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*, 2017, pp. 261–272.

[15] R. Iglesias, F. Rossi, K. Wang, D. Hallac, J. Leskovec, and M. Pavone, “Data-driven model predictive control of autonomous mobility-on-demand systems,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia: IEEE Press, 2018, pp. 1–7.

[16] M. Tsao, R. Iglesias, and M. Pavone, “Stochastic model predictive control for autonomous mobility on demand,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 3941–3948.

[17] C. Riley, P. van Hentenryck, and E. Yuan, “Real-time dispatching of large-scale ride-sharing systems: Integrating optimization, machine learning, and model predictive control,” in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, Special track on AI for CompSust and Human well-being, Jul. 2020.

[18] J. Wen, J. Zhao, and P. Jaillet, “Rebalancing shared mobility-on-demand systems: A reinforcement learning approach,” in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 220–225.

[19] J. Holler *et al.*, “Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem,” in *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, J. Wang, K. Shim, and X. Wu, Eds., IEEE, 2019, pp. 1090–1095.

[20] T. Oda and C. Joe-Wong, “Movi: A model-free approach to dynamic fleet management,” *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 2708–2716, 2018.

[21] M. Guériau and I. Dusparic, “Samod: Shared autonomous mobility-on-demand using decentralized reinforcement learning,” *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1558–1563, 2018.

[22] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '18, London, United Kingdom: Association for Computing Machinery, 2018, pp. 1774–1783, ISBN: 9781450355520.

[23] E. Liang, K. Wen, W. Lam, A. Sumalee, and R. Zhong, "An integrated reinforcement learning and centralized programming approach for online taxi dispatching," *IEEE Transactions on Neural Networks and Learning Systems*, Feb. 2021.

[24] C. Mao, Y. Liu, and Z.-J. Shen, "Dispatch of autonomous vehicles for taxi services: A deep reinforcement learning approach," *Transportation Research Part C: Emerging Technologies*, vol. 115, p. 102 626, 2020.

[25] S. Banerjee, R. Johari, and C. Riquelme, "Pricing in ride-sharing platforms: A queueing-theoretic approach," in *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, ser. EC '15, Portland, Oregon, USA: Association for Computing Machinery, 2015, p. 639, ISBN: 9781450334105.

[26] K. Bimpikis, O. Candogan, and D. Saban, "Spatial pricing in ride-sharing networks," *Operations Research*, vol. 67, no. 3, pp. 744–769, 2019.

[27] G. P. Cachon, K. M. Daniels, and R. Lobel, "The role of surge pricing on a service platform with self-scheduling capacity," *Manufacturing & Service Operations Management*, vol. 19, no. 3, pp. 368–384, 2017. eprint: https://doi.org/10.1287/msom.2017.0618.

[28] L. Zha, Y. Yin, and Z. Xu, "Geometric matching and spatial pricing in ride-sourcing markets," *Transportation Research Part C: Emerging Technologies*, vol. 92, pp. 58–75, 2018.

[29] E. Ozkan, "Joint pricing and matching in ride-sharing systems," *European Journal of Operational Research*, vol. 287, May 2020.

[30] Y. Chen and M. Hu, "Pricing and matching with forward-looking buyers and sellers," *Manufacturing & Service Operations Management*, vol. 22, Aug. 2019.

[31] H. Qiu, R. Li, and J. Zhao, "Dynamic pricing in shared mobility on demand service," *arXiv: Optimization and Control*, 2018.

[32] H. Chen *et al.*, "Inbede: Integrating contextual bandit with td learning for joint pricing and dispatch of ride-hailing platforms," in *2019 IEEE International Conference on Data Mining (ICDM)*, 2019, pp. 61–70.

[33] C. Lei, Z. Jiang, and Y. Ouyang, “Path-based dynamic pricing for vehicle allocation in ridesharing systems with fully compliant drivers,” *Transportation research procedia*, vol. 38, pp. 77–97, 2019.

[34] H. R. Sayarshad and 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*, vol. 106, pp. 60–77, 2017.

[35] A. Braverman, J. G. Dai, X. Liu, and L. Ying, “Empty-car routing in ridesharing systems,” *Operations Research*, vol. 67, no. 5, pp. 1437–1452, 2019.

[36] T.-Y. Ma, S. Rasulkhani, J. Y. Chow, and S. Klein, “A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 128, pp. 417–442, 2019.

[37] NYC, *Nyc taxi & limousine commission - trip record data*, https://www1.nyc.gov/site/tlc/passengers/taxi-fare.page, 2019.

[38] OpenStreetMap, *Openstreetmap*, https://www.openstreetmap.org, 2017.

[39] GurobiOptimization, *Gurobi optimizer reference manual*, 2022.

[40] NYC, *Nyc taxi & limousine commission - trip record data*, https://www1.nyc.gov/site/tlc/passengers/taxi-fare.page, 2019.

[41] Z. Lei, X. Qian, and S. V. Ukkusuri, “Efficient proactive vehicle relocation for on-demand mobility service with recurrent neural networks,” *Transportation Research Part C: Emerging Technologies*, vol. 117, p. 102 678, 2020.

[42] K. R. Rebman, “Total unimodularity and the transportation problem: A generalization,” *Linear Algebra and its Applications*, vol. 8, no. 1, pp. 11–24, 1974.

[43] NYC, *Nyc taxi & limousine commission - trip record data*, Accessed: 2020-10-01, 2019.

[44] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *International Conference on Learning Representations*, Dec. 2014.

[46] A. Paszke *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H.

Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[47] T. Verma, P. Varakantham, S. Kraus, and H. C. Lau, "Augmenting decisions of taxi drivers through reinforcement learning for improving revenues," in *ICAPS*, 2017.

[48] J. Ramírez, W. Yu, and A. Perrusquía, "Model-free reinforcement learning from expert demonstrations: A survey," *Artificial Intelligence Review*, vol. 1, no. 1, 2021.

[49] L. Weng, "Policy gradient algorithms," *lilianweng.github.io/lil-log*, 2018.