2021

Fleet management strategies for urban Mobility-on-Demand systems

BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

FLEET MANAGEMENT STRATEGIES FOR URBAN MOBILITY-ON-DEMAND SYSTEMS

by

HARSHAL ANIL CHAUDHARI

B.E., Birla Institute of Technology and Science, Pilani, 2013
M.S., Boston University, 2015

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2021

Approved by

First Reader

Evimaria Terzi, Ph.D.
Professor of Computer Science

Second Reader

John W. Byers, Ph.D.
Professor of Computer Science

Third Reader

Mark Crovella, Ph.D.
Professor of Computer Science

The people who are crazy enough
to think they can change
the world are the ones who do.

– Apple's 'Think Different' commercial, 1997.

Acknowledgments

"They say the secret of success is being at the right place at the right time. But since you never know when the right time is going to be, I figure the trick is to find the right place, and just hang around!"

– Calvin, in the comic book 'Calvin and Hobbes'.

The truth of the matter is that I took the words of Bill Watterson a tad bit too seriously, but the time spent at Boston University has unequivocally been one of the most beautiful chapters of my life so far. I have so many great experiences and memories that will last a lifetime. So, I dedicate this dissertation to all the great individuals I met along the way, to all those who inspired it but may not read it.

First, I am incredibly grateful to have found two amazing research advisors in Professor Evimaria Terzi and Professor John Byers. Evimaria's open-minded and cheerful guidance has made my research work an enjoyable experience. John's broad vision and attention to detail have undoubtedly raised the quality of my work. When I felt confused, both of them were always there to guide me out of the troubled waters and help me find my way. My Ph.D. work would have been impossible without their generous help and guidance.

I would like to extend my sincere gratitude to Professor Michael Mathioudakis, whose patience and help during the initial stages of my Ph.D. contributed to a significant part of this dissertation. I would also like to thank the other members of my dissertation committee. Professor Manos Athanassoulis introduced me to exciting problems outside of my primary field of research and made my last few years in the program thoroughly enjoyable. Professor Mark Crovella was always an encouraging presence in the department. Having followed his impactful research, I hope to work

with him in the future. I am very grateful to the amazing colleagues and friends in the department front office whose timely help and support were pivotal to my research.

Meanwhile, my life in Boston has been full of great memories with my peers and friends. From my friends within our research community (Giovanni, Harry, Andy, Panayota, Sofia, Thomas, Gavin, Sanaz, Isidora, Konstantinos, etc.), to my friends from other walks of life (Alexander, Marin, Joan, Dana, Kirstie, Annika, Mason, Jessica, Andrea, Tanmaya, Suraj, Gabriel, etc.), we had so much fun together. I will never forget the time we spent hanging out together, trying great food, making jokes about one another, and playing board games. I would like to thank them for putting up with my eccentricities and wish them all the best. A special shoutout to Marley, whose constant warmth and companionship I could not repay even with a lifetime of dog treats. But I will try anyway. Last but not least, I want to thank my brother, Pratik, for our numerous insightful discussions and my parents, Anil and Jyotsna, for their words of support and encouragement.

This journey may have reached its ceremonial end, but it would continue to be played out over and over again in the quietest chambers of my mind.

FLEET MANAGEMENT STRATEGIES FOR URBAN MOBILITY-ON-DEMAND SYSTEMS

HARSHAL ANIL CHAUDHARI

Boston University, Graduate School of Arts and Sciences, 2021

Major Professor: Evimaria Terzi, Ph.D.
Professor of Computer Science

ABSTRACT

In recent years, the paradigm of *personal urban mobility* has radically evolved as an increasing number of Mobility-on-Demand (MoD) systems continue to revolutionize urban transportation. Hailed as the future of sustainable transportation, with significant implications on urban planning, these systems typically utilize a fleet of shared vehicles such as bikes, electric scooters, cars, etc., and provide a centralized matching platform to deliver point-to-point mobility to passengers. In this dissertation, we study MoD systems along three operational directions – (1) modeling: developing analytical models that capture the rich stochasticity of passenger demand and its impact on the fleet distribution, (2) economics: devising strategies to maximize revenue, and, (3) control: developing coordination mechanisms aimed at optimizing platform throughput.

First, we focus on the metropolitan bike-sharing systems where platforms typically do not have access to real-time location data to ascertain the exact spatial distribution of their fleet. We formulate the problem of accurately predicting the fleet distribution as a Markov Chain monitoring problem on a graph representation of a city. Specifically, each monitor provides information on the exact number of bikes

transitioning to a specific node or traversing a specific edge at a particular time. Under budget constraints on the number of such monitors, we design efficient algorithms to determine appropriate monitoring operations and demonstrate their efficacy over synthetic and real datasets.

Second, we focus on the revenue maximization strategies for individual strategic driving partners on ride-hailing platforms. Under the key assumption that large-scale platform dynamics are agnostic to the actions of an individual strategic driver, we propose a series of dynamic programming-based algorithms to devise contingency plans that maximize the expected earnings of a driver. Using robust optimization techniques, we rigorously reason about and analyze the sensitivity of such strategies to perturbations in passenger demand distributions.

Finally, we address the problem of large-scale fleet management. Recent approaches for the fleet management problem have leveraged model-free deep reinforcement learning (RL) based algorithms to tackle complex decision-making problems. However, such methods suffer from a lack of explainability and often fail to generalize well. We consider an explicit *need-based* coordination mechanism to propose a non-deep RL-based algorithm that augments tabular Q-learning with a combinatorial optimization problem. Empirically, a case study on the New York City taxi demand enables a rigorous assessment of the value, robustness, and generalizability of the proposed approaches.

Contents

List of Tables

List of Figures

List of Abbreviations

A2C	.....	Advantage Actor-Critic Network
DQN	.....	Deep Q-Network
GDPR	.....	General Data Protection Regulation
MARL	.....	Multi-agent Reinforcement Learning
MDP	.....	Markov Decision Process
MoD	.....	Mobility-on-Demand
RL	.....	Reinforcement Learning

Chapter 1

Introduction

In the past century, unprecedented growth in urban population has caused a dramatic increase in the cost of a finite resource – urban space. The ever-increasing demand for congestion-free roads and parking spaces in dense urban environments has made private automobile ownership an unsustainable solution for an urbanite's transportation needs. However, in recent years, widespread adoption of smartphones, advancements in wireless communication technologies, coupled with tremendous growth in the computational power at our disposal have paved the way for a transformational change in urban mobility in the form of Mobility-on-Demand (MoD) services.

In MoD systems, transportation solutions can come in the form of fleets of cars, bikes, electric scooters, etc., managed by private companies, public entities, or even independent contractors. Built upon the bedrock of technology, such systems have revolutionized personal transit by providing an individual with the convenience and comfort of private automobiles without the associated high cost of vehicle maintenance, fueling, or parking problems. In addition to providing a reliable mode of mobility, they also offer a sustainable longer-term solution to external challenges such as greenhouse gas emissions, transit time uncertainties, and propensity for accidents. This dissertation focuses on understanding and resolving some of the key operational challenges – MoD fleet monitoring and management – faced during their deployment.

1.1 Rise of Mobility-on-Demand Systems

There are multiple kinds of MoD systems deployed in different cities across the world. All major cities across the globe typically have bike-sharing systems. In such cases, the MoD systems provide bike stations at closely spaced intervals throughout the city. When a user wishes to go somewhere, they walk to the nearest station, swipe a card to pick up a bike, ride it to the nearest station to their destination, and drop it off. Recently, station-less electric scooter MoDs such as Bird follow similar setups such as bike-sharing. However, the users are allowed to drop off the scooters anywhere near their destination. In a similar vein, car-sharing MoDs such as ZipCar, Enterprise Rentals, etc., offer a car rental with the caveat that the rented car has to be returned to its original pickup location.

In the last decade, the rise of MoD ride-hailing services such as Uber, Lyft, DiDi, Ola, etc., have disrupted the century-old taxi industry. Such services utilize smartphone-based applications to dispatch their vehicles to the location of a user. The emergence of automated vehicles provides opportunities for creating fleets of MoD vehicles with centralized control. Integration of automated vehicle fleets into existing MoD services such as ride-hailing is widely considered to be the future of urban vehicular transportation. Irrespective of the specific kind of MoD, the key factors determining the success of such systems are the costs to users and the wait times, i.e., the times needed to walk to/from the bike station or the time spent waiting for the driver pickup. Well-designed and well-managed MoD systems usually provide better combinations of costs and wait times than traditional alternatives such as taxis and mass transit systems.

1.2 Operational Challenges

MoD systems currently face a lot of challenges. Specifically, the spatio-temporal asymmetric nature of urban transportation demand leads to the evolution of *vehicle imbalances* over time. Vehicles tend to accumulate in some regions of the city, while they are depleted in others. Over time unless the operator of MoD undertakes a rebalancing operation, this uneven distribution of vehicles can be severely detrimental to the utilization of the fleet and the quality of service being provided to the users. Currently, different kinds of MoD systems typically address this challenge in different ways. Systems using smaller vehicles such as bikes or electric scooters typically employ a human-in-loop rebalancing operation wherein a pickup truck collects excess vehicles from one region of the city and drops them off at places experiencing vehicle deficit.

Ride-hailing services have deployed more sophisticated solutions in the form of *dynamic pricing*, in which mathematical models determine appropriate prices for each ride to ensure that cost-sensitive excess demand drops out. At the same time, the potential to increase their earnings incentivizes the human drivers of their fleet to relocate themselves to regions with no supply, thereby alleviating the imbalances. Thus, the optimal MoD system performance in terms of maximizing pickup vehicle availability across the city, minimizing user wait times, minimizing cost to the platform operator is predicated upon several factors such as:

- Real-time passenger demand data
- Size of the vehicle fleet
- Vehicle location data
- System rebalancing operations
- Dynamic pricing algorithms

– Driver-passenger matching algorithms

Inevitably, such a complex system with varied components gives rise to a number of different operational challenges, which have stoked the interest of researchers from different disciplines such as Computer Science, Economics, Management Science, Operations Research, etc. Moreover, as a growing number of algorithms used to power such complicated systems rely on non-deterministic techniques from Machine Learning, the issues of fairness of such algorithms towards human operators and passengers have been brought to light. The European Union's General Data Protection Regulation (GDPR), which came into effect in 2018, has imposed anti-discriminatory regulations on such algorithms, thereby drawing significant interest of the legal community towards MoD system operations.

1.3 Summary of Contributions

In this dissertation, we address multiple problems of theoretical and practical importance to the operations of MoD systems.

1.3.1 Fleet Monitoring

In Chapter 3, we focus on the problem of monitoring fleet distribution. MoDs with shared bikes typically do not have access to real-time location data of their entire fleet. The platform operators cannot track the bikes during transits and thus can only observe the spatial distribution of partial fleet at a time. Inability to accurately estimate the distribution of fleet introduces a time delay in the rebalancing operations. In this chapter, given an initial starting distribution of the fleet, we focus on minimizing the variance of estimated distribution in future timesteps. We do this by issuing queries that retrieve the precise information regarding exact number of vehicles departing a particular MoD station or traversing a specific road. Under bud-

get constraints on the maximum number of queries that can be issued, we propose efficient algorithms to minimize uncertainty in the fleet distribution. We demonstrate the efficacy of our algorithms on synthetic datasets as well as real dataset obtained from Hubway bike-sharing service in Boston.

1.3.2 Driver Revenue Maximization

In Chapter 4, we turn our attention towards the welfare of the independent driving partners on ride-hailing platforms. While ride-hailing platforms typically adopt a longer-term view and optimize the systems for the best quality of service to their customers, it can often lead to a suboptimal system with respect to driver earnings. In this chapter, we formalize the problem of devising strategies for maximizing the earnings of individual self-interested drivers. We augment the publicly available New York City taxi demand data with a newly collected real-time dataset from Uber API to analyze our proposed strategies. One of the fundamental challenges in using historical data to inform driver strategies is the potential for sub-optimal performance in the presence of perturbations in demand distribution in the future. Using techniques from the field of robust optimization, we alleviate this concern by performing a rigorous analysis on the sensitivity of our strategies to perturbations in observed data.

1.3.3 Fleet Management

Finally, in Chapter 5, we shift our focus to the problem of fleet management. While the strategies proposed in Chapter 3 will increase the earnings of individual drivers, they are not optimal when adopted by a large portion of the drivers in the fleet. Any approach towards fleet management needs to find an optimal tradeoff between maximizing driver earnings and maintaining a high demand satisfaction. Typically, coordination mechanisms for fleet management are learned using deep neural networks. Such black-box approaches suffer from a lack of explainability. Under GDPR

described above, algorithmic recommendations made by an MoD platform are mandated to be explainable to prevent discrimination. In pursuit of explainability, we design an explicit need-based coordination mechanism that allows drivers to coordinate only when required for a vehicle rebalancing operation and operate independently otherwise. Our approach combines classical techniques like model-based tabular Q-learning with reward maximizing combinatorial optimization problems to propose an efficient, scalable and robust framework that can be used to optimize either driver revenue or platform throughput. Furthermore, similar to driver revenue maximization strategies, we extensively study the generalizability of our framework with respect to hyperparameter tuning and changes in data distribution.

1.4 Related Publications

The work presented in this dissertation has been partially presented in three publications listed below.

Harshal A. Chaudhari, Michael Mathioudakis, and Evimaria Terzi. *Markov Chain Monitoring.* SIAM International Conference on Data Mining 2018

Harshal A. Chaudhari, John W. Byers, and Evimaria Terzi. *Putting Data in the Driver's Seat: Optimizing Earnings for On-Demand Ride-Hailing.* ACM International Conference on Web Search and Data Mining 2018.

Harshal A. Chaudhari, John W. Byers, and Evimaria Terzi. *Learn to Earn: Enabling Coordination Within a Ride-Hailing Fleet.* IEEE International Conference on Big Data 2020.

Chapter 2

Literature Review

Urban transportation has been transformed in the recent decade due to the emergence of MoD systems. Their success has been made possible due to advances in both wireless communication technology and algorithmic research on the interesting problems and challenges. There is a vast amount of academic and industrial literature covering several levels of planning involved in the deployment of MoD systems, from strategic to operational planning. Locating shared-vehicle stations, choosing the number of vehicles per station, moving vehicles between stations, encouraging users to change their destinations, etc., are some of the management challenges to be resolved to provide a high quality of service. The development of effective tools to guide such decisions provides important motivation to researchers across different disciplines of science. In this chapter, we position this dissertation in the context of other related works on similar problems and fields of study.

2.1 Bike-sharing Systems

While the oldest public bike-sharing systems have been tested in Europe since the 1960s, the concept experienced a dramatic improvement with advancements in wireless communication technology. As of December 2016, roughly 1,000 cities worldwide have a bike-sharing program. This has led to a significant amount of research interest across different domains of bike-sharing – station location, fleet sizing, station sizing, rebalancing incentives, and vehicle repositioning – which we discuss in this

section [94]. However, we refer the reader to Laporte *et al.* [47] for a more comprehensive survey on the state-of-the-art literature about bike-sharing. We should note that bike-sharing literature is often relevant to other MoDs such as shared-car rental systems, which are also included in this section.

2.1.1 Station Location

The success of a bike-sharing system relies predominantly on two factors, viz., (1) stations located at convenient walking distance from the popular origin and destination spots in the city (2) availability of bikes at these stations. Some of the earliest works by Lin and Yang *et al.* [55] and Lin *et al.* [56] on station location problem for bike-sharing formulate it as a joint non-linear optimization problem considering decision variables such as the number of stations, user flow, bike lanes, etc.

Some researchers have used real-life data – Martinez *et al.* [60] simultaneously optimizes station locations, size of the fleet, and rebalancing operation in Lisbon. In contrast, Chow and Sayarshad [20] use a game-theoretic approach to evaluate the impact of shared-transportation network design in Toronto. Regarding one-way car rental systems, Kumar and Bierlaire [46] study the problem of locating electric car-sharing stations in the city of Nice, while Li *et al.* [53] uses a case study of Sioux Falls in North Dakota to explore a similar problem.

2.1.2 Fleet Size and Station Inventory

Queuing-theoretic approaches have been used to analyze the effect of fleet size and station capacity on overall system performance by multiple studies such as that of George and Xia [31], Fricker and Gast [25], etc. Nair and Miller-Hooks [64] also addressed the problem of setting the proper initial inventory of bikes at each station. Raviv and Kolka [72] devised an approximation method to minimize user dissatisfaction by determining the correct number of bikes at each station. Vogel *et al.* [88] has

approached the vehicle imbalance problem through the lens of determining the right inventory size for each station.

2.1.3 Vehicle Rebalancing

As noted in Chapter 1, the need to rebalance stations by redistributing the fleet vehicles across the city is pivotal to the success of an MoD system. Incentives could be provided to users to pick up vehicles from stations with ample supply and drop them off at stations with low inventory vehicles. Chemla *et al.* [15], Pfrommer *et al.* [70], and Waserhole [90] have studied dynamic pricing models for bike-sharing where the price paid by users depends on the current state of the system.

An alternate approach to vehicle rebalancing involves the use of human-operated trucks to reposition bikes. Raviv *et al.* [72] and Schuijbroek *et al.* [76] use a time-indexed model and queuing theory to compute the optimal inventory size required to maintain service levels. Later, they route the repositioning truck between stations to rebalance the bikes, taking into account the capacity constraints of the truck itself. Benchimol *et al.* [3] adopt a more theoretical view of the rebalancing problem and propose an approximation algorithm to minimize routing cost. Numerous other works [44, 10, 54, 24, 23, 21] have studied the problem of vehicle rebalancing under different settings with constraints based upon routing costs, unmet demand, number of vehicles repositioned, number of truck stops, etc.

A common theme across all the literature related to bike-sharing systems is the assumption of perfect knowledge regarding the exact spatiotemporal distribution of the fleet of bikes. Such an assumption contrains the problem formulations to be static in nature. For example, they assume that vehicle rebalancing operation occurs at a time when users are not actively interacting with the system. Any dynamic problem formulation needs to account for the fleet's movement in real-time, thereby creating an additional difficulty in knowing the exact spatiotemporal distribution of the fleet.

Our work on fleet monitoring aims to address this difficulty in estimating the fleet distribution at any given time. The *uncertainty* in fleet distribution, as defined in Chapter 3, can be used to augment the optimization problems in the above works to reduce their reliance on perfect knowledge of the system.

2.2 Ride-sharing Systems

Ride-sharing systems are a collection of websites and smartphone applications that match passengers with drivers of vehicles for hire. While rudimentary ride-matching programs have existed since the late 1990s, a report by the Federal Transit Administration of the United States Department of Transportation [39] stated that dynamic ridematching in real-time had not yet been successfully implemented in 2006. However, the advent of online ride-matching platforms such as Uber and Lyft in the United States, Ola, Didi Chuxing, etc., in Asia rejuvenated the ride-sharing industry by the late 2010s. The mode of operation of ride-sharing systems, wherein a single passenger is matched with a driver, is commonly known as *ride-hailing*. While the problems related ride-sharing systems appear under different names in the literature, viz., order dispatching, order-driver assignment, etc., they usually refer to an online bipartite matching problem where both supply of vehicles and demand are dynamic, with uncertainty arising from spatiotemporal variations in both supply and demand. Now, we describe some of the prominent works studying different facets of the ride-sharing systems.

2.2.1 Taxi Fleet Optimization

A considerable body of work has focused on optimizing taxi fleets, for example building economic network models to describe demand and supply equilibria of taxi services under various tariff structures, fleet size regulations, and other policy alternatives [1, 97]. Other work seeks to optimize the allocation of taxi market resources [78].

Another direction focuses on route optimization by a centralized administrator (e.g., taxi dispatching services) [59, 66] or maximizing occupancy and minimizing travel times in a shared-ride setting [42]. Other work has studied the supply side of the driving market from the viewpoint of behavioral economics. A seminal paper by Camerer *et al.* [12] studied cab drivers and found that inexperienced cab drivers (1) make labor supply decisions "one day at a time" instead of substituting labor and leisure across multiple days, and (2) set a loose daily income target and quit working once they reach that target.

2.2.2 Vehicle Repositioning

The problem of spatiotemporal demand prediction to inform an individual taxi driver of favorable locations for passenger pickups had been studied extensively even before the advent of ride-hailing services. These studies typically assume that overall system dynamics are agnostic to the choices of individual drivers. We can view these studies from the lens of developing recommender systems for individual drivers. Alongside individual earnings, some other common objectives to be optimized include total cruising distance [29], vehicle utilization, i.e., the ratio of passenger trip mileage to idle cruising mileage [28], individual trip fares [74], etc. Li *et al.* [51] use a large-scale taxi GPS trace dataset to identify salient features associated with successful passenger pickup locations. In two separate studies, Yuan *et al.* [98, 99] develop a recommender system to guide both idle taxi drivers and waiting passengers to convenient locations in order to optimize social welfare. More recently, Özkan and Ward [68] looked at strategic matching between supply (individual drivers) and demand (requested rides) for Uber and Lyft. Initial works focused on optimizing the objective up to the next successful passenger trip, while recently, the focus has shifted to optimization over a longer horizon.

Most of the works above are relatively diverse in specifics of the approach, but

they broadly adopt value-based reinforcement learning approaches to solve Markov Decision Processes (MDP). A few, such as Verma *et al.* [85] propose Monte Carlo learning, while more recent work by Wen *et al.* [91] use Deep Q-Networks (DQN).

In Chapter 4, we devise driver-oriented strategies to recommend favorable driving schedules and pickup locations to optimize the earnings of an individual on-demand ride-sharing driver. While our work tackles the problem of an individual self-interested strategic driver, it notably differs from a typical taxi routing literature because it takes into account elements from ride-hailing platforms such as flexibility of schedule, surge pricing, etc.

2.2.3 Capacity Repositioning

A major limitation of the recommender systems developed for individual drivers is that they are agnostic to driver interactions and may result in unfavorable supply excesses in certain locations when adopted by many drivers simultaneously. Platform level optimization for objectives such as platform revenue maximization, vehicle utilization, demand fulfillment, etc., requires a robust system-level coordination mechanism.

While traditional works in driver dispatch systems [48, 101, 77] typically relied on queuing-theoretical fluid models to asymptotically optimize supply-demand matching, numerous recent works [52, 41, 102] leverage advances in Multi-agent Reinforcement Learning (MARL) to directly optimize matching for multiple drivers simultaneously. Some works [95, 100] have attempted to leverage demand volume and ride destination forecasting models to use in combinatorial optimization techniques. However, these approaches do not scale well to use cases on contemporary platforms, where fleets of as many as 10,000 drivers serve a single city. Lin *et al.* [57], Wang *et al.* [89] and Tang *et al.* [81] have attempted to bypass the complication of optimizing joint actions in the multi-agent setting by simply using a single value function aggregated over all

the drivers' data. We refer the reader to Qin *et al.* [71] for a comprehensive survey on different reinforcement learning-based approaches for supply-demand matching.

While providing the advantage of simplicity, in Chapter 5, we show that model-free reinforcement learning-based methods often fail to generalize well. Moreover, deep-learning-based techniques suffer from a lack of explainability. They often require highly specialized proprietary model inputs to aid in the learning process. Cognizant of these issues, our approach does not rely on any proprietary models but instead learns high-quality solutions *from scratch* based solely upon historically observed data. Moreover, we achieve that without sacrificing the explainability of the model. In the absence of the need for coordination, our model assumes homogeneity of drivers in the same location and provides *envy-free* recommendations while also making it scalable. This is achieved by augmenting vanilla reinforcement learning (in the form of tabular Q-learning) with combinatorial techniques to aid the rebalancing of driver distribution.

2.2.4 Platform Studies

Studies of ride-hailing services as multi-sided economic marketplaces have investigated the impacts of the platform's pricing policies on the platform profits, the consumer surplus, and the driver wages [13, 5, 4, 58]. The popular press has investigated the supply-side effects of specific incentives (e.g., surge pricing) that Uber and Lyft provide to drivers [84]. Chen and Sheldon [19] showed a causal relationship that drivers on Uber respond to surges by driving more during high surge times, differentiating from previous work that suggests taxi drivers primarily focus on achieving earnings goals [12]. Hall and Krueger [37] showed that drivers were attracted to the Uber platform due to the flexibility it offers and the level of compensation, but earnings per hour do not vary much with the number of hours worked. In another line of research, Chen *et al.* [18] measured many facets of Uber in New York City, including the

prevalence and extent of surge pricing. Castillo *et al.* [13] showed that surge pricing is responsible for effectively relocating drivers during high-demand periods, thereby preventing them from engaging in 'wild goose chases' to pick up distant customers, which only exacerbates the problem of low driver availability.

In another line of work, Banerjee *et al.* [2] studied dynamic pricing strategies for ride-hailing platforms using a queuing-theoretic economic model. They showed that dynamic pricing is robust to changes in system parameters, even if it does not achieve higher performance than static pricing. Sühr et al. [79] investigate fairness in driver earnings distribution using driver-passenger matchings optimized to attain income equality goals. Recently, Chen et al. [16] combines platform economics with the capacity repositioning problem using a contextual bandit framework. They showed that matching based on time-varying parameters like driver and customer arrival rates and the willingness of customers to wait could achieve better performance than naively matching passengers with the closest driver. Although these works build attractive models for ride-hailing economies, they are orthogonal to the contents of this dissertation, as they take a holistic view of such economies, while we focus on specific supply-demand matching problems.

There is a growing body of literature studying the interplay between platform pricing and strategic driver behaviors, for which we refer the readers to Yan *et al.* [96]. Our work contributes to this domain by developing a scalable framework that can be used to verify the results of asymptotic dynamic pricing models via realistic simulations.

Chapter 3

Fleet Monitoring

Ever since their first deployment in Paris in 2007, modern bike-sharing systems have proliferated across the globe and transformed urban transit. Roughly 1,000 cities worldwide have a bike-sharing program, with some of the most extensive programs being operated in China, with fleets comprised of close to 100,000 bikes [94]. The explosive growth of urban bike-sharing programs has led to numerous academic and industrial research studies exploring problems at the intersection of multiple acadmeic disciplines such as Operations Research, Economics and Computer Science, etc. Operations Research community focuses on practical problems such as choosing appropriate locations for bike-sharing stations and determining the correct inventory levels at these stations to satisfy the demand. Economists design dynamic pricing algorithms and determine appropriate customer incentive mechanisms to maintain a high quality of service. Computer scientists have concentrated their efforts on developing efficient algorithms to rebalance bike fleets across the city. While considering such platform challenges, these studies generally assume perfect knowledge about the spatial distribution of the fleet of shared bikes. However, even today, most bike-sharing programs do not typically track the location of their bikes in real-time. Until recently, the platforms only became aware of the updated locations of the bikes when they were docked into stations at the end of the ride.

In this chapter, we formalize the problem of minimizing the uncertainty in the spatial distribution of a bike-sharing MoD fleet with access to only partial information

regarding its distribution. It should be noted that in a discrete time setting, the fleet distribution at any point in time depends only upon its distribution in the immediately preceeding timestep and the passenger demand. In other words, the fleet distribution is appropriately modeled by a *memoryless* stochastic process in the form of a *Markov Chain* in which shared bikes traverse along the edges of a graph representation of a city. While the expected distribution of bikes can be easily estimated using higher powers of the corresponding transition matrix from the Markov Chain, we seek to identify a limited number of nodes (or edges) so that, once we know exactly how many bikes reside on (resp. traverse) them, the uncertainty in the spatial distribution of the fleet is minimized. Henceforth, we refer to this objective as the Markov Chain Monitoring problem. Indeed, solutions to this problem can have widespread applications in domains beyond bike-sharing MoDs, viz., urban traffic networks, peer-to-peer computer networks, postal and freight networks, etc. In all of these settings, one wishes to have an accurate estimate of the number of items that reside at various nodes of a network (e.g., vehicles that are at the intersections of a road network) but faces constraints on how big a part of the network (e.g., intersections or road segments) they can monitor at any time.

We make two assumptions in our problem definition: (1) there is a point in time when we have an accurate estimate of the placement of all items on the graph (2) we can monitor the subsequent placement of items by issuing a predefined number of *monitoring operations* (i.e., real-time queries on the Markov chain). We consider different types of operations: ones that retrieve the number of items that reside on specific nodes; and ones that retrieve the number of items that traverse specific edges. In bike-sharing MoD systems, the queries correspond to information regarding the number of bikes docked in a particular station or bike rides between a pair of stations. The concept generalizes to applications such as urban and computer traffic networks,

wherein the queries correspond to placing measurement (or monitoring) devices on particular nodes or edges of the network and retrieving their measurements.

Technically, different monitoring operations result in different variants of Markov Chain Monitoring. For example, monitoring of the items that pass through an edge leads to a different problem definition than monitoring of the nodes. For each variant, we design efficient polynomial-time algorithms. For some of these algorithms, we demonstrate that they are optimal, while for others, we show that they perform well in practice. Our experiments use a diverse set of datasets from urban networks and computer networks to demonstrate the practical utility of our setting. We then exemplify our approaches using data from the Hubway bike-sharing network of Boston (rebranded as BlueBikes) and identify candidate monitoring stations. The monitoring operations chosen by the algorithms encompass stations in close proximity to downtown Boston or university student housings and are well supported by anecdotal experiences.

Chapter Organization

The rest of the chapter is organized as follows. First, in Section 3.1, we place the fleet monitoring problem in the context of other graph centrality measures. We formally define the Markov Chain Monitoring problem and its variants in Section 3.2. Next, in Sections 3.3-3.4, we provide efficient algorithms to solve each of the variants. Finally, in Section 3.5, we evaluate the performance of our proposed algorithms on various real and synthetic datasets. Notably, we use the `Hubway` dataset to showcase their use on a fleet monitoring operation on a bike-sharing MoD system.

3.1 Background

To the best of our knowledge, we are the first to introduce and study the Markov Chain Monitoring problem. However, this problem is similar to tasks such as

outbreak detection, sparsification of influence network, and the broader field of node and edge *centrality* on graphs, where one seeks to identify k "central" nodes or edges to intercept the movement of items on a graph to contain the spread. A major difference in our definition of the Markov Chain Monitoring is that the "centrality" of nodes or edges is based not just on the underlying graph structure but also on the dynamic propagation of items through the network. Nevertheless, in this section, we discuss some of the existing works on graph centrality measures.

Graph centrality measures can be broadly cast into two categories: *individual* and *group centrality* measures. Individual measures assign a score to each node or edge. Group centrality measures assign scores to sets of nodes or edges. Usually, computing group-centrality measures requires solving complex combinatorial problems.

Examples of individual centrality scores for nodes and edges are the *Pagerank* [69], betweenness [7, 22, 73], and current flow centralities [8]. Pagerank is one classic example of a centrality measure based on a Markov chain – where the centrality of nodes is quantified as the stationary distribution of a Markov chain on the graph. Betweenness centrality and current flow centrality assign high centrality scores to nodes/edges that participate in one or many shortest paths between all pairs of nodes in the input network. Although a Markov chain is used for Pagerank, its computation is very different from ours – after all, Pagerank is an individual centrality measure, while our measures are group centralities.

Group centrality measures that use a Markov chain model include the *Absorbing Random Walk Centrality* introduced by Mavroforakis *et al.* [62]. In their work, the centrality of a set of nodes is defined as the transient time of the Markov chain when the given nodes are "absorbing". The work of Gionis *et al.* [32] also assumes an absorbing Markov chain. In that setting, the authors aim to maximize the positive opinion of the network concerning a specific topic by picking k nodes appropriately

to endorse this positiveness via posts (e.g., in a social network). Both these problems are different from ours because we do not consider absorbing random walks but rather a simple Markov chain. Moreover, our objective to minimize uncertainty is different from the objectives of the above papers.

While Markov chain is a simple model that quantifies centrality, different models may be more appropriate in other settings. For example, Ishakian *et al.* [40] proposed an extension of betweenness centrality called group betweenness centrality. They aim to find a group of nodes with have maximum number of shortest paths passing through them. In epidemiology and information propagation settings, the underlying process of interest, spread of a virus or a piece of information, is better modeled as a random cascade. Centrality of a node attempts to capture the notion of its *i*nfluence, i.e., the expected size of the random cascade beginning at that node [6, 26, 35, 43, 67, 82]. Two common applications of this methodology are those of outbreak detection [50] and graph sparsification [61]. The first one tasks a practitioner with identifying central nodes that can intercept an evolving cascade as soon as possible, while the latter one seeks to identify edges that are cruicial towards the explanation (from a model-sparsity point of view) of an observed cascade.

In terms of categorization, the Markov Chain Monitoring problem is a group-centrality measure. The distinguishing factor of the Markov Chain Monitoring problem in comparison to the aforementioned works is that we define centrality in terms of a combination of a static graph structure, a dynamic Markov chain, and a real-time activity in the form of moving items.

3.2 Problem Setup

In this section, we introduce our setting and notation.

Markov Chain

Consider a weighted directed graph $G = (V, E, p)$ with $|V| = n$ and $|E| = m$. We use $\pi(v) \subseteq V$ and $\kappa(v) \subseteq V$ to denote the set of parent and child nodes of node v, respectively.

$$\pi(v) = \{u \mid u \in V, e(u \to v) \in E\}$$
$$\kappa(v) = \{u \mid u \in V, e(v \to u) \in E\}$$

Edges $e(u \to v) \in E$ are associated with real-valued weights $p(u \to v) \in [0, 1]$ such that $\forall u \in V$: $\sum_{v \in V} p(u \to v) = 1$.

These weights give rise to a Markov chain with transition matrix $\mathbf{P}$ – where $p(u, v)$ denotes the probability of a transition from node u to node v. Moreover, we assume that a set of items are distributed among the nodes of G. We use the row vector $\mathbf{x}$ to denote the the *initial* number of items per node; that is, $x(u)$ is the number of items initially at node u. For the entirety of this chapter, transition matrix $\mathbf{P}$ and distribution $\mathbf{x}$ are assumed known and part of the input.

Consider now a single step of the Markov chain. During this step, each and every item transitions from the node u where it resides originally to another node v, according to transition probabilities $p(u, v)$. At the end of this step, items are redistributed among the nodes - and we are no longer certain about their position. We use $\mathbf{z}$ to denote the random vector with the number of items at each node after one transition. The expected number of items at each node is given by: $E[\mathbf{z}] = \mathbf{xP}$.

Quantifying uncertainty

We quantify uncertainty in terms of the variance in the number of items on each node afer the transition step. Specifically, let us consider the number $z(v)$ of items on node u after the transition step: an item previously at node u will transition to

node v according to a Bernoulli distribution with success probability $p(u, v)$; and since items transition from node to node independently from each other, the variance in the number of items $z(v)$ is

$$\begin{aligned}\operatorname{var}(z(v)) =& \operatorname{var}(z(v)|\mathbf{P}, \mathbf{x}) \\ =& \sum_{u \in V} x(u)p(u, v)\left(1 - p(u, v)\right). \end{aligned} \tag{3.1}$$

To obtain an aggregate measure of uncertainty F_0, we opt to sum the aforementioned quantity over all nodes.

$$\begin{aligned} F_0 &= \sum_{v \in V} \operatorname{var}(z(v)) \\ &= \sum_{u \in V} x(u) \sum_{v \in V} p(u, v)\left(1 - p(u, v)\right) \end{aligned} \tag{3.2}$$

Monitoring

Given the transition matrix $\mathbf{P}$ and the initial distribution of items $\mathbf{x}$, we estimate the distribution of items $\mathbf{z}$ after one transition step, with the uncertainty given in Equation (3.2). After one transition step, we are allowed to retrieve information about the position of the items and thus reduce uncertainty. We do this by performing "monitoring operations", i.e. queries on the position of items on the Markov chain. These operations are of the following types:

- ParentTransitions Retrieve the number of items that transitioned to node v from each $u \in \pi(v)$;
- NodeItems Retrieve the number of items that reside on node v after the transition step;
- EdgeTransitions Retrieve the number of items that transitioned from node u to node v;

- ChildrenTransitions Retrieve the number of items that transitioned from node u to each child $v \in \kappa(u)$.

From the above four types of monitoring operations, the last one (i.e., ChildrenTransitions) leads to trivial combinatorial problem. Thus, we omit it from the rest of the discussion.

Expected uncertainty

Once we retrieve the answer A to a set of monitoring operations, we have more information about the positioning of items over nodes V – and thus an updated (and non-increased) uncertainty

$$F(A) = \sum_{v \in V} \text{var}(z(v)|A).$$

In the setting we consider, however, the challenge we face is not to compute the uncertainty given the information retrieved via a monitoring operation, but rather to select the monitoring operations so that the uncertainty faced after retrieving A is minimized in expectation. Therefore, the quantity of interest is that of *expected uncertainty* for a set of operations that we choose to perform, expressed as $E[\sum_{v \in V} \text{var}(z(v)|A)]$.

Let us now assume we have chosen to perform a set of operations of either one of the aforementioned types. In what follows, we provide formulas for the *expected uncertainty* in each case.

Expected uncertainty under ParentTransitions

We perform monitoring operations for a subset $S \subseteq V$ of nodes – and obtain an answer $A_{PT}(S) = \{n_{uv}; v \in S,\ e(u \to v) \in E\}$, where n_{uv} is the number of transitions to v from its parent node u. The expected value $F_{PT}(S)$ of the uncertainty $F(A_{PT}(S))$

after these operations is given by

$$\begin{aligned} F_{PT}(S) =& E[F(A_{PT}(S))] \\ =& \sum_{u \in V} x'(u) \sum_{v \in V \setminus S} p'(u,v)\,(1 - p'(u,v)) \end{aligned} \tag{3.3}$$

where

$$\rho(u,S) = \sum_{v \in S} p(u,v)$$

$$x'(u) = x(u)\,(1 - \rho(u,S)) \tag{3.4}$$

$$p'(u,v) = \frac{p(u,v)}{1 - \rho(u,S)}. \tag{3.5}$$

Intuitively, $x'(u)$ expresses the expected number of items that transition from u to nodes v *other than* those in S; and $p'(u,v)$ expresses the probability an item transitions from u to v *given* that it does *not* transition to those in S. We see, then, that Equation (3.3) has the same form as Equation (3.2) but is evaluated on adjusted values of $\mathbf{x}$ and $\mathbf{P}$, to take into account the information we obtain via A_{PT}.

Note that the expected uncertainty after the monitoring operations is no larger than F_0.

Lemma 1. *The information retrieved via* ParentTransitions *monitoring operations decrease the expected uncertainty about the positioning of items after the transition step.*

$$F_{PT}(S) \leq F_0$$

Proof. Using Equations (3.4) and (3.5) to expand Equation (3.3) we have

$$\begin{aligned} F_{PT}(S) &= \sum_{u \in V} x'(u) \sum_{v \in V \setminus S} p'(u,v)\,(1 - p'(u,v)) \\ &= \sum_{u \in V} x(u) \sum_{v \in V \setminus S} p(u,v) \left(1 - \frac{p(u,v)}{1 - \rho(u,S)}\right) \\ &\leq \sum_{u \in V} x(u) \sum_{v \in V \setminus S} p(u,v)\,(1 - p(u,v)) \\ &= F_{PT}(\emptyset). \end{aligned}$$

□

Expected uncertainty under NodeItems

We perform monitoring operations for a subset $S \subseteq V$ of the nodes – and obtain an answer $A_{NI}(S) = \{n_v; v \in S\}$, where n_v is the number of items at node v after the transition. For an instance of an answer $A_{NI}(S)$, let also $A_{PT}(S) = \{n_{uv}; v \in S, e(u \to v) \in E\}$ be an answer for ParentTransitions on the same set S of nodes. By definition $A_{NI}(S)$ can be computed from $A_{PT}(S)$ using the relationship

$$n_v = \sum_{\substack{u \in V \\ e(u \to v) \in E}} n_{uv}, \forall v \in S. \tag{3.6}$$

Hence, we use the notation $A_{PT}(S) \simeq A_{NI}(S)$ to denote this interchangeability between the answers of two different kinds of monitoring operations. It can be shown that the *expected uncertainty* is equal for the two cases. That is: $F_{PT}(S) = F_{NI}(S)$.

Theorem 1. *For the same set of monitored nodes,* ParentTransitions *and* NodeItems *lead to the same value of the objective function.*

$$F_{NI}(S) = F_{PT}(S)$$

Proof. We express $F(A_{NI})$ in terms of $F(A_{PT})$ as follows.

$$F(A_{NI}) = \sum_{A_{PT} \simeq A_{NI}} F(A_{PT}) \Pr(A_{PT}|A_{NI}) \tag{3.7}$$

We can now use the above equation to express the expected uncertainty $F_{NI}(S)$ in terms of $F_{PT}(S)$ as:

$$\begin{aligned}
F_{NI}(S) &= E[F(A_{NI})] \\
&= \sum_{A_{NI}} F(A_{NI}) \Pr(A_{NI}) \\
&= \sum_{A_{NI}} \sum_{A_{PT} \simeq A_{NI}} F(A_{PT}) \Pr(A_{PT}|A_{NI}) \Pr(A_{NI}) \\
&= \sum_{A_{NI}} \sum_{A_{PT} \simeq A_{NI}} F(A_{PT}) \Pr(A_{PT}, A_{NI}) \\
&= \sum_{A_{NI}} \sum_{A_{PT} \simeq A_{NI}} F(A_{PT}) \Pr(A_{PT}) \\
&= \sum_{A_{PT}} F(A_{PT}) \Pr(A_{PT}) \\
&= F_{PT}(S),
\end{aligned} \tag{3.8}$$

which concludes the proof. □

Expected uncertainty under EdgeTransitions

We perform monitoring operations for a subset $D \subseteq E$ of the edges – and obtain an answer $A_{ET} = A_{ET}(D) = \{n_e; e \in D\}$, where n_e is the number of transitions over edge e. The expected value $F_{ET}(D)$ of the uncertainty $F(A_{ET}(D))$ after these operations is given by

$$\begin{aligned}
F_{ET}(D) =& E[F(A_{ET})] \\
=& \sum_{u \in V} x''(u) \sum_{e(u \to v) \in E \setminus D} p''(u,v) \left(1 - p''(u,v)\right)
\end{aligned} \tag{3.9}$$

where

$$\rho(u, D) = \sum_{e(u \to v) \in D} p(u, v)$$

$$x''(u) = x(u)\,(1 - \rho(u, D)) \tag{3.10}$$

$$p''(u, v) = \frac{p(u, v)}{1 - \rho(u, D)} \tag{3.11}$$

Similar to ParentTransitions and NodeItems, expected uncertainty $F_{ET}(S)$ is no greater than F_0.

3.2.1 Problem Statement

The general problem of Markov Chain Monitoring is to select the appropriate monitoring operations to reduce the expected uncertainty after they are performed. Stated formally:

Problem 1 (Markov Chain Monitoring)**.** *Given a transition matrix P and an initial distribution of items $\mathbf{x}$, select a set of up to k monitoring operations to minimize the expected uncertainty F.*

We study variants of the problem – each defined for a specific type of monitoring operation. For simplicity, we refer to these problems with the same name as that of the operation type: ParentTransitions, NodeItems, ChildrenTransitions, and EdgeTransitions.

Furthermore, as we saw in Section 3.2, variants ParentTransitions and NodeItems are equivalent: for the same set of nodes, operations of the first type reduce expected uncertainty as much as the second. Therefore, in what follows, we treat only the variant of NodeItems, as our claims apply directly to ParentTransitions as well.

3.3 Greedy Algorithm for Node-Monitoring

In this section, we provide the formal problem definition of the NodeItems problem variant and describe a greedy polynomial-time algorithm for solving it.

Problem 2 (NodeItems)**.** *Given $G = (V, E)$, transition matrix $\mathbf{P}$, initial distribution of items to nodes $\mathbf{x}$ and integer k, find $S \subseteq V$ such that $|S| = k$ such that $F_{NI}(S)$ is minimized.*

A brute-force way to solve Problem 2 would be to evaluate the objective function over all node-sets of size k. Obviously such an algorithm is infeasible as the runtime is exponential in k. Thus, we study a natural greedy algorithm for the problem, namely `NodeGreedy`.

The `NodeGreedy` algorithm

This is a greedy algorithm that performs k iterations; at each iteration, it adds one more node in the solution. If S^t is the solution at iteration t, then solution S^{t+1} is constructed by finding the node $u \in V \setminus S^t$ such that:

$$v^* = \arg\min_{v \in V \setminus S^t} F_{NI}\left(S^t \cup \{v\}\right). \tag{3.12}$$

Although in the majority of our experiments that compare the brute-force solutions with those of `NodeGreedy` the two solutions were identical, we identified some contrived instances for which this was not the case. Thus, `NodeGreedy` is not an optimal algorithm for Problem 2.

Running time

`NodeGreedy` evaluates Equation (3.12) at each iteration. A naive implementation of this would require computing Equation (3.3) $O(|V|)$ times per iteration, each time using $O(|V|^2)$ numerical operations. As a first improvement, we avoid the full double

summation over V via a summation over edges E,

$$\begin{aligned} F_{PT}(S) &= \sum_{u \in V} x'(u) \sum_{v \in V \setminus S} p'(u,v)\left(1 - p'(u,v)\right) \\ &= \sum_{\substack{(u,v) \in E \\ v \in V \setminus S}} x'(u) p'(u,v)\left(1 - p'(u,v)\right), \end{aligned} \tag{3.13}$$

that involves $O(k|V||E|)$ numerical operations.

Clearly, the `NodeGreedy` is infeasible to run even for small-size datasets over dense graphs, as the runtime approximately translates to $O(k|V|^3)$. We can further speed-up the algorithm if we re-use at each step the computations done in the previous one. To see how, let S_t (resp. S_{t+1}) be the solution we construct after t (resp. $(t+1)$) iterations and let v^* be the node such that $S_{t+1} = S_t \cup v^*$. Then, for any $u \in V$ we have $\rho(u, S_t) = \sum_{v \in S_t} p(u,v)$, and therefore

$$\rho(u, S_{t+1}) \quad = \rho(u, S_t) + p(u, v^*). \tag{3.14}$$

Moreover, for any $S \subseteq V$ let

$$\begin{aligned} B(u,S) = & \sum_{\substack{(u,v) \in E \\ s.t.\ v \in V \setminus S}} p'(u,v)\left(1 - p'(u,v)\right) \\ = & \sum_{v \in V \setminus S} \frac{p(u,v)}{1 - \rho(u,S)} \left(1 - \frac{p(u,v)}{1 - \rho(u,S)}\right). \end{aligned} \tag{3.15}$$

We can then express $B(u, S_{t+1})$ in terms of $B(u, S_t)$:

$$B(u, S_{t+1}) = B(u, S_t) - 2p(u, v^*)\left(1 - \rho(u, S_t) - p(u, v^*)\right). \tag{3.16}$$

Finally, using Equations (3.14) and (3.16) and algebraic manipulations, we can express

$F_{NI}(S_{t+1})$ as follows:

$$F_{NI}(S_{t+1}) = \sum_{u \in V} x(u) \left(\frac{B(u, S_t)}{1 - \rho(u, S_{t+1})} - 2p(u, v^*) \right)$$

Thus, if we store $B(u, S_t)$ and $\rho(u, S_t)$ at iteration t, then evaluating Equation (3.17) at iteration $t + 1$ takes only $O(|V|)$ numerical operations.

For all iterations but the first one, the above sequence of rewrites enables us to achieve a speedup from $O(|V||E|)$ to $O(|V|^2)$ numerical operations per iteration. For the first iteration, initializing the auxiliary quantities $B(u, \emptyset)$, $u \in V$, still takes $O(|E|)$. With this book-keeping, the running time of `NodeGreedy` is reduced from $O(k|V||E|)$ to $O(|E| + k|V|^2) = O(k|V|^2|)$. Note also that `NodeGreedy` is amenable to parallelization, as, given the auxiliary quantities from the previous step, we can compute the objective function independently for each candidate node.

3.4 Algorithms for Edge-Monitoring

Whereas NodeItems (Problem 2) seeks k nodes to optimize expected uncertainty, EdgeTransitions seeks k edges.

Problem 3 (Edge-Monitoring)**.** *Given $G = (V, E)$, transition matrix $\mathbf{P}$, initial distribution of items to nodes $\mathbf{x}$ and integer k, find $S \subseteq V$ such that $|S| = k$ such that $F_{ET}(S)$ (Equation (3.9)) is minimized.*

We provide two polynomial-time algorithms to solve the problem, namely `EdgeDP` and `EdgeGreedy`. For the former, we can also prove that it is optimal, and thus Problem 3 is solvable in polynomial time.

3.4.1 The `EdgeDP` Algorithm

`EdgeDP` is a dynamic-programming algorithm that selects edges in two steps: first, it sorts the *outgoing* edges of each node in decreasing order of transition probability,

thus creating $|V| = n$ corresponding lists; secondly, it combines top edges from each list to select a total of k edges.

In more detail, let $SOL_i(k)$ be the cost of an optimal solution for the special case of a budget of k edges allocated among outgoing edges of nodes $V_{i:} = \{i, i+1, \ldots, n\}$. According to this notational convention, the cost of an optimal solution D_{opt} for the problem is given by $SOL_1(k)$. Moreover, considering Equation (3.9), let F_i be the function that corresponds to the (outer) summation term for node i

$$F_i(D) = x''(i) \sum_{e(i \to v) \in E \setminus D} p''(i,v)\,(1 - p''(i,v)) \tag{3.17}$$

(under the auxilliary definitions of Equations (3.10) and (3.11)) and $ISOL_i(m)$ its optimal value when D contains no more than $m \leq k$ outgoing edges from node i. Let also D_i^m be a subset of k outgoing edges of i with the highest transition probabilities. It can be shown that the optimal value for $F_i(D)$ is achieved for the edges D_i^m with *highest* transition probability.

Lemma 2. *With choice restricted among the outgoing edges of a node, the optimal objective value in the* EdgeTransitions *setting is obtained for the edges of highest transition probability.*

$$ISOL_i(m) = F_i(D_i^m)$$

Proof Sketch. The optimization function is proportional to the following quantity:

$$f(E) \propto \sum_{i \in D_u(E)} p_i - \frac{\sum_{i \in D_u(E)} p_i^2}{\sum_{i \in D_u(E)} p_i} \tag{3.18}$$

where $D_u(E)$ are the remaining (i.e., non-queried) outgoing edges of parent-node u.

Consider two sets of edges E_0, $E_1 \subseteq \kappa(u)$ of the same size, all outgoing from a single parent-node u, that differ only at one element. The probabilities of the corresponding sets of remaining edges are:

$$D_u(E_0) : \{p_0\} \cup C; \quad D_u(E_1) : \{p_1\} \cup C \tag{3.19}$$

where $p_0, p_1 \notin C$, and without loss of generality $p_0 \leq p_1$.

Let $S = \sum_{i \in C} p_i$ and $SS = \sum_{i \in C} p_i^2$. We take the difference of the optimization functions for the two sets E_0 and E_1.

$$f(E_0) - f(E_1) \propto p_0 - p_1 - \frac{\sum_{i \in D_u(E_0)} p_i^2}{\sum_{i \in D_u(E_0)} p_i} + \frac{\sum_{i \in D_u(E_1)} p_i^2}{\sum_{i \in D_u(E_1)} p_i}$$
$$\propto -(p_1 - p_0)\frac{SS + S^2}{(S + p_0)(S + p_1)} \leq 0.$$

The above shows that selecting the set of edges so that the remaining edges are associated with smaller probabilities leads to lower (better) values of the optimization function. □

Having the outgoing edges of i sorted by transition probability, we can compute $ISOL_i(m)$ for all $m = 0 \ldots k$. The dynamic programming equation is:

$$SOL_i(k) = \arg\min_{0 \leq m \leq k} \{ISOL_i(m) + SOL_{i+1}(k - m)\} \tag{3.20}$$

`EdgeDP` essentially computes and keeps in memory $\|V\| \times (k+1)$ values according to Equation(3.20) and is described in Algorithm 1.

ALGORITHM 1: Dynamic programming for the EdgeTransitions variant.

Input: k
Output: SOL: Dynamic programming array
Initialize empty array $SOL_{\|V\| \times (k+1)}$;
for $i = \|V\| \cdots 1$
  for $k' = 0 \cdots k$
    $SOL[i, k'] := \arg\min_{0 \leq k_i \leq k'} \{ISOL_i(k_i) + SOL[i+1, k' - k_i]\}\}$
return SOL;

Theorem 2. *`EdgeDP` is optimal for the* EdgeTransitions *variant of the* Markov Chain Monitoring *problem.*

Proof. The proof follows from Lemma 2 and by construction of the dynamic programming algorithm (Equation (3.20)). □

Running time

`EdgeDP` computes $k \times |V|$ values. For each value to be computed, up to $O(k)$ numerical operations are performed. Therefore, `EdgeDP` runs in $O(k^2|V|)$ operations. Backtracking to retrieve the optimal solution requires at most equal number of steps, so it does not increase the asymptotic running time.

3.4.2 The `EdgeGreedy` Algorithm

`EdgeGreedy` is a natural greedy algorithm that selects k edges in an equal number of steps, in each step selecting one more edge to minimize F_{ET}. `EdgeGreedy` is described in Algorithm 2.

ALGORITHM 2: Greedy algorithm for the EdgeTransitions variant.

Input: k
Output: `ResultEdges`: Set of selected edges

```
1 ResultEdges = {}
2 for i = 1 ··· k
3 |   Select e ∈ E := arg min F_ET(ResultEdges ∪ {e})
4 |   ResultEdges := ResultEdges ∪ {e};
5 |   E := E \ {e};
6 return ResultEdges;
```

In all our experiments the output of `EdgeGreedy` is the same as that of the optimal `EdgeDP` algorithm. However, we do not have a proof that the former is also optimal. We leave this as a problem for future work.

Running time

Following Equation (3.9), to select k edges, `EdgeGreedy` invokes up to $k \times O(|E|)$ evaluations of F_{ET}. As we discussed for `NodeGreedy`, if the evaluation of the objective function is naively implemented with a double summation, the running time of `EdgeGreedy` is $O(k|E||V|^2)$ numerical operations. If the objective function is imple-

mented as a summation over edges, the running time improves to $O(k|E|^2)$. Furthermore, following the observations similar to those we saw for `NodeGreedy`, the running time of `EdgeGreedy` becomes $O(|E| + k|E|) = O(k|E|)$.

We notice that `EdgeDP` has better performance than `EdgeGreedy` for dense graphs ($|E| \approxeq |V|^2$) and small k. Moreover, as with `NodeGreedy`, `EdgeGreedy` is amenable to parallelization - the new value of the objective function can be computed in independently for each candidate edge.

3.5 Data and Experiments

In this section, we describe the results of our experimental evaluation using real and synthetic data. The results demonstrate that our methods perform better than other baseline methods with respect to our objective function. Moreover, using the bike-sharing network of Boston, we provide anecdotal evidence that our methods pick meaningful nodes to monitor.

3.5.1 Experimental Setup

Let us first describe the experimental setup, i.e., the datasets and baseline algorithms used for evaluation.

Graph datasets

We use the following graphs to define Markov chains for our experiments.

- `AS` graphs: The `AS` is a graph that contains information about traffic between *Autonomous Systems* of the Internet. An Autonomous System (AS) is a set of IP addresses, typically characterized by their common prefix and belonging to a single internet provider or large organization. The dataset was retrieved through the Stanford Large Network Dataset Collection (SNAP) [49]. We experimented with three snapshots of the Autonomous Systems communication

graphs between the years 1997–2000. Here we demonstrate results for one of the snapshots (1999-2000), as we did not find a significant difference among them.

The `AS` graph contains one node for each AS. Moreover, for every pair of nodes with internet traffic between them, we place two directed edges between the nodes, one in each direction. To create an instance of the transition matrix, we assign equal probabilities to the outgoing edges of each node.

- `Grid` graphs: The `Grid` graphs are planar, bi-directed grid graphs, where each node has in- and out-degree 4 (with the exception of border nodes).

- `Geo` graphs: The `Geo` graphs are bi-directed geographic graphs. They are generated as follows: each node is placed randomly within a unit square on the two-dimensional Euclidean plane. Subsequently, pairs of nodes are connected with directed edges in both directions if their euclidean distance is below a pre-defined threshold $ds = 0.01$.

- `BA` graphs: The `BA` graphs are generated according to the Barabasi-Albert model. According to the model, nodes are added to the graph incrementally one after the other, each of them with outgoing edges to m existing nodes selected via preferential attachment. Here we show results for $m = 3$, but they are similar for other values m.

Similar to the methodology of Gionis *et al.* [33], the `Grid`, `Geo` and `BA` graphs provide us with different varieties of synthetic graphs to explore the performance of our methods. While the `AS` and `Grid` graphs are included to capture the network and city-wide traffic applications, the `Geo` and `BA` graphs are motivated from applications in understanding the spread of an influence or an outbreak.

Item distributions

While the graph datasets described above provide us with illustrative graph structures underneath the Markov Chain, they are not naturally associated with a wide varity of item distributions relevant to our problem. Hence, for each of the aforementioned graphs, we generate an initial distribution of items $\mathbf{x}$ according to one of the following four schemes.

- `Ego`: Items are assigned in two steps. Firstly, one node is selected uniformly at random among all nodes. Secondly, 70% of items are assigned randomly to the neighbors of the selected node (including the selected node itself). Finally, the remaining items are distributed randomly to the nodes outside the neighborhood of the selected node.
- `Uniform`: Each node is assigned the same number of items.
- `Direct`: The number of items on each node is directly proportional to its out-degree. Note that items are distributed in a deterministic manner.
- `Inverse`: The number of items on each node is assigned deterministically to be inversely proportional to its out-degree.

Now each graph described above is combined with each item-distribution scheme. As a result, we obtain datasets of the form `G-X`, where `G` is any of `AS`, `Grid`, `Geo` and `BA` and `X` is any of the `Ego`, `Uniform`, `Direct` and `Inverse`. For simplicity, we perform experiments over a single fixed instantiation of all the datasets generated above.

The `Hubway` dataset

Hubway (now rebranded as BlueBikes) is a bike-sharing system in the Boston metro area, with a fleet of over 1000 bikes and over 100 stations where users can pick up or

drop off bikes. Whenever a user picks up a bike from a Hubway station, the system records basic information about the trip, such as the pick-up and drop-off station and the corresponding pick-up and drop-off times. Moreover, the data contain the number of available bikes at each Hubway station every minute. The dataset was made publicly available by Hubway for its Data Visualization Challenge[1].

Using the dataset, we create instances of the problems we consider as follows. Firstly, we create a complete graph by representing each station with one node in the graph and considering all possible edges between them. Next, we consider a time interval (t_s, t_e) and the bikes that are located at each station (node). Representing bikes as items in our setting, we assign a transition probability $p(u, v)$ between nodes u and v by considering the total number n_u of bikes at station u at start time t_s and, among these bikes, the number n_{uv} of them that were located at station v at end time t_e. We then set $p(u, v) = n_{uv}/n_u$ and ignore edges with zero transition probability.

We experimented with a large number of such instances for different intervals (t_s, t_e), with a moderate length of 2 hours, to capture real-life transitions from one node to another. For the experiments presented in the chapter, we use a fixed instance for the interval between 10 am and 12 pm on April 1st, 2012. In this interval, we consider 61 stations with at least one trip starting or ending at each. We refer to the dataset so constructed as the `Hubway` dataset.

Baseline algorithms

To the best of our knowledge, we are the first to tackle the prolem of Markov Chain Monitoring. However, in order to assess the performance of our proposed algorithms for the Node-Monitoring and Edge-Monitoring problem variants, we compare it to that of well-known baseline algorithms targeting similar centrality-based objectives. Below, we describe the respective baselines for the two variants of

[1] http://hubwaydatachallenge.org/

the problem.

Baselines for Node-Monitoring

For a budget k, the following baselines return a set of k nodes with highest value for the respective measure:

- `In-Degree`: number of incoming edges;
- `In-Probability`: total probability of incoming edges;
- `Node-Betweenness`: as defined in [7, 22, 73];
- `Closeness`: as defined in [75];
- `Node-NumItems`: number of items before transition;

Baselines for Edge-Monitoring

For a budget k, the following baselines return a set of k edges with highest value for the respective measure:

- `Edge-Betweenness`: as defined in [7, 22, 73];
- `Edge-NumItems`: expected number of items to transition over the edge;
- `Probability`: transition probability of the edge.

Henceforth, the baseline and the variants are deterined by context.

Graph	Item Distribution	$r(\texttt{NodeGreedy})$	$r(\texttt{Node} - \texttt{Baseline}^*)$	$r(\texttt{EdgeGreedy})$	$r(\texttt{Edge} - \texttt{Baseline}^*)$
AS	Ego	0.06	0.24	0.14	0.15
	Direct	0.66	0.67	0.99	0.99
	Uniform	0.38	0.40	0.97	0.99
	Inverse	0.38	0.40	0.97	0.99
Geo	Ego	0.00	0.06	0.01	0.02
	Direct	0.00	0.06	0.20	0.65
	Uniform	0.00	0.06	0.15	0.65
	Inverse	0.00	0.07	0.15	0.65
Grid	Ego	0.27	0.27	0.29	0.29
	Direct	0.92	0.92	0.98	0.98
	Uniform	0.92	0.92	0.98	0.98
	Inverse	0.92	0.92	0.98	0.98
BA	Ego	0.18	0.56	0.26	0.26
	Direct	0.71	0.71	0.99	0.99
	Uniform	0.63	0.63	0.98	0.98
	Inverse	0.63	0.63	0.98	0.98

Table 3.1: Comparison of greedy algorithms with the best-performing baseline ($\texttt{Node} - \texttt{Baseline}^*$ and $\texttt{Edge} - \texttt{Baseline}^*$) for $k = 50$. For a given pair of graph and item-distribution scheme, $r(A)$ expresses the ratio of the expected uncertainty that algorithm A achieves with $k = 50$ monitoring operations over the initial uncertainty F_0 (for $k = 0$). Note that the best-performing baseline is different for different rows of the table.

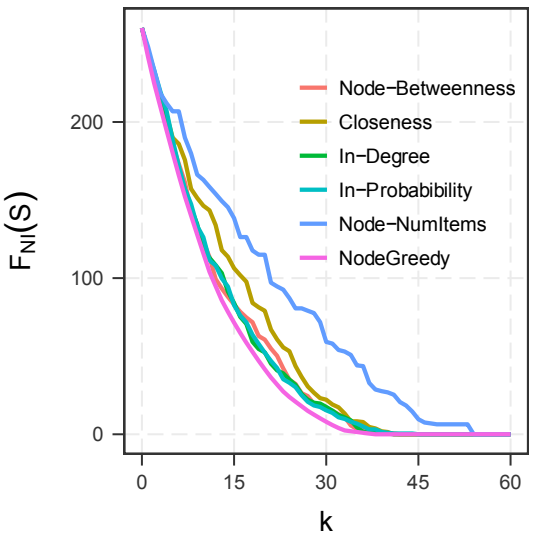

Figure 3·1: NODE-MONITORING on `Hubway` data; y-axis: expected uncertainty, x-axis: number of monitored nodes or edges.

3.5.2 Experimental Results

In this section, we report the performance of algorithms for the MARKOV CHAIN MONITORING problem - first on the graph datasets, combined with item distribution schemes; then on the `Hubway` dataset. As objective we always use the expected uncertainty achieved for a given budget k of nodes or edges – the smaller its value, the better the performance of the algorithm. While `EdgeDP` always provides same solution as that of `EdgeGreedy`, it compares unfavorably in terms of runtime. Hence, we do not report its statistics separately.

We provide the results for the graph datasets in Table 3.1. In all these experiments we use $k = 50$. Moreover, $r(A)$ is the ratio of the achieved objective value (for $k = 50$) over the initial value F_0 of the measure (for no monitoring operations, i.e., $k = 0$). The table shows four quantities for every graph-item distribution pair: $r(A)$ for $A = \{$`NodeGreedy`, `Node-Baseline`*, `EdgeGreedy`, `Edge-Baseline`* $\}$. Note that `Node-Baseline`* (resp. `Edge-Baseline`*) refers to the baseline algorithm with the

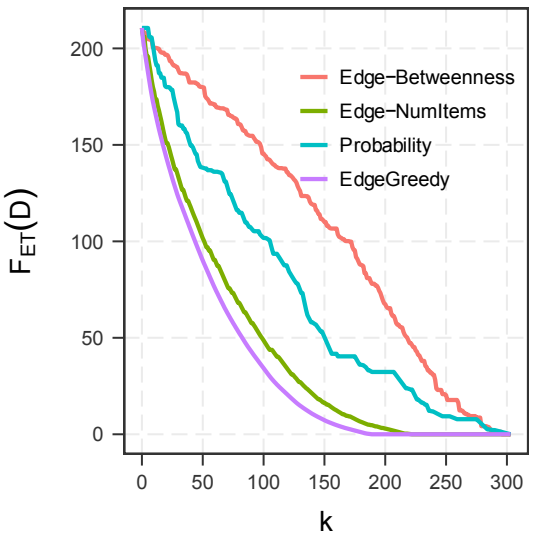

Figure 3·2: EDGE-MONITORING on `Hubway` data; y-axis: expected uncertainty, x-axis: number of monitored nodes or edges.

best performance. For every algorithm A, $r(A) \in [0, 1]$ and the smaller the value of $r(A)$ the better the performance of the algorithm.

From the table, we observe that for the `AS` dataset, `NodeGreedy` significantly outperforms the best baseline for the `Ego` item distribution, while performing marginally better for other item distributions. The value of r(`EdgeGreedy`) is only slightly less than the best baselines across all the configurations. However, we observe that there is no baseline which performs uniformly the best across different item distributions. For example, `Edge-Betweenness` is the best baseline for `Direct` item distribution, the `Edge-NumItems` for `Ego`, while they both perform worse than even randomly chosen edges for `Uniform` and `Inverse` item distributions. Notably, for the `Geo` graphs, the greedy algorithms significantly outperform the baselines

For the `Grid` graphs, the baselines perform exactly the same as our algorithms. This can be explained by the nature of the `Grid` graph, where all the nodes except the ones on the boundary are similar to each other, thereby rendering the `Direct`, `Uniform` and `Inverse` item distributions very similar to each other. For the `Ego` dis-

tribution, the greedy algorithms perform marginally better than the baselines. Again, there is no baseline which performs uniformly the best. Similar is our explanation for the results on `BA` graphs as in these graphs most of the nodes have almost the same (small) degrees too.

Figure 3·3 shows the performance of the `NodeGreedy` algorithm for the the `Geo` graphs, with each plot corresponding to a different item distribution schemes. Observe that `NodeGreedy` significantly outperforms all other baselines, which capture different semantics of centrality. In particular, we observe that `NodeGreedy` achieves zero or near-zero expected uncertainty with a small fraction of selected nodes compared to baselines. Among the baselines, `Closeness` performs second-best in many cases, while `In-Degree` performs as well as `Closeness` for small k.

Similarly, Figure 3·4 shows the performance of the different algorithms for the EDGE-MONITORING and the `Geo` graphs, for all possible item-distribution schemes. As before, we observe that `EdgeGreedy` outperforms the baselines in all cases. We notice also that the pattern of performance differs somewhat for the case of `Ego` item distribution. With the exception of one baseline (`Probability`), all algorithms achieve steep decline in expected uncertainty for small value of k - `EdgeGreedy` performs best, but baselines are competitive. However, for larger k, the performance of baselines does not keep up with that of `EdgeGreedy`. We believe that this is can be explained as follows: the first edges selected by baselines are either central in terms of graph structure – and therefore near the part of the graph with high concentration of items (`Edge-Betweenness`) – or directly in the area of the graph with many items (`Edge-NumItems`). In terms of reducing expected uncertainty, this is beneficial at first. However, these baselines as they do not optimize our objective are not able to continue reducing the expected uncertainty with their subsequent selections.

Figure 3·5 and Figure 3·6 show the performance of the greedy algorithms on

the NODE-MONITORING and the EDGE-MONITORING problems respectively. We observe that both the `NodeGreedy` and the `EdgeGreedy` algorithms are consistently the best when compared to the baselines. However, $k = 50$ represents about 1% of the total edges in the graph, hence their monitoring does not decrease the uncertainty significantly. While experiments with larger values of k are prohibitive due to time complexity of the `EdgeGreedy` algorithm, we postulate that the greedy algorithm will still continue outperforming the baselines.

Figures 3·7 and 3·8 provide a similar comparison for the different configurations of the `BA` graph. The greedy algorithms provide marginal benefits or perform on par with competitive baselines. On the `BA` graphs, for `Direct`, `Uniform` and `Inverse` item distributions, some baselines perform exactly the same as the greedy algorithms for relatively small number of monitoring operations i.e., $k = 50$. Lastly, we observe similar trends in case of the `Grid` graphs as evident in Figures 3·9 and 3·10. It should be noted that there is no baseline method that provides a consistently competitive performance with the greedy algorithms across all different configurations described above.

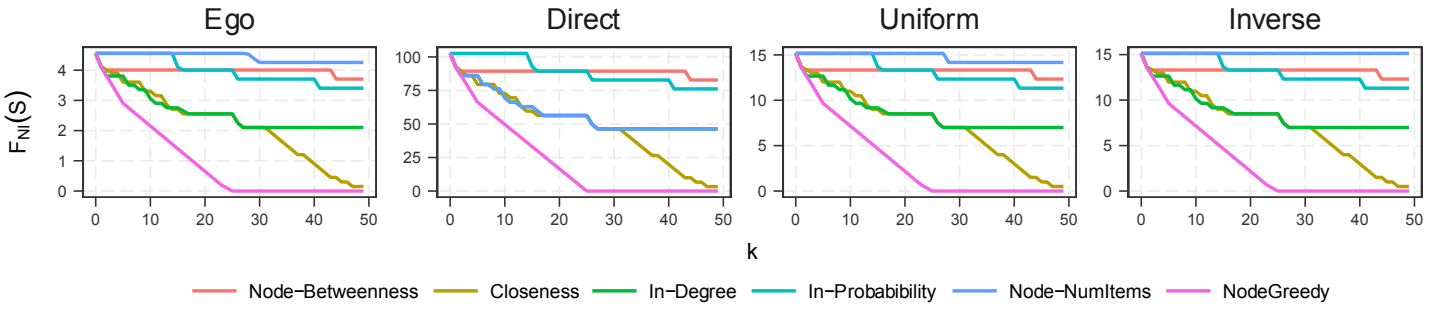

Figure 3·3: NODE-MONITORING `Geo` dataset; y-axis expected uncertainty, x-axis: number of monitored nodes (k).

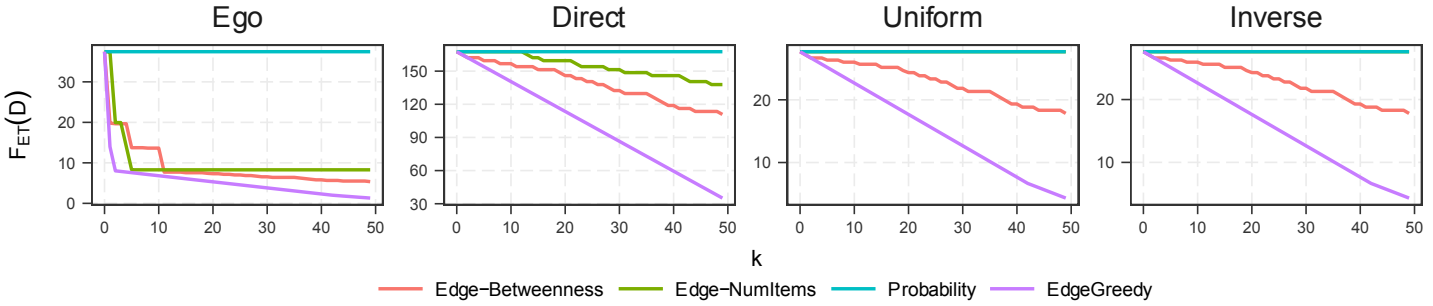

Figure 3·4: EDGE-MONITORING `Geo` dataset; y-axis expected uncertainty, x-axis: number of monitored edges (k).

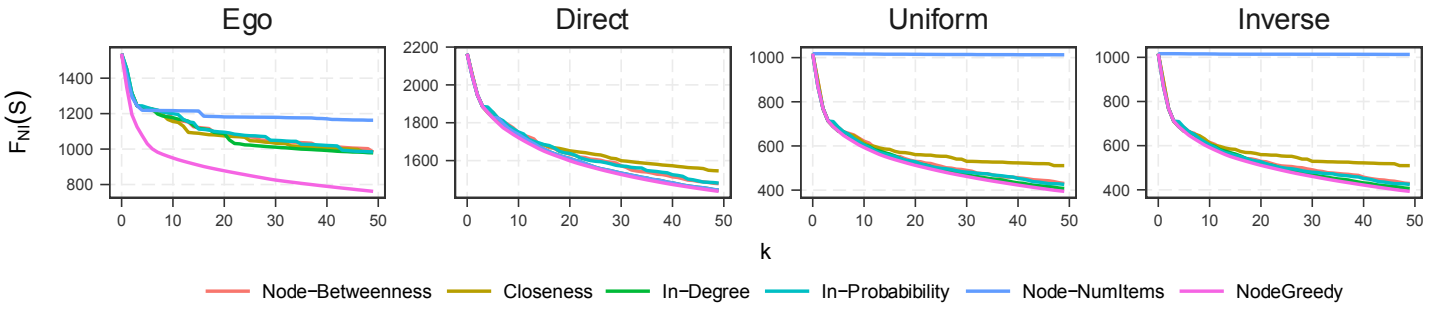

Figure 3·5: NODE-MONITORING AS dataset; y-axis expected uncertainty, x-axis: number of monitored nodes (k).

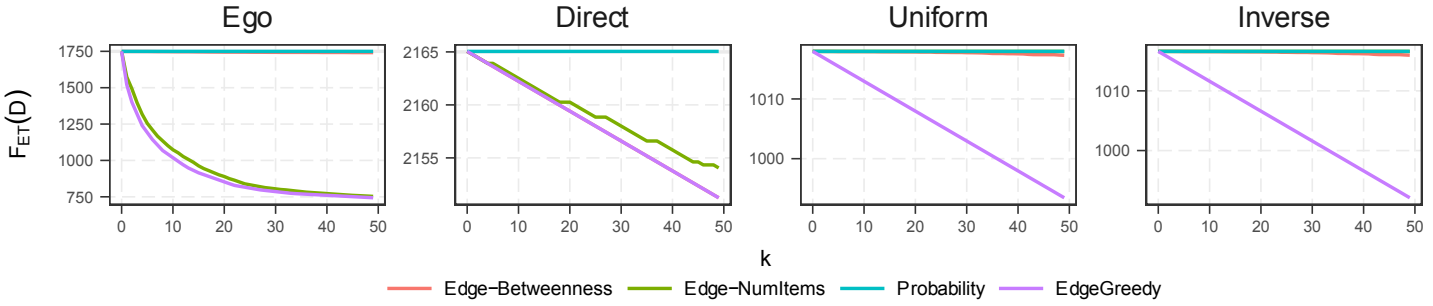

Figure 3·6: EDGE-MONITORING AS dataset; y-axis expected uncertainty, x-axis: number of monitored edges (k).

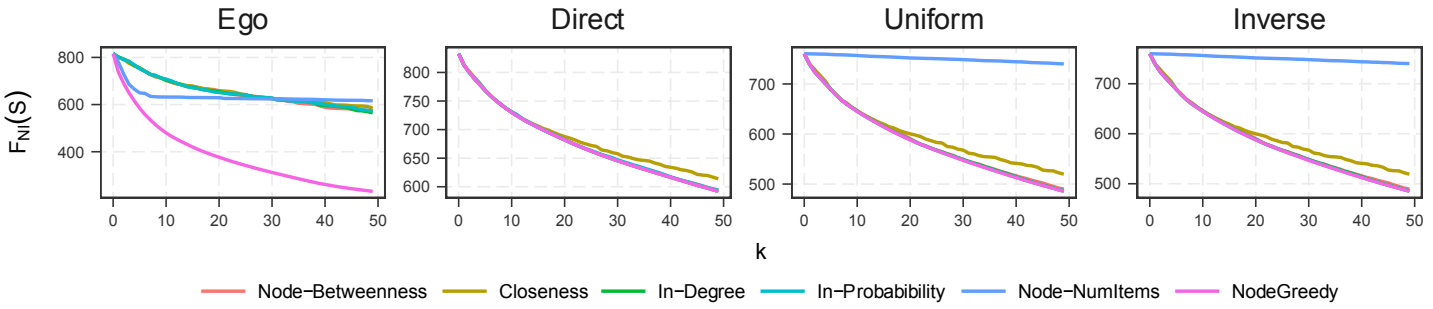

Figure 3·7: NODE-MONITORING BA dataset; y-axis expected uncertainty, x-axis: number of monitored nodes (k).

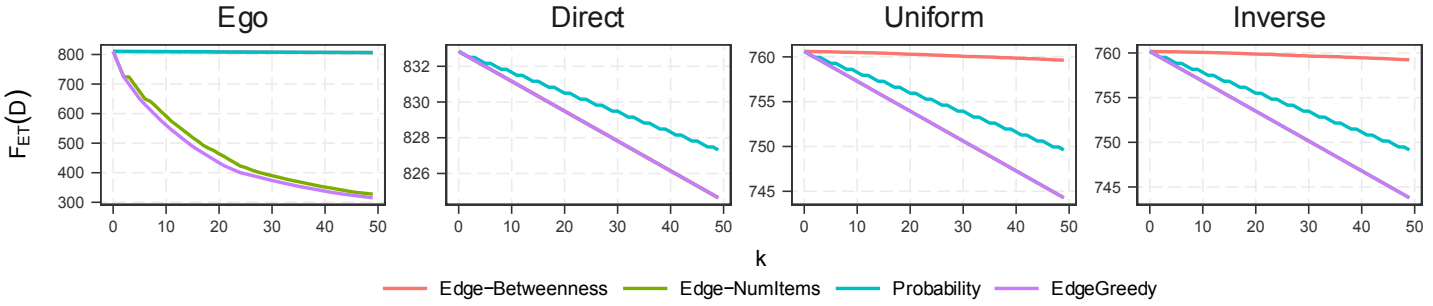

Figure 3·8: EDGE-MONITORING BA dataset; y-axis expected uncertainty, x-axis: number of monitored edges (k).

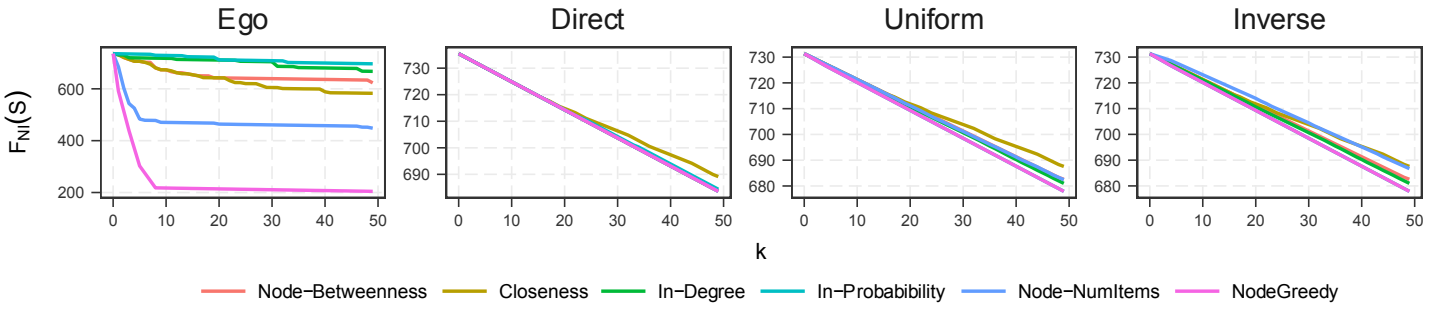

Figure 3·9: NODE-MONITORING `Grid` dataset; y-axis expected uncertainty, x-axis: number of monitored nodes (k).

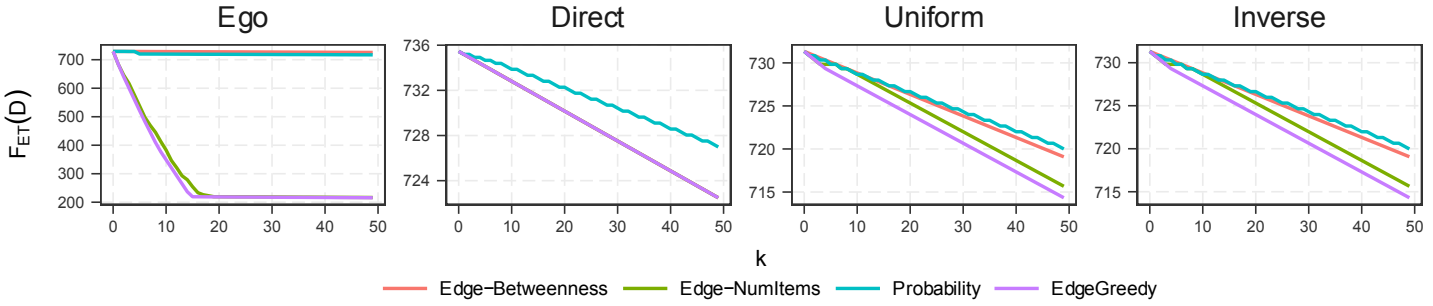

Figure 3·10: EDGE-MONITORING `Grid` dataset; y-axis expected uncertainty, x-axis: number of monitored edges (k).

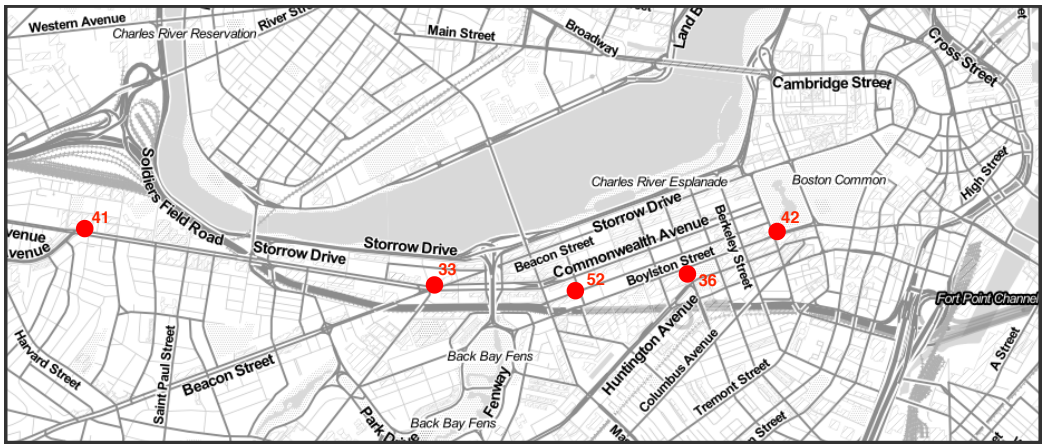

Figure 3·11: `Hubway` dataset. IDs of stations picked as a solution to NODE-MONITORING for $k = 5$; 33: Kenmore Sq., 36: Copley Sq./Boston Public Library, 41: Packard's Corner, 42: Boston Public Garden, 52: Newbury Street.

Experiments with `Hubway` data

In our last experiment, we explore the performance of our algorithms on the `Hubway` dataset. From Figure 3·1 and Figure 3·2 we observe that the `NodeGreedy` and `EdgeGreedy` algorithms are consistently the best at reducing expected uncertainty, although the baselines are competitive on the relatively smaller graph. In Figure 3·11, we plot the Hubway stations across Boston chosen by the `NodeGreedy` algorithm with $k = 5$. The nodes chosen by the algorithm are supported by the anecdotal evidence of being exactly some of the of the most popular landmarks around the city. While Boston Public Garden, Boston Public Library, and Newbury Street are important landmark locations near downtown Boston, Kenmore Square and Packard's Corner are busy intersections near Boston University. In Figure 3·12, we plot the pairs of Hubway stations across Boston chosen by the `EdgeGreedy` algorithm for $k = 10$. Anecdotally, it is should be noted that chosen source–destination pairs connect stations that are not otherwise well connected by subway. From a managerial perspective, tracking the number of trips starting or ending at these Hubway stations can help

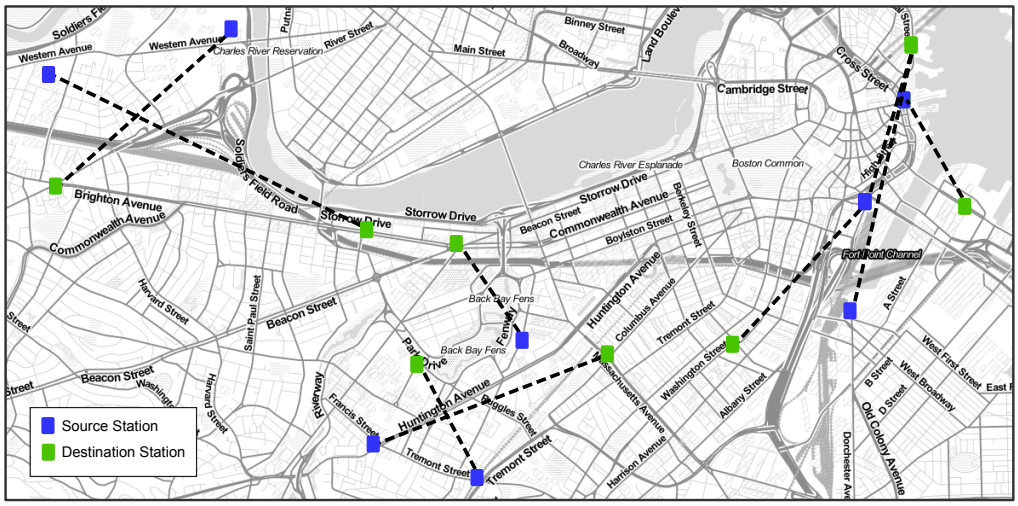

Figure 3·12: `Hubway` dataset. Pairs of Boston stations picked as solution to EDGE-MONITORING for $k = 10$.

the operators better reduce the expected uncertainty around the expected number of bikes available at its different stations and anticipate future bike "re-balancing" [2] operations.

Running times

For all our experiments we use a single process implementation of our algorithms on a 24-core 2.9GHz Intel Xeon E5 processor with 512GB memory. For the largest graph in our experiments, the parallelized version of the `NodeGreedy` takes about $5 - 10$ seconds per selected node, while the parallelized version of `EdgeGreedy` takes about 1 minute per selected edge.

Discussion

Our experiments show that `NodeGreedy` and `EdgeGreedy` consistently perform better than or on par with other popular baseline methods. Also, for graphs with rela-

[2] https://www.citylab.com/transportation/2014/08/balancing-bike-share-stations-has-become-a-serious-scientific-endeavor/379188/

tively large number of nodes, the solutions to the NODE-MONITORING problem are more effective at reducing the expected uncertainty than the solutions to the EDGE-MONITORING problem for the same number of node (resp. edge) monitors. This is especially important considering our analysis from Section 3.3 and 3.4 which show that the `NodeGreedy` algorithm has a better time complexity compared to the `EdgeGreedy` for dense graphs.

Chapter Summary

In this chapter, we introduced the problem of MARKOV CHAIN MONITORING, a problem of both theoretical and practical interest in the operations of an MoD system. Given an initial distribution of fleet vehicles over the graph representation of a city in the form of a Markov chain, we aim to perform a limited number of monitoring operations in order to minimize the uncertainty around the fleet distribution in future steps. We studied variants of this problem and proposed efficient algorithms to solve them. We rigorously evaluated these algorithms on various real and synthetic datasets. Finally, we showcased its use on a dataset from the bike-sharing MoD system, Hubway, from the city of Boston.

Chapter 4

Driver Revenue Maximization

The proliferation of on-demand ride-hailing platforms like Lyft and Uber has begun to fundamentally change the nature of urban transit. In the last two years alone, the number of daily trips using ride-hailing platforms like Uber and Lyft in New York City has grown five-fold, to about 350,000 trips per day. Today, over 65,000 drivers drive on the streets of New York City as Uber or Lyft drivers. The explosive growth of these ride-hailing platforms has motivated a wide array of questions for academic research at the intersection of computer science and economics, ranging from the design of effective pricing mechanisms, to equilibrium analysis, to the design of reputation management systems for drivers, to algorithms for matching drivers with customers, as we discuss in our related work section.

While these studies consider the study of ride-hailing platforms holistically, little work has been done on optimizing strategies for individual drivers. Nevertheless, the challenge of how to maximize one's individual earnings as a driver for a ride-hailing platform like Uber or Lyft is a pressing question that millions of micro-entrepreneurs across the world now face. Anecdotally, many drivers spend a great deal of time strategizing about where and when to drive. However, drivers today are self-taught, using heuristics of their own devising or learning from one another, and employ relatively simple analytics dashboards such as SherpaShare. Indeed, rumors suggest that some drivers even collude in attempts to induce spikes in surge prices that they can then exploit. But in terms of concrete guidance, to date, there are only articles in

the popular press and on blogs that offer (often contradictory) advice to ride-hailing drivers on how to maximize their earnings [36, 38, 83].

In this chapter, we formalize the problem of devising a driver strategy to maximize expected earnings and describe a series of dynamic programming algorithms to solve this problem under different sets of modeled actions available to the drivers. Our strategies take as input a detailed model of city-level data that constitutes a fine-grained weekly projection of forecasted demand for rides, comprising predicted spatiotemporal distributions of source-destination pairs, driver payments, transit times, and surge multipliers. The optimization framework we propose not only produces contingency plans in the form of highly optimized driving schedules and real-time in-course corrections to drivers, but also enables us to rigorously reason about and analyze the sensitivity of our output results to perturbations in the input data. Thus, we can justify the proposed strategies even under an uncertainty level in the collected data and the data model itself.

We then exemplify our results with a large-scale simulation of driving for Uber in New York City. For this simulation, we assemble a new dataset that uses both the publicly available New York City taxi rides dataset [1] as well as calls to the Uber API. From the former, we obtain information about over 200,000 taxi rides that occurred between different New York City zones. From the latter, we obtain representative pricing and traffic-time information for those trips, were they to reoccur on Uber. From this dataset, we construct a mathematical model to produce input to our algorithms. However, we view the dataset to be of independent interest that could subsequently be used for a multitude of other studies.

Our experiments with our methods on this dataset demonstrate the following findings. Being strategic about the areas they focus on picking up riders and the times they work, drivers can significantly increase their income, sometimes by as much as

[1] `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`

1.5x, when compared to a naive optimization strategy. Moreover, we show that a pronounced difference between earnings holds even when there is large uncertainty in the input data. We argue that our results are therefore not purely an artifact of the New York City dataset we employ, but also have high potential to generalize. Finally, our experiments show that naively chasing surging prices does not typically lead to significant earnings gains, but it can actually introduce large opportunity costs, as drivers waste time driving to subsiding surges.

Chapter Organization

The rest of the chapter is organized as follows. We begin by discussing the problem setup in Section 4.1. Based on the assumptions discussed in Section 4.1, we formulate various strategic driver strategies in Section 4.2. Sections 4.3-4.4 are devoted to developing robust optimization techniques to evaluate the impact of perturbations in input data. Finally, Section 4.5 starts by discussing the data collection and pre-processing strategies and follows it up with a rigorous evaluation of the proposed driver revenue maximization strategies.

4.1 Problem Setup

In this section, we describe the basics of our problem setup and provide the necessary notation.

4.1.1 Modeling the City

Throughout the chapter, we will assume that a city is divided into non-overlapping set of zones denoted by $\mathcal{X}$, and time t runs in discrete time steps. We represent a city in the form of a complete weighted directed graph $G = (\mathcal{X}, E)$ with $|\mathcal{X}| = n$ and $|E| = \binom{n}{2}$ edges, where the edge weight on edge $e(i \to j)$ corresponds to the likelihood of a driver currently at location i receiving a ride request to location j. Additionally,

each edge is associated with a travel time $\tau(i, j)$, a travel cost, and a reward $r(i, j)$. In the general formulation of our problem, all of these edge attributes are time-varying, e.g., the rewards would vary with t as $r^t(i, j)$, but to avoid excess notation, we drop those superscripts in our following discussion of models and algorithms, and reintroduce them only in our experiments in Section 4.5. These attributes of a city, which we use as an input to our solver, are specified as follows:

Empirical transition matrix (F)

Every edge $e(i \to j) \in E$ is associated with a transition probability $f(i, j) \in [0, 1]$ such that $\sum_{j \in \mathcal{X}} f(i, j) = 1$, $\forall i \in \mathcal{X}$.

Since the entries of $\mathbf{F}$ correspond to probabilities, the weights give rise to a Markov chain with a transition matrix $\mathbf{F}$ – where each entry $f(i, j)$ denotes the probability of a passenger in zone i traveling to zone j. As we disallow trips within the same zone in our model (an assumption which could be relaxed), we let $f(i, i)$ denote the probability of a driver not finding a passenger in zone i at a given time step.

Travel time matrix (T)

Every edge $e(i \to j) \in E$ is also associated with $\tau(i, j) > 0$, the travel time of a ride from zone i to zone j. These weights give us a travel time matrix $\mathbf{T}$ with entries $\tau(i, j)$.

Rewards matrix (R)

Every edge $e(i \to j) \in E$ is also associated with a real valued reward $r(i, j)$ denoting the net reward for a driver delivering a passenger from zone i to zone j. The net rewards include the driver's share of earnings from a passenger minus the sundry costs like gas, vehicle depreciation, etc. Since these earnings and costs vary with mileage and transit time, each entry in the rewards matrix $\mathbf{R}$ is of the form $r(i, j) =$

earnings(i, j) - cost(i, j).

Again, in general, all of the input matrices: $\mathbf{F}$, $\mathbf{T}$ and $\mathbf{R}$, are time-dependent, i.e., their entries could change throughout the day

4.1.2 Modeling the Driver

Our model assumes that each driver comes with a maximum work budget of B time units, during which the driver can pick up passengers. Depending on the specific setting, the driver can work B time units consecutively or split them over a finite horizon of N time units, where $N \geq B$. As an example, a driver seeking to optimize an 8 hour work day over a 24 hour day at a ten-minute decision granularity (at most six decisions per hour), will have $B = 48$ and $N = 144$.

Home zone (i_0)

Each driver has a unique home zone denoted by $i_0 \in \mathcal{X}$. We always assume that each driver starts from their home zone and returns to it at the end of each of their shifts.

Driver actions ($\mathcal{A}$)

In a driver strategy, whenever faced with a choice regarding their next decision, a driver has $n + 2$ possible actions to choose from:

- *Get Passenger* (a_0): Wait for a passenger in the current zone.
- *Go Home* (a_1): Log out of the on-demand ride service, relocate to the home zone (if needed) and stop working. This action does not consume the driver's budget.
- *Relocate* ($a_2(j)$): Relocate to city zone j. This action consumes the driver's budget.

Driver policy (π)

A driver policy is a sequence of time and location-dependent actions taken by a driver at different steps of the strategy. As the total number of actions taken by a driver while exhausting the budget B depends on the actual actions, the length of a driver policy π varies.

Each time and location dependent action in π, denoted by $\mathbf{a}$, can be expressed in form of a 3-tuple – $(\hat{i}, \hat{t}, \hat{a})$ where $\hat{a} \in \mathcal{A}$ refers to actual action, $\hat{i} \in \mathcal{X}$ is the zone at which action was taken and $\hat{t} \leq N$ is the time at which the action was taken. Finally, we use Π to denote the set of all possible policies.

4.1.3 Computing Driver Earnings

In this section, we describe the computation of the expected earnings of a driver who at a specific time t is in zone i and takes action a. We denote this by $E(i, t, a)$ and depending on the action a it is computed as follows.

- For action a_0 (*Get Passenger*), taken inside zone i at time t, the action earnings function is calculated as an expectation over possible rides,

$$E(i, t, a_0) = \mathbf{F}_i \cdot \mathbf{R}_i \tag{4.1}$$

  where $\mathbf{F}_i$ and $\mathbf{R}_i$ denote the i-th rows of $\mathbf{F}$ and $\mathbf{R}$ respectively.

- For action a_1 (*Go Home*), taken inside zone i at time t, the action earnings function is simply

$$E(i, t, a_1) = -cost(i, i_0) \tag{4.2}$$

  where we incur a negative reward due to the absence of a paying customer.

- Action $a_2(j)$ (*Relocate*), taken inside zone i at time t, takes the driver to zone

$j \neq i$. Therefore, the action earnings function is

$$E(i, t, a_2(j)) = -cost(i, j) \tag{4.3}$$

where the driver again incurs a negative reward due to the absence of a paying customer.

4.1.4 Problem Statement

Given input specification matrices $\mathbf{F}$, $\mathbf{T}$ and $\mathbf{R}$, as well as the driver's budget B, the *total expected earnings* of the driver with policy π is:

$$\mathcal{E}(\pi, \mathbf{F}, \mathbf{T}, \mathbf{R}, B) = \sum_{(\hat{i},\hat{t},\hat{a})\in\pi} E(\hat{i}, \hat{t}, \hat{a}), \tag{4.4}$$

where $E(\hat{i}, \hat{t}, \hat{a})$ is computed using the Equations (4.1), (4.2) and (4.3).

As we seek to maximize the *total expected earnings* of the driver, we aim to solve the following optimization problem.

Problem 4 (MaxEarnings)**.** *Given sets of time-evolving* $\mathbf{F}$*,* $\mathbf{T}$ *and* $\mathbf{R}$*, as well as the driver's budget* B*, find a* π^* *such that:*

$$\pi^* = \arg\max_{\pi\in\Pi} \mathcal{E}(\pi, \mathbf{F}, \mathbf{T}, \mathbf{R}, B).$$

4.2 Driver Strategies

We now describe the different driver strategies, which are defined based on the set of actions $\mathcal{A}$ at the driver's disposal. We also show how to optimally solve the MaxEarnings problem in polynomial time for different sets $\mathcal{A}$.

For the rest of the section, we will denote by $\Phi(i, b, t)$ the *total expected future earnings* of a driver who is in zone i at time t with budget b time units remaining. Hence, the *total expected earnings* of a driver can be expressed as $\Phi(i_0, B, N)$.

If a driver at zone i at time t with b budget units remaining either takes a passenger ride to zone j or relocates to zone j, that trip ends at time $t' = t + \tau^t(i,j)$ with remaining budget $b' = b - \tau^t(i,j)$. The total expected future earnings at that point for the driver is: $\Phi(j, b', t')$. Let $\mathbf{v}(i,b,t)$ denotes the vector of such cumulative earnings across different zones j induced when a driver takes an a_0 action i.e., $\mathbf{v}(i,b,t) = \left[\Phi(j,b',t')\right]_{j\in\mathcal{X}}$.

We now define the driver strategies as well as the solutions to the instances of the MaxEarnings problem they induce.

The *flexible-relocation* strategy

This is the most general strategy where a driver has complete freedom for choices regarding work schedule as well relocation to different zones. Specifically, a driver has a budget constraint of B time units to be consumed over a finite horizon N time units. An idle driver in zone i following this strategy has following set of available choices,

$$\mathcal{A} = \{a_0, a_1\} \cup \{a_2(j) | \forall j \in \mathcal{X}, j \neq i\} \tag{4.5}$$

Note that we restrict the *Relocate* actions to ones which do not result in $t \geq N$ or $b < 0$.

A driver following the *flexible-relocation* strategy chooses the action that maximizes *total expected earnings*. For this strategy, the solution to the MaxEarnings problem can be found by the following dynamic programming (DP) recurrence:

$$\Phi(i,b,t) = \max_{a\in\mathcal{A}} \begin{cases} \mathbf{F}_i(\mathbf{R}_i + \mathbf{v}(i,b,t)), & \text{if } a = a_0 \\ -cost(i,i_0) + \Phi(i_0,b,t'), & \text{if } a = a_1 \\ \max_j\{-cost(i,j) + \Phi(j,b',t')\}, & \text{if } a = a_2(j) \end{cases} \tag{4.6}$$

Each of the $O(nNB)$ entries in the output of this dynamic program involves consid-

eration of at most $O(n)$ actions. Hence, the solution to the MaxEarnings problem can be found in $O(n^2NB)$ time.

Other strategies

In addition to the general *flexible-relocation* strategy, we also consider the following three special cases to model other plausible strategies of ride-hailing drivers: the *naive*, the *relocation* and the *flexible* strategies.

In the *naive* strategy, a driver performs a random walk over the city on weekdays from 9AM - 5PM, with locations dictated exclusively by the passengers picked up. At the end of every passenger ride, the driver waits in the current zone for next passenger pickup. Hence, the only allowable action is *Get Passenger*.

In the *relocation* strategy, an idle driver in zone i has two choices: *Get Passenger* and *Relocate*. Hence, the set of allowable actions for a driver contains n different actions, one of which is *Get Passenger* and $(n-1)$ *Relocate* actions, one for each different city zone. Thus: $\mathcal{A} = \{a_0\} \cup \{a_2(j) | \forall j \in \mathcal{X}, j \neq i\}$. We remove from consideration the zones where relocating exhausts the budget or where $t \geq N$.

In the *flexible* strategy, a driver has the flexibility to decide working times, modeling a driver who uses heuristics to decide the most profitable times to work. As a result, we impose an additional constraint of a working time budget B that a driver can split over a finite horizon of N time units. Thus, this strategy aims to figure out an optimal in-expectation work schedule for the driver. At any stage, a driver can log out of the on-demand ride service and return to home zone. Hence, the set of allowable actions for a driver contains 2 different actions, *Get Passenger* and *Go Home*. Thus: $\mathcal{A} = \{a_0, a_1\}$. It is common for drivers to structure their day around a desired target earning, rather than a time budget. The *flexible* strategy also naturally computes a schedule that minimizes working time required for achieving the desired target earning.

Solving MaxEarnings for the *naive*, the *relocation* and the *flexible* strategies can be done by streamlined versions of the DP presented in Equation (4.6).

4.3 Maximizing Earnings under Uncertainty

The primary source of variability in the input of the MaxEarnings problem is the set of empirical transition matrices $\mathbf{F}$. In a typical application, we expect that predictive models would be employed to generate estimates of these matrices based upon observations from historical data (as we do in our own experiments). Empirically observed transition matrices may suffer from estimation errors due to the presence of external confounding factors (e.g., weather, special events inside the city) while gathering the data. As a result, the dynamic programming solution to MaxEarnings may also be sensitive to the transition probabilities. In this section, we address the question of how the results of the solutions we described in the previous section change under the assumption that there is some uncertainty (and thus noise) in the underlying empirical transition matrices we use as part of our input.

Concretely, we now assume that the empirical transition matrix ($\mathbf{F}$) is generated from an underlying traffic matrix, or count matrix, recording trips between locations i and j.

Count matrix (C)

Every edge $e(i \rightarrow j) \in E$ is associated with an integer-valued weight $c(i, j)$ that denotes the number of requests at zone i that had node j as their destination. Then, we compute frequencies $f(i, j) = \frac{c(i,j)}{\sum_k c(i,k)}$, for all outbound trips from i.

With this, we now describe how to quantify uncertainty in the rows of $\mathbf{F}$ (and the underlying $\mathbf{C}$, by construction). This will enable us to modify the MaxEarnings into the RobustEarnings problem following ideas developed by [65].

4.3.1 Modeling Uncertainty

We now assume that there is an underlying *true* transition matrix $\mathbf{P}$, and the question we explore is our confidence that the $\mathbf{C}$ we observe is actually generated by the true transition matrix $\mathbf{P}$. As before, both $\mathbf{P}$ and $\mathbf{C}$ are clearly time-dependent in practice, but for ease of exposition, we ignore the time-dependency aspect of the problem here.

We consider each row of the true transition matrix and the count matrix separately; let $\mathbf{p}$ and $\mathbf{c}$ denote any particular row of $\mathbf{P}$ and $\mathbf{C}$ respectively.

Following the ideas of Kullback *et al.* [45], we have a discriminatory random variable $2\hat{I}$, which follows a χ^2 distribution with $(n-1)$ degrees of freedom. Heuristically, $2\hat{I}$ can be considered as a measure of the "divergence" of $\mathbf{c}$ from $\mathbf{p}$. Thus, for $\mathbf{c}$ to be in the $(1-\alpha)$ *(or* $100(1-\alpha)\%$*) confidence interval of* $\mathbf{p}$, we need:

$$\begin{aligned} \mathrm{F}_{\chi^2_{n-1}}\left[2\hat{I}\right] &= \mathrm{F}_{\chi^2_{n-1}}\left[2\sum_{i=1}^{n}\mathbf{c}(i)\log\mathbf{c}(i) - 2n\log n - 2\sum_{i=1}^{n}\mathbf{c}(i)\log\mathbf{p}(i)\right] \\ &= 1-\alpha, \end{aligned}$$

where $\mathbf{p}(i)$ (resp. $\mathbf{c}(i)$) is the i-th element of vector $\mathbf{p}$ (resp. $\mathbf{c}$). In the above equation, α quantifies the uncertainty that one can tolerate and is an upper bound on what one believes actually exists in the set of observations $\mathbf{p}$. Thus, we call α the *input uncertainty level*.

By setting $\beta_{\max} = \sum_{i=1}^{n}\mathbf{c}(i)\log\mathbf{c}(i)$, we get

$$\sum_{i=1}^{n}\mathbf{c}(i)\log\mathbf{p}(i) \geq \frac{2(\beta_{\max} - n\log n) - \mathrm{F}^{-1}_{\chi^2_{n-1}}(1-\alpha)}{2}, \tag{4.7}$$

where $\mathrm{F}^{-1}_{\chi^2_{n-1}}$ is the inverse of the χ^2 cdf. In other words, for all vectors $\mathbf{p}$ for which Equation (4.7) is satisfied, $\mathbf{c}$ is within the $(1-\alpha)$-confidence interval of $\mathbf{p}$.

Thus given $\mathbf{C}$ and α, we define the *α-feasible matrices* $\mathcal{P}_\alpha$ to be the set of true transition matrices such that for every matrix $\mathbf{P}$ in $\mathcal{P}_\alpha$ and every row $\mathbf{p}$ of $\mathbf{P}$, Equa-

tion (4.7) is satisfied.

4.3.2 The RobustEarnings Problem

Our approach is to compute the *worst-case total expected earnings* for a driver, by finding the $\mathbf{P}$ among all matrices in $\mathcal{P}_\alpha$ such that the *total expected earnings* of the driver are minimized. This quantifies the worst-case difference between the earnings computed as a solution to the MaxEarnings and the worst-case earnings of the driver, given bounded uncertainty α. We formalized this as the following problem definition:

Problem 5 (RobustEarnings)**.** *Given sets of time evolving* $\mathbf{C}$*,* $\mathbf{T}$ *and* $\mathbf{R}$*, the driver's budget* B *and input uncertainty level* α*, find* $\widehat{\pi}$ *such that:*

$$\widehat{\pi}_* = \arg\max_{\pi \in \Pi} \min_{\mathbf{P} \in \mathcal{P}_\alpha} \mathcal{E}(\pi, \mathbf{P}, \mathbf{T}, \mathbf{R}, B).$$

Note that the above problem requires searching among all possible true transition matrices in $\mathcal{P}_\alpha$, which is a non-enumerable set. In fact, we can show that Problem 5 can be solved by enhancing the *total expected future earnings* associated with *Get Passenger* action in the dynamic-programming routines we described in Section 4.2 with an optimization problem. We use an off-the-shelf minimizer to solve this optimization problem in the results presented in this chapter. However, using techniques introduced by Nilim and El Ghaoui [65], in the following section, we show that a bisection algorithm can approximate this problem within an accuracy ϵ in $O(\log(V_{\max}/\delta))$ time, where $V_{\max}$ is the maximum value of the value function.

4.4 Robust Dynamic Programming

In this section, we describe the setup of the robust finite horizon dynamic program and a bisection algorithm to solve it approximately. We demonstrate our approach for solving robust dynamic program using the *relocation* strategy. However, the approach

generalizes to all the other strategies described in the earlier sections. Without loss of generality, we make the assumption that all entries of the rewards matrix $\mathbf{R}$ are non-negative and finite.

4.4.1 Dynamic Program Formulation

We consider a case of an *adversarial* two-player game between the driver and the nature, where the driver seeks to maximize the minimum expected earnings, while the nature is the minimizing player. Nature does this by choosing the worst-case transition matrix $\mathbf{P} \in \mathcal{P}^\alpha$ at each step of the play. Thus, the worst case optimal value vector $\mathbf{v}(i,t)$ is given by,

$$\mathbf{v}(i,t) = \max_{a \in \mathcal{A}} \left[E(i,t,a) + \sigma_{\mathcal{P}_i^\alpha}\big(\mathbf{v}(i,t)\big) \right], \forall i \in \mathcal{X}, t \leq N$$

where $\mathcal{P}_i^\alpha$ refers to the set of the i-th rows of the transition matrices $\mathbf{P} \in \mathcal{P}^\alpha$ and

$$\sigma_{\mathcal{P}}(\mathbf{v}) = \inf(\mathbf{p}^\intercal \mathbf{v} : \mathbf{p} \in \mathcal{P})$$

is the *inner problem* to be solved in each step of the recursion. A corresponding optimal robust policy $\widehat{\pi}_*$ is obtained by solving,

$$\widehat{a}_*(i,t) = \arg\max_{a \in \mathcal{A}} \left[E(i,t,a) + \sigma_{\mathcal{P}_i^\alpha}\big(\mathbf{v}(i,t)\big) \right], \forall i \in \mathcal{X}$$

for all time steps. With trivial algebraic simplification, we can show that Equation (4.7) implies

$$\mathcal{P}_i^\alpha = \big\{ \mathbf{p} \in \Delta^n : \sum_j \mathbf{f}(j) \log \mathbf{p}(j) \geq \beta \big\} \tag{4.8}$$

where,

$$\beta = \frac{2\beta_{\max} - \mathrm{F}^{-1}_{\chi^2_{n-1}}(1-\alpha)}{2}$$

$\beta_{\max} = \sum_j \mathbf{f}(j) \log \mathbf{f}(j)$, and Δ^n denotes the probability simplex of size n.

4.4.2 Bisection Algorithm

In this section, we describe the bisection algorithm that can be used to approximately solve the robust dynamic program described above. The inner problem of the robust dynamic program can be formalized as,

$$\sigma^* = \min_{\mathbf{p}} \mathbf{p}^\intercal \mathbf{v} : \mathbf{p} \in \Delta^n, \sum_j \mathbf{f}(j) \log \mathbf{p}(j) \geq \beta.$$

The Lagrangian $\mathcal{L}$ associated with the inner problem is therefore,

$$\mathcal{L}(\mathbf{v}, \zeta, \mu, \lambda) = \mathbf{p}^\intercal \mathbf{v} - \zeta^\intercal \mathbf{p} + \mu(1 - \mathbf{p}^\intercal \mathbf{1}) + \lambda(\beta - \mathbf{f}^\intercal \log \mathbf{p})$$

where ζ, μ, and λ are Lagrange multipliers. The dual function d is the minimum value of the Lagrangian over $\mathbf{p}$ for $\zeta \in \mathbb{R}^n, \mu \in \mathbb{R}$, and $\lambda \in \mathbb{R}$. The optimal $\mathbf{p}^*$ minimizing the above dual function is obtained by solving $\frac{\partial \mathcal{L}}{\partial \mathbf{p}} = 0$, which gives us,

$$\mathbf{p}^*(j) = \frac{\lambda \mathbf{f}(j)}{\mathbf{v}(j) - \zeta(j) - \mu}.$$

Plugging in the value of $\mathbf{p}^*(i)$ into the dual function d gives us the dual problem,

$$\overline{\sigma} = \max_{\zeta, \mu, \lambda} \lambda(1 + \beta) + \mu - \lambda \sum_j \mathbf{f}(j) \log \left(\frac{\lambda \mathbf{f}(j)}{\mathbf{v}(j) - \zeta(j) - \mu} \right) : \lambda \geq 0, \zeta \geq 0, \mathbf{v} \geq \zeta + \mu \mathbf{1}$$

where the inequalities hold element-wise. As this problem has a feasible set with non-empty interior, there is no duality gap and $\overline{\sigma} = \sigma^*$. Moreover, by monotonicity argument, we can conclude that ζ is zero. Thus, the last constraint of the dual

problem can be expressed as $\mu \leq \mathbf{v}_{\min} = \min_i \mathbf{v}(i)$. Hence, we get,

$$\begin{aligned} \sigma^* &= \max_{\lambda,\mu} h(\lambda,\mu) \\ \text{where,} & \\ h(\lambda,\mu) &= \begin{cases} \lambda(1+\beta) + \mu - \lambda \sum_i \mathbf{f}(j) \log\left(\frac{\lambda \mathbf{f}(j)}{\mathbf{v}(j)-\mu}\right), & \text{if } \lambda > 0, \mu < \mathbf{v}_{\min}, \\ -\infty, & \text{otherwise} \end{cases} \end{aligned} \tag{4.9}$$

The gradient of h is given by,

$$\begin{aligned} \frac{\partial h(\lambda,\mu)}{\partial \lambda} &= \beta - \sum_j \mathbf{f}(j) \log\left(\frac{\lambda \mathbf{f}(j)}{\mathbf{v}(j)-\mu}\right) \\ \frac{\partial h(\lambda,\mu)}{\partial \mu} &= 1 - \lambda \sum_j \left(\frac{\mathbf{f}(j)}{\mathbf{v}(j)-\mu}\right) \end{aligned}$$

The optimal value of λ for a fixed value of μ, $\lambda(\mu)$ given by,

$$\lambda(\mu) = \left(\sum_j \frac{\mathbf{f}(j)}{\mathbf{v}(j)-\mu}\right)^{-1}$$

which further reduces the problem to a 1-dimensional optimization problem,

$$\sigma^* = \max_{\mu < \mathbf{v}_{\min}} \sigma(\mu)$$

where, $\sigma(\mu) = h(\lambda(\mu),\mu)$. $\sigma(\mu)$ is a concave function. Now, we may use a bisection algorithm to maximise this function.

To initialize the bisection algorithm, we need upper and lower bounds μ_+ and μ_- on a minimizer of σ. When $\mu \to \mathbf{v}_{\min}, \sigma(\mu) \to \mathbf{v}_{\min}$ and $\sigma'(\mu) \to -\infty$ Thus, we may set upper bound $\mu_+ = \mathbf{v}_{\min}$. The lower bound μ_- must be chosen such that $\sigma'(\mu_-) > 0$.

$$\sigma'(\mu) = \frac{\partial h}{\partial \mu}(\lambda(\mu),\mu) + \frac{\partial h}{\partial \lambda}(\lambda(\mu),\mu)\frac{\partial \lambda(\mu)}{\partial \mu}$$

By construction of $\lambda(\mu)$, the first term of the above derivative is zero. Furthermore,

$\frac{\partial\lambda(\mu)}{\partial\mu} < 0$. Hence, we need μ such that $\frac{\partial h}{\partial\lambda}(\lambda(\mu),\mu) < 0$. Using the bounds on $\lambda(\mu)$, $\mathbf{v}_{\min} - \mu \leq \lambda(\mu) \leq \mathbf{v}_{\max} - \mu$, we can show that,

$$\begin{aligned}
\frac{\partial h}{\partial\lambda}(\lambda(\mu),\mu) &= \beta - \sum_j \mathbf{f}(j)\log\left(\frac{\lambda(\mu)\mathbf{f}(j)}{\mathbf{v}(j)-\mu}\right) \\
&\leq \beta - \beta_{\max} - \log\left(\sum_j \frac{\mathbf{f}(j)\lambda(\mu)}{\mathbf{v}(j)-\mu}\right) \\
&\leq \beta - \beta_{\max} - \log\left(\sum_j \frac{\lambda(\mu)}{\mathbf{v}(j)-\mu}\right) \\
&\leq \beta - \beta_{\max} - \log\left(\sum_j \frac{\mathbf{v}_{\min}-\mu}{\mathbf{v}(j)-\mu}\right) \\
&\leq \beta - \beta_{\max} - \log\left(\sum_j \frac{\mathbf{v}_{\min}-\mu}{\mathbf{v}_{\max}-\mu}\right) \\
&< \beta - \beta_{\max} - \log\left(\frac{\mathbf{v}_{\min}-\mu}{\mathbf{v}_{\max}-\mu}\right)(\mathrm{n} > 1)
\end{aligned}$$

Therefore, the sufficient condition is,

$$\begin{aligned}
0 &> \beta - \beta_{\max} - \log\left(\frac{\mathbf{v}_{\min}-\mu}{\mathbf{v}_{\max}-\mu}\right) \\
\log\left(\frac{\mathbf{v}_{\min}-\mu}{\mathbf{v}_{\max}-\mu}\right) &> \beta - \beta_{\max} \\
\left(\frac{\mathbf{v}_{\min}-\mu}{\mathbf{v}_{\max}-\mu}\right) &> e^{\beta-\beta_{\max}} \\
(\mathbf{v}_{\min}-\mu) &> e^{\beta-\beta_{\max}}(\mathbf{v}_{\max}-\mu) \\
\mathbf{v}_{\min} - e^{\beta-\beta_{\max}}\mathbf{v}_{\max} &> \mu(1-e^{\beta-\beta_{\max}})
\end{aligned}$$

Hence,

$$\mu_-^0 = \frac{\mathbf{v}_{\min} - e^{\beta-\beta_{\max}}\mathbf{v}_{\max}}{1-e^{\beta-\beta_{\max}}} > \mu$$

By construction, the interval $[\mu_-^0, \mathbf{v}_{\min})$ is guaranteed to contain the global maximiser

of σ over $(-\infty, \mathbf{v}_{\min})$. The bisection algorithm is as follows:

1. Set $\mu_+ = \mathbf{v}_{\min}$ and $\mu_- = \mu_-^0$. Let $k = 1, \mu_1 = (\mu_+ + \mu_-)/2$.

2. While $k \leq N$:

   (a) If $\sigma^{'}(\mu_1) > 0$, set $\mu_- = \mu_1$, if $\sigma^{'}(\mu_1) < 0$ then $\mu_+ = \mu_1$, else break

   (b) $k = k + 1$,

   (c) $\mu_k = (\mu_+ + \mu_-)/2$

3. $\hat{\mu}^* = \arg\max_j \{\sigma(\mu_j)\}$

Lemma 3. *After* $N \approx \log_2(V/\delta)$ *where* $V = \max\big(\sigma^* - \sigma(\mu_+), \sigma^* - \sigma(\mu_-)\big)$, *the bisection algorithm provides optimal solution to the inner problem within an accuracy* $\delta > 0$, *i.e.,* $\sigma^* - \sigma(\hat{\mu}^*) \leq \delta$ *which we call as the* δ*-solution.*

Proof. Let the interval $[\mu_-, \mu_+]$ from step 1 of the bisection algorithm be denoted by G. At each iteration, the length of the interval that contains the global maximiser of σ is exactly halved. Hence, let $[\mu_{N-}, \mu_{N+}]$ be the corresponding interval after N iterations of the bisection algorithm. Length of the interval G_N is 2^{-N} times the length of G. Using the bisection algorithm, we know that,

$$\forall \mu : \mu \in G \setminus G_N \rightarrow \sigma(\mu) \leq \sigma(\hat{\mu}^*)$$

Let $2^{-N} < \alpha < 1$. α-contraction of G to μ^* is the segment given by points,

$$G^\alpha = (1-\alpha)\mu^* + \alpha G = \big\{(1-\alpha)\mu^* + \alpha z \big| z \in G\big\}$$

Since, $\alpha > 2^{-N}$, we know that length of $G^\alpha > G_N$, in fact, length of G^α is $\alpha(\mu_+ - \mu_-)$. Hence, $\exists y \in G^\alpha$ such that $y \notin G_N$. Furthermore, $\exists z \in G$ such that $y = (1-\alpha)\mu^* + \alpha z$. By the concavity of $\sigma(\mu)$ function,

$$\begin{aligned} \sigma(y) &\geq (1-\alpha)\sigma(\mu^*) + \alpha\sigma(z) \\ \sigma(\mu^*) - \sigma(y) &\leq \alpha(\sigma(z) - \sigma(\mu^*)) \\ &\leq \alpha V \end{aligned}$$

Hence, y is an αV-solution to our problem.

$$\begin{aligned} \sigma(\hat{\mu}^*) &\geq \sigma(y) \\ \sigma(\mu^*) - \sigma(\hat{\mu}^*) &\leq \sigma(\mu^*) - \sigma(y) \leq \alpha V \end{aligned}$$

Hence, sufficient condition for obtaining a δ-solution is, $\alpha V \leq \delta$ i.e., $N \geq \log_2(V/\delta)$. However, V is unknown by itself. But, the objective function of the inner problem, $\mathbf{p}^\intercal \mathbf{v}$, is bounded from above by $\mathbf{v}_{\max}$. Therefore, the lemma still holds for $V = \max\big(\mathbf{v}_{\max} - \sigma(\mu_+), \mathbf{v}_{\max} - \sigma(\mu_-)\big)$. □

Hence, the bisection algorithm provides an optimal solution to the inner problem within an accuracy $\delta > 0$. Given a $\mathbf{v} \in \mathbb{R}^n_+$ and $\delta > 0$, we can use the bisection algorithm above to find a solution $\hat{\sigma}_{\mathcal{P}}(\mathbf{v}) = \sigma_{\mathcal{P}}(\mathbf{v}) - \delta_{\mathcal{P}}(\mathbf{v})$ where $0 < \delta_{\mathcal{P}}(\mathbf{v}) \leq \delta$. For a total of N time steps in the finite horizon, setting $\delta = \epsilon/N$ for each timestep, we can compute an ϵ-suboptimal robust policy for the RobustEarnings problem.

4.5 Data and Experiments

We now evaluate our strategies for drivers in practice. First, we discuss how we collect and combine the appropriate data from multiple data sources. Then, we perform a comprehensive experimental study that provides specific insights as to how New York City drivers can maximize their earnings.

4.5.1 Data Pre-processing

In order to evaluate our strategies, we need to construct time-evolving matrices $\mathbf{F}$, $\mathbf{T}$ and $\mathbf{R}$ as defined in Section 4.1, and $\mathbf{C}$ as defined in Section 4.3. For this, we use two data sources: (1) the New York City taxi rides dataset [2] and (2) information we obtain from the Uber platform via queries to the Uber API. [3]

[2] `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`

[3] `https://developer.uber.com/docs/riders/ride-requests/tutorials/api/introduction`

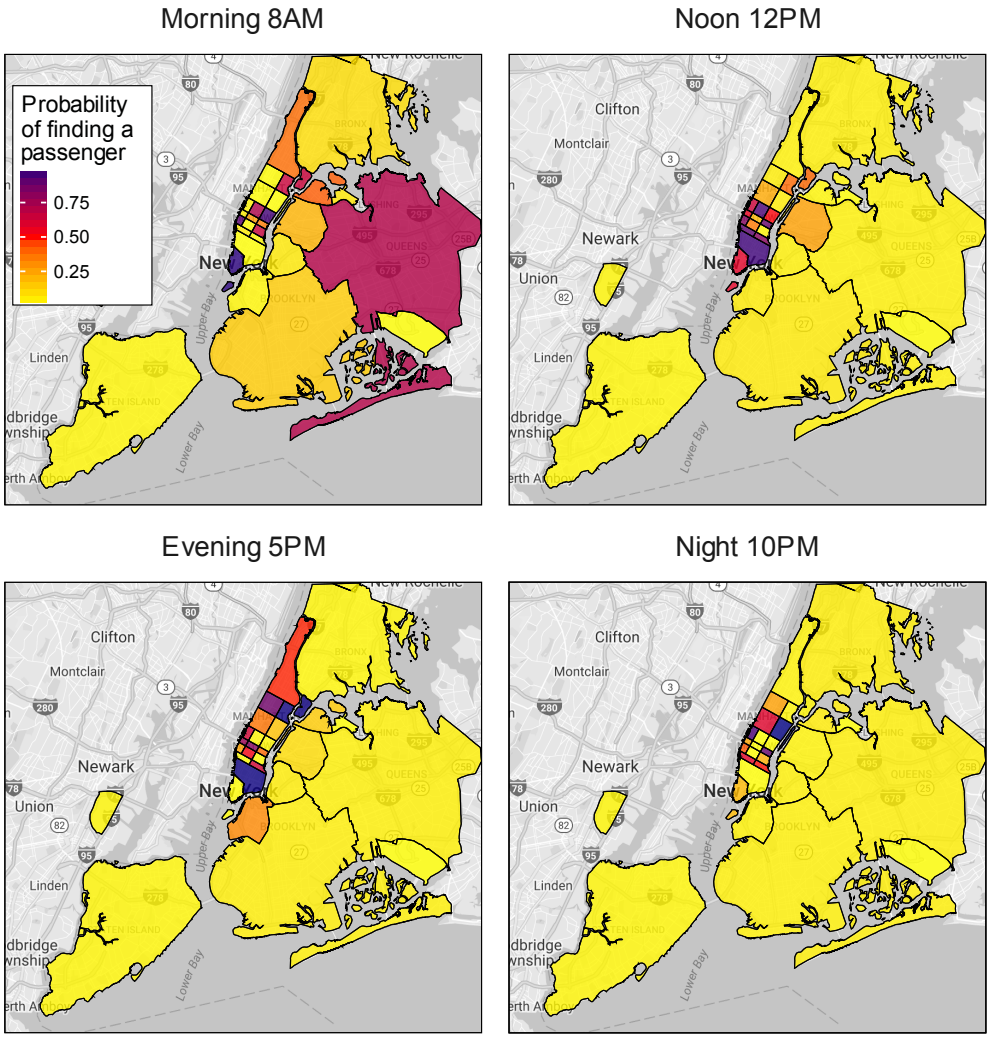

Figure 4·1: Probability of finding a passenger in 10 minutes across New York City zones at different times of a representative day.

Forming time-evolving matrices C and F

Our starting point is the New York City Taxi dataset (2015-2016), which contains yellow street-hail records of over 200,000 taxi rides per day with fields capturing pickup and dropoff times, location co-ordinates, trip distances, and fares. Each taxi record is accompanied with a taxi location ID for the pick-up and drop-off locations. Each location ID is associated with one of 29 non-overlapping city zones, as defined in the dataset. While the set of taxi rides is undoubtedly produced from a different ridership than Uber, it nonetheless provides a useful baseline that reflects many of the broader dynamics of ridership demand in New York City.

Given this data, we divide each 24-hour day of the week into 144 time-slices of duration 10 minutes each, indexed by their start time. To model traffic demand in the city at time t, the $c(i, j)$ entry of count matrix $\mathbf{C}^t$ is the total number of rides from zone i to zone j in a 30-minute long time window centered at time t. For example, $c(i, j)$ for the time slot [10:40, 10:50] on a Wednesday is a count of all rides from i to j that were initiated between 10:30 and 11:00 on *any* Wednesday in the dataset. Since our model disallows rides within the same zone, we ignore such rides while populating the entries of the matrix $\mathbf{C}^t$, resulting in all diagonal entries of the count matrix being zero.

To populate the entries of the empirical transition matrix $\mathbf{F}^t$, as defined in Section 4.1, we must estimate its diagonal entries, which correspond to the probability of not finding a ride, as well as the transition probabilities. We derive these from the data as follows. Assuming that the parameters do not change significantly within a single time-slice, let $N(\lambda)$ and $N(\mu)$ denote the number of passenger and driver arrivals in zone i in one time unit, with independent Poisson arrival rates λ and μ respectively. Although we assume the independence of the passenger and driver arrival processes, we can also accommodate correlated processes with slight modification. Hence, the

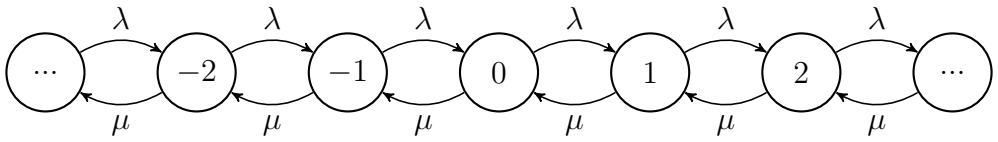

Figure 4·2: States of a skellam-distributed random variable K.

random variable $K = N(\lambda) - N(\mu)$ follows a Skellam distribution such that:

$$\Pr[K = k] = e^{-(\lambda+\mu)} \left(\frac{\lambda}{\mu}\right) I_k\big(2\sqrt{\lambda\mu}\big)$$

where $I_k(z)$ is the modified Bessel function of the first kind [92]. States of the random variable K are depicted in Figure 4·2.

Whenever $K < 0$, there are more drivers than passengers. We assume a worst case scenario in which a driver (conceptually) joins the end of a FIFO queue for that zone. Hence, for $k \leq 0$, the driver has to wait for $(|k| + 1)$ passenger arrivals for a successful passenger pickup. Then, the probability of a successful passenger pickup is:

$$\Pr[N(\lambda) = |k| + 1] = \frac{\lambda^{\left(|k|+1\right)} e^{-\lambda}}{\big(|k| + 1\big)!}.$$

Thus, we can express a diagonal entry $f^t(i, i)$ as follows:

$$f^t(i, i) = 1 - \sum_{k \leq 0} \Pr[K = k] \times \Pr[N(\lambda) \geq |k| + 1].$$

For $\mathbf{F}$ to be stochastic, we set every other entry $f^t(i, j)$ to:

$$f^t(i, j) = (1 - f^t(i, i)) \times \frac{c^t(i, j)}{\sum_j c^t(i, j)}.$$

The matrix $\mathbf{F}^t$ built in this manner satisfies all our assumptions.

Figure 4·1 shows an example of varying estimated probabilities of successful pick-ups in different zones at various times of the day derived from the New York City data

using the methods above. As expected, we see that the probability of a successful pickup is higher outside Manhattan in the morning, and this trend reverses in the evening.

Forming time-evolving matrices T and R

We obtain information regarding travel times and rewards using the `estimates/price` endpoint of the Uber API. The API takes longitude and latitude of pick-up and drop-off locations and returns price estimates for all types of Uber products – UberX, UberXL and UberBlack – together with the active surge multiplier rate at the pick-up location at the time of query. We only focus on UberX, the most popular Uber product. We also use the `/products` API endpoint to get information on the base fare, minimum fare, cost per minute and cost per unit distance for UberX. However, none of the Uber API endpoints provide information about the supply of drivers or demand of passengers; we impute this information from the New York City taxi rides dataset.

To create a representative sample of the data, we "recreated" New York City taxi rides virtually on the Uber platform. Using the Uber API, we were able to take a New York City taxi ride recorded in 2015, and capture the Uber attributes of that ride exactly one year later, collecting price estimates and other data above for that virtual ride. To respect the Uber rate limit of 1,000 API requests per hour per account, we sub-sampled one ride between each pair of zones in the city every 15 minutes. We implicitly assume that price estimates, travel times, and distance of preferred travel paths by drivers do not vary significantly in 15 minutes. Every 5 minutes, we also queried the surge multiplier active within each zone. Chen *et al.* [18] have observed that 90% of the surges on Uber platform have durations lasting multiples of 5 minutes. Using this approach, we collected data from the Uber API for a 6-month period (Oct. 2016–Mar 2017), recreating rides that originally occurred from Oct. 2015 to Mar

2016. Thus, we built realistic estimates for $r(i, j)$ and $\tau(i, j)$ for all pairs of zones We take into account the Uber fee structure in New York City as reported by the Uber API, as well as the overall *cost per mile* estimates provided by the American Automobile Association (AAA) in order to build estimates for $r(i, j)$. Finally, we maintained same-day of week estimates, so that, for example, travel time estimates and rewards computed for Sunday, Oct 16, 2016, were paired with frequency estimates drawn from the New York City taxi rides dataset for Sunday, Oct. 18, 2015. In the remainder of this section, we provide results for driving during one representative week in October. Our results do not vary qualitatively across different weeks, with the exception of seasonal peak days, such as New Year's Eve.

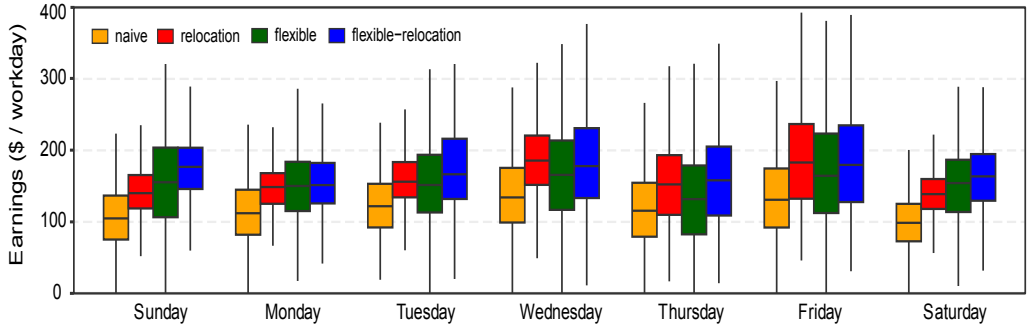

Figure 4·3: Daily driver earnings for different strategies averaged over different home zones on a representative day.

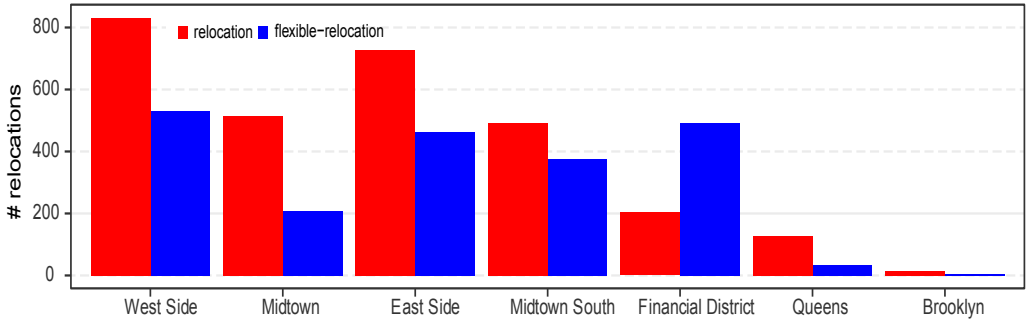

Figure 4·4: Contrast between preferred relocation destinations for drivers with *relocation* and *flexible-relocation* strategies on a representative day.

4.5.2 Experimental Results

For all our experiments we use a single process implementation of our algorithms on a 24-core 2.9GHz Intel Xeon E5 processor with 512GB memory. Running time for *naive* and *relocation* is less than a minute, and about 5 minutes for *flexible* and *flexible-relocation*. Uncertainty analysis (Section 4.5.2) with an off-the-shelf minimizer takes around 3 hours. Our code has been made publicly available in order to encourage reproducible research [14].

Comparison of strategies

First, we address the question: *what is the best driver strategy?* Intuitively, it is clear that *flexible-relocation* is the best strategy, as it takes advantage of spatial as well as temporal variations in the passenger demand across New York City. In order to verify this intuition, we compare driver earnings across different strategies. Drivers following the *naive* and the *relocation* strategies are assumed to drive from 9 AM to 5 PM, a standard 8 hour workday, while those following the *flexible* or the *flexible-relocation* strategies drive for a total of 8 hours each day with a flexible schedule.

In order to evaluate the performance of our strategies, we find the solution to MaxEarnings and simulate 100 drivers, each randomly assigned a home zone, operating on these strategies on the same day of the following week, for a total of 10 weeks. Figure 4·3 presents a box-plot of the resulting earnings. The lower and upper edges of the boxes in Figure 4·3 indicate quartiles Q1 and Q3 respectively, and length of whisker is 1.5 times IQR.

We observe that all "smart" strategies consistently outperform *naive*; as expected. On most days, *flexible-relocation* is the strategy with the highest earnings. The median earning of a driver following the *naive* strategy on a Sunday is \$104 while that of a driver following *flexible-relocation* is \$177, representing a 70% increase in

median earnings. Averaged over all days of the week, this results in a 47% increase in median earnings per work day when following the *flexible-relocation* strategy. Thus, our strategies do exploit the spatial and the temporal variation in demand across New York City. The results also show that for a part-time Uber driver in New York City, it is more beneficial to drive midweek, from Wednesday to Friday, and Sunday than during Saturday and Monday.

Spatial dynamics of strategies

Next, we address the question - *what are the benefits of the Relocate action?* Figure 4·1 already shows the spatial variation in the demand across different New York City zones at different times of the day. Intuitively, this spatial variation can cause a disparity in the driver earnings based on the zone of the driver. For example, drivers based in Manhattan should be expected to earn more than those based in Brooklyn due to persistently higher demand in Manhattan. Similarly, Figure 4·3 shows temporal variation in earnings across days of the week. We observe that on the days of low-demand, not only are the median earnings for *relocation* consistently higher than those for *naive* but also the inter-quartile range (IQR) and the length of whiskers for *relocation* are narrower. On days with high but localized demand like Fridays, the *relocation* strategy performs on par with the *flexible-relocation* strategy and significantly outperforms *naive*.

These observations indicate that the location-based disparity in earnings for the *naive* strategy is much larger than the *relocation* strategy. Thus, we conclude that smart relocations throughout the day prevent a driver from becoming "trapped" in low-earning neighborhoods, translating into significant increases in the earnings. This may be counterintuitive to some drivers, as a *Relocate* action (essentially an empty ride) incurs a cost to the driver. Yet, the results demonstrate that these actions, when timed appropriately, lead to earnings far higher than the costs they incur.

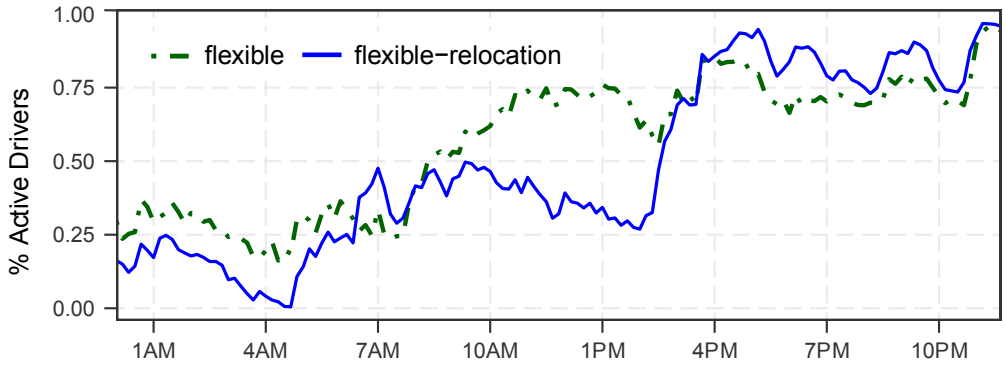

Figure 4·5: Active drivers with *flexible* and *flexible-relocation* strategies at different times of a representative day. Drivers adopting *flexible-relocation* strategy are more active during the evening hours; x-axis: time of the day, y-axis: percentage of drivers active on the platform.

Temporal dynamics of strategies

Intuitively, due to the periodicity of demand, we expect driver earnings to strongly depend on the time of the day they are driving. Thus, we address the question: *what is the best time of the day to drive in order to maximize earnings?* To answer this, we simulate 1000 drivers, each randomly assigned a home zone, for each of the *flexible* and *flexible-relocation* strategies. We solve the MaxEarnings problem for both strategies and create a recommended plan of action for the simulated drivers. Then, at every step of the simulation, a driver undertakes the personalized action recommended by the strategy, corresponding to their location, the time of day and their budget remaining.

In Figure 4·5, we plot the percentage of simulated drivers driving in the city at different times of the day. We observe a noticeable difference between the "preferred" driving schedules output by *flexible* and *flexible-relocation*. In particular, a high percentage of *flexible* schedule drivers are active during the standard working hours of the day from 9AM to 6PM. This also supports our choice to evaluate fixed schedule strategies in the interval 9AM to 5PM. In contrast, the number of active drivers that

follow *flexible-relocation* exhibits two distinct peaks, corresponding to the morning and the evening rush hours. Furthermore, over 50% of *flexible-relocation* drivers use their driving budget in the latter half of the day starting approximately at 3PM, continuing through until midnight. Since *flexible* and *flexible-relocation* only differ in the *Relocate* action, all observed differences are due to this action. Hence, we can conclude that the *Relocate* action is most effective in the evening hours, thereby prompting higher active percentages of *flexible-relocation* drivers at that time.

Preferred relocation zones

By simulating drivers, we can also compare the *Relocate* actions between drivers following the *relocation* strategy and those following the *flexible-relocation* strategy. The contrast between popular destinations of *Relocate* actions for drivers following the two strategies can be seen in Figure 4·4. Drivers following the *relocation* strategy predominantly relocate themselves to the center of Manhattan. In contrast, the drivers following the *flexible-relocation* strategy do not exhibit a clear most-preferred relocation destination. Furthermore, the *number* of relocations performed by the *relocation* strategy drivers, surprisingly, is significantly higher than those performed by the *flexible-relocation* strategy drivers. This is due to the flexible work schedule of the latter, which allows them to drive continuously during the hours of highest demand, reducing the frequency of *Relocate* actions they take.

Surge chasing

We now turn our attention to surge pricing. Surge pricing is a feature of the Uber platform aimed at matching supply with passenger demand by increasing prices at times of high demand. According to Uber, it incentivizes drivers to start driving during the peak hours in order to efficiently meet demand with supply, albeit at a higher cost to passengers. It also decreases demand, since more price-sensitive

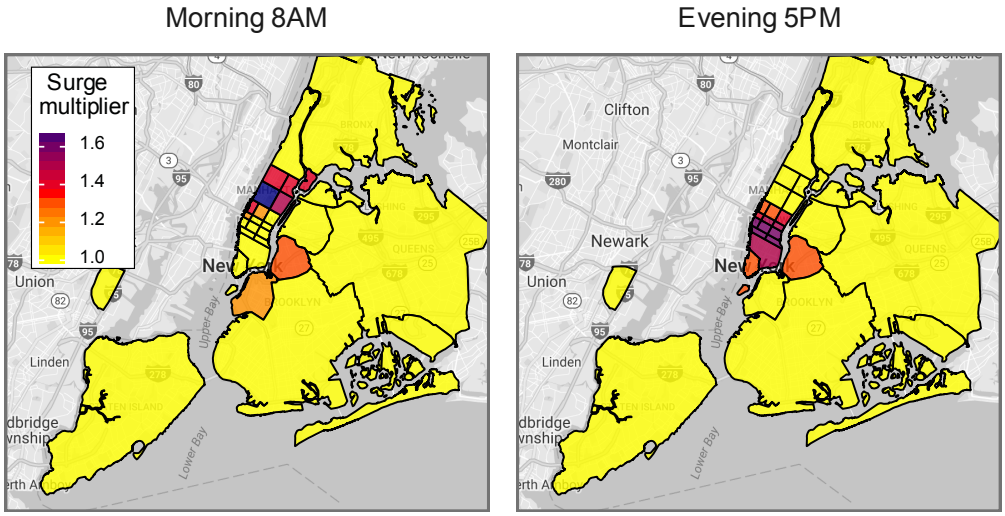

Figure 4·6: Active surge multiplier across New York City zones at different times of a representative day.

customers drop out, as surge prices rise.

Figure 4·6 shows the active surge multiplier across different neighborhoods of New York City at different times of the day. This information is readily available to the drivers; however, due to uncertainty in the duration of surges as well as the proprietary nature of Uber's surge pricing algorithm, it is unclear whether drivers should relocate themselves to surging areas in order to maximize their earnings.

Next, we address the question– *Should drivers engage in surge chasing?* In order to do so, we evaluate earnings of simulated drivers in three scenarios viz., "no surge" - where we disable the surge multiplier to compute earnings; "surge" - where the multiplier is used while calculating earnings; and "surge chasing" - wherein a driver located in a non-surging zone always relocates to the zone with highest surge multiplier within a 10-minute drive radius. Simulated driver earnings in these three scenarios for each of the strategies are shown in Figure 4·7. We observe that blind "surge chasing" leads to lower earnings irrespective of the strategy being followed. Figure 4·7 reinforces our previous observation regarding the high variance of the *naive* strategy. At times,

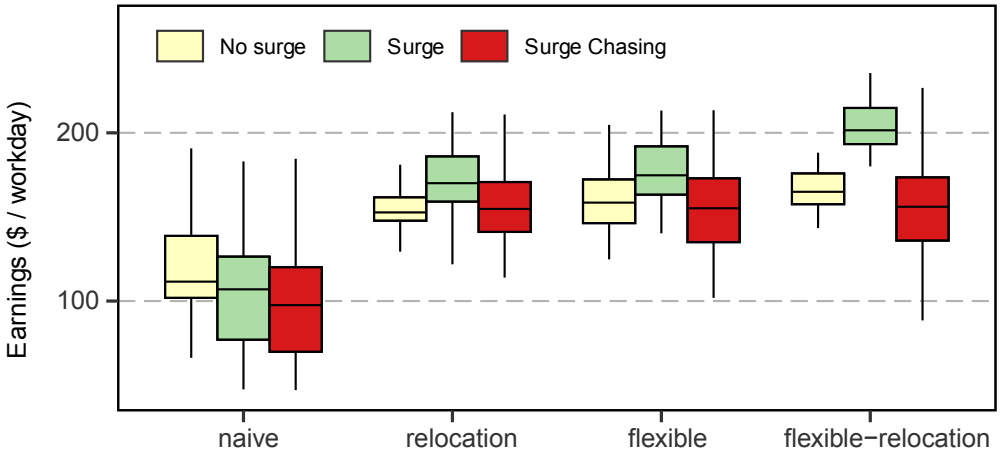

Figure 4·7: Exploring surge: Naively 'chasing surge' is sub-optimal in terms of improved earnings in comparison to proposed strategies; x-axis: driver strategies, y-axis: earnings of simulated drivers over an 8-hour workday.

drivers following the *naive* strategy with surge multiplier enabled may earn less than when it is disabled. For other strategies, "surge chasing" consistently fails to provide any tangible benefits as compared to following the pre-determined strategy. We conclude that actively and blindly chasing the surge is an ill-advised strategy and may lead to losses. Furthermore, surges last for short durations, and an unsuccessful surge chase may land a driver in a sub-optimal location with respect to longer term earnings. Note that although the New York City taxi demand data strongly correlates with active surge multipliers, we do currently model the impact of surge multiplier on consumer demand. This should be considered a limitation of our study.

Effect of uncertainty

Our experiments indicate that our strategies always outperform a *naive* strategy that is likely prevalent among Uber drivers. However, all our strategies use historical data. Consequently, our results can potentially be sensitive to perturbations of the empirically-observed transition matrices. Thus, we can only conclude that our results

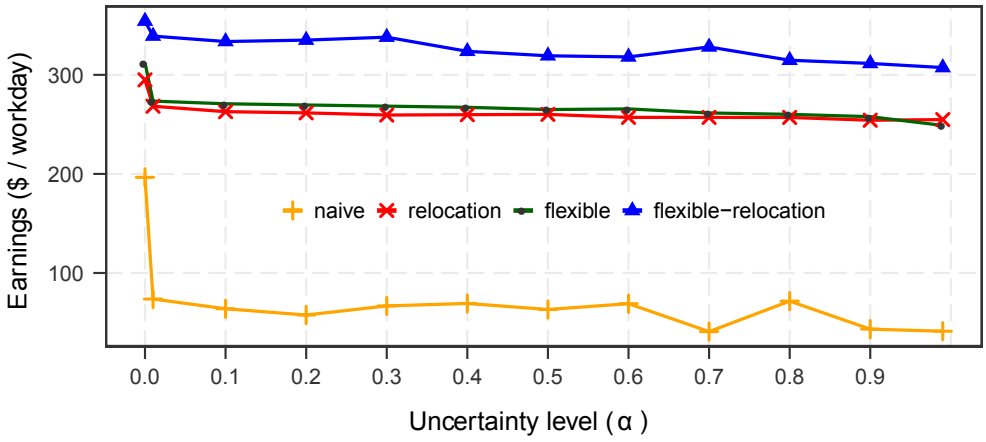

Figure 4·8: Sensitivity to uncertainty in parameters. Even in presence of high levels of uncertainty in the input data, the proposed strategies continue to outperform a naive strategy with perfect knowledge; x-axis: level of uncertainty in the input data, y-axis: earnings of simulated drivers over an 8-hour workday.

are robust if the drivers following one of the *relocation*, *flexible* and *flexible-relocation* strategies have higher earnings than those following *naive*, even when the input data is perturbed.

Hence, the question we have to address is the following: *Are the conclusions we drew above robust to perturbations of the empirical transition matrices?* We do so using the framework we developed in Section 4.3: we solve the RobustEarnings problem for each of the four strategies for increasing levels of uncertainty (α) using the Sequential Least Squares Programming (SLSQP) minimizer implementation provided by Jones *et al.* [87].

Figure 4·8 shows the effect of increasing uncertainty on the earnings of drivers for each of the four strategies. We observe two main takeaways. First, we find that all strategies suffer a loss under small amounts of uncertainty, even at levels of α in the range of 0.02, so all strategies are tuned closely to the empirical data. However, all strategies then remain resilient to a wide range of additional uncertainty, and we

find that the *relocation*, *flexible* and *flexible-relocation* strategies are most tolerant to uncertainty in the input transition matrices. Interestingly, even with 99% uncertainty, the *flexible-relocation* strategy significantly outperforms the *naive* strategy with no uncertainty. This observation further supports our claim that being strategic using historical data can significantly improve driver earnings in on-demand ride-hailing platforms.

Chapter Summary

This chapter focussed on maximizing the earnings of an individual self-interested driving partner of a ride-hailing MoD system. We showed that adopting a data-driven strategic approach while driving on such platforms can significantly increase daily earnings. Moreover, we favorably evaluated the impact of perturbations in data distribution on the effectiveness of such strategies. One of the limitations to our approach is that their effectiveness decreases with an increasing number of drivers adopting the same strategy. We resolve this difficulty in Chapter 5, where we maximize the earnings of an entire fleet.

Chapter 5

Fleet Management

Popular ride-hailing platforms, e.g., Uber, Lyft, Didi Chuxing, and Ola, have revolutionized the daily commute in cities across the world. Globally valued at over $61 billion and expected to grow to over $200 billion by 2025 these platforms operate as multi-sided marketplaces, seamlessly connecting drivers with riders through their smartphone applications [11]. The explosive growth of these ride-hailing platforms has motivated a wide array of questions for academic research at the intersection of computer science and economics, as we discuss in the related work section.

A large segment of these works aims to improve the performance of the platforms, ensuring high-reliability service for the passengers alongside high utilization and earnings for the drivers. The two main thrusts are *dynamic pricing* and *capacity repositioning*. Dynamic pricing [96, 4, 2, 13, 30] aims to balance demand and supply by increasing prices in certain neighborhoods. Intuitively, temporary increases in prices curtail price-sensitive demand and assist the platform in ensuring a high-reliability service. On the flip side, the potential for higher earnings also encourages more drivers to join the platform during such "price surges". The dynamic pricing literature uses game-theoretic analyses of the ride-hailing markets to show its effectiveness as a platform control mechanism.

The capacity repositioning approach aims to improve the platforms' performance by assisting drivers with recommendations for relocations inside a city. Although the initial work in this domain has focused on modeling the driver-repositioning problems

as combinatorial optimization problems [48, 101, 77, 95, 100], the need for optimizing large driver fleets and the availability of high-dimensional historical data has recently led to the development of machine learning methods for the same problem [63, 81, 57, 91, 89]. Such approaches predominantly use multi-agent coordination reinforcement learning to solve a global capacity repositioning problem, using various forms of *attention mechanisms* in neural networks to achieve coordination. By design, they make a key assumption that there is *always* a need for coordination in the market. This assumption necessitates them to leverage recent breakthroughs in the scalability of deep learning models to exploit the high-dimensional historical data. Deep-learning based methods have a large number of hyperparameters required in their training, making their performance susceptible to external perturbations. Moreover, the *black-box* coordination issue, as yet unresolved, is a potential liability when using deep-learning based systems in domains such as this, where platform controllers would like to understand how the choices of recommendations to human drivers were made.

We aim to revise the capacity-repositioning approach by relaxing the assumption that coordination is always necessary. Specifically, our approach leverages the observation that driver actions are independent at most times of the day, with coordination required only during periodic times of peak demand, such as rush hours. Furthermore, the instances of supply-demand imbalances in a city are usually restricted to distinct neighborhoods. We exploit this loose spatio-temporal coupling of supply and demand to learn *when* and *where* the drivers need to coordinate, and otherwise act independently for the rest of the time. This observation allows us to combine vanilla reinforcement learning (i.e., not deep learning) algorithms with simple combinatorial techniques for solving the repositioning problem. Moreover, our framework is scalable because the sizes of the combinatorial problems we need to solve in order to achieve

capacity repositioning are constrained by the number of imbalanced neighborhoods. Broadly, our model is a combination of the combinatorial and machine-learning approaches to capacity repositioning.

As our framework does not rely on deep learning, we are able to explain *ex-post* all the recommendations given to the drivers, taking a step in direction of *transparent* AI proposed in the recent *General Data Protection Regulations* (GDPR) guidelines [86]. Moreover, our approach is *envy-free* in the sense that drivers at the same location and time do not envy one another's future earnings. The resulting model is relatively parameter-free and hence generalizes well in presence of daily variations in supply and demand. Finally, our framework is amenable to integration with any dynamic-pricing models by easily augmenting our data with the effects of such a model.

Chapter Organization

The rest of the chapter is organized as follows. We begin by setting up the problem of fleet management in Section 5.1 where we discuss different modeling assumptions. Section 5.2 provides a methodical explanation of our proposed approach. We also discuss some of the pivotal components of our approach and contrast them with some of the related works. Finally, Section 5.3 concludes the chapter by designing representative experiments to evaluate our fleet management framework.

5.1 Problem Setup

In this section, we describe the basics of our problem setup and provide the necessary notation.

5.1.1 City Attributes

Throughout the chapter, we assume that the city is divided into a set of m non-overlapping hexagonal zones denoted by $\mathcal{H}$. We also assume that time t advances in

discrete time steps i.e., $\mathcal{T} = \{1, \cdots, T\}$, a standard industry practice [17]. Finally, we assume a total of n homogeneous drivers traveling between hexagon zones picking up and dropping off the passengers.

Our model uses the following city matrices and vectors that are time-varying; i.e., their entries change at every time step. However, for notational convenience, we do not introduce the time step t subscript unless required for context.

Demand matrix (D): A matrix $\mathbf{D} \in \mathbb{R}^{m \times m}$ such that each entry $d(h, h')$ denotes the number of passengers requesting a ride from zone h to zone h' at time t. With appropriately sized hexagonal city zones, we find that $\forall h \in \mathcal{H}, d(h, h) = 0$.

Travel time matrix (T): A matrix $\mathbf{T} \in \mathbb{R}^{m \times m}$ such that each entry $\tau(h, h')$ denotes the number of discrete time steps required for transiting from zone h to zone h'.

Reward matrix (R): A matrix $\mathbf{R} \in \mathbb{R}^{m \times m}$ such that each entry $r(h, h')$ denotes the net reward for a taxi driver carrying a passenger from zone h to zone h'. The net rewards include driver's earnings for delivering the passenger at the destination minus the sundries such as gas cost, vehicle depreciation, etc. Hence, each entry of the matrix is of form $r(h, h') = \text{earnings}(h, h') - \text{cost}(h, h')$.

Driver actions (A): At each time step t, a driver in zone h who is not currently on a trip can choose one of the two actions.

- **Wait:** A wait action $a(h, h)$ involves waiting for a passenger in the current zone i for the current time step. If successful, it can lead to a trip to some other zone h' with the driver earning a reward of $r(h, h')$. When the number of drivers choosing to wait in a zone exceeds the demand of the zone at the particular time, an unsuccessful wait may occur, and the driver earns a net reward of zero while staying in the same zone h for the next time step.

- **Relocate:** A relocate action $a(h, h')$ involves relocation *without a passenger*

from zone h to zone h'. Undertaking a relocate action costs a driver a value denoted by $\text{cost}(h, h')$.

Thus, we consider a total of $|\mathbf{A}| = m^2$ actions. In case of a relocate action or a successful passenger pickup to zone h', the driver is busy traveling for next $\tau(h, h')$ time steps and is presented with the next action choice at time $t + \tau(h, h')$, while in case of an unsuccessful wait, the driver chooses the next action at time $t + 1$.

5.1.2 Model Attributes

Using the city attributes from the previous section, we now define the attributes of our model:

Policy (π): A policy function $\pi : \mathcal{H} \times \mathcal{T} \rightarrow \mathbf{A}$ recommends the best action to drivers in every zone of the city at each time step, to maximize the model's objective function. We impose a constraint that all drivers in the same zone at the same time be recommended the same action unless driver coordination is required to resolve a supply-demand imbalance in the zone.

A driver i following a policy π performs location and time-dependent actions represented by a 3-tuple $\phi_t^\pi(i) = (t, h, a)$, where h and a are the location of the driver and the action chosen at time t respectively. We assume that if a driver is busy at time t, the corresponding 3-tuple is $(t, \varnothing, \varnothing)$.

Driver earnings ($\mathcal{E}$)

Let function $E(t, h, a)$ denote the net earnings of a driver on taking action a at time t while located in zone h. If the action leads a driver to zone h', $E(t, h, a) = r(h, h')$. In the case of the relocate action, net earnings simply constitute the cost of relocation i.e., $r(h, h') = -\text{cost}(h, h')$. We can denote the gross earnings of n drivers following a policy π by: $\mathcal{E}^\pi(n, \mathbf{D}, \mathbf{T}, \mathbf{R}) = \sum_{t=1}^{T} \sum_{i=1}^{n} E\big(\phi_t^\pi(i)\big)$, where $E(t, \varnothing, \varnothing) = 0$.

Supply (S)

A policy π induces the movement of drivers between different city zones through action choices. The supply, i.e., the number of drivers at zone h at time t induced by a policy π, is denoted using the supply function $S^{\pi}(t, h)$.

Demand fulfillment ($\mathcal{F}$)

A driver in zone h choosing the wait action $a(h, h)$ at time t is randomly matched with any of the $\sum_{h'} d_t(h, h')$ passengers requesting a ride in zone h at the same time. Hence, a policy π, via its supply function, induces a demand fulfillment function. Demand fulfilled in zone h at time t when drivers follow a policy π is denoted using the demand satisfaction function $F^{\pi}(t, h)$. Obviously, $\forall_{\pi \in \Pi} F^{\pi}(t, h) \leq \sum_{h'} d_t(h, h')$. Hence, total demand fulfilled over the course of time steps $t \in \mathcal{T}$ by n drivers following the policy π can be given by: $\mathcal{F}^{\pi}(n, \mathbf{D}, \mathbf{T}, \mathbf{R}) = \sum_{t=1}^{T} \sum_{h=1}^{m} F^{\pi}(t, h)$.

5.1.3 Problem Statement

Based on the above definitions, we now formulate the problem that we solve.

Problem 1(MaxEarnings): Given time-varying matrices $\mathbf{D}, \mathbf{T}, \mathbf{R}$ and the number of homogeneous drivers n, devise a policy π^* such that

$$\pi^* = \arg\max_{\pi \in \Pi} \mathcal{E}^{\pi}(n, \mathbf{D}, \mathbf{T}, \mathbf{R}). \tag{5.1}$$

Replacing the driver earnings ($\mathcal{E}$) by demand fulfillment ($\mathcal{F}$) in the Equation (5.1) above results in a variant of MaxEarnings problem, in which the goal is to maximize fulfilled rides, referred to henceforth as the MaxFulfillment problem.

Discussion

At a high level, our revenue maximization problem statement is similar to existing work in the area. However, our modeling assumptions are significantly more realistic than previous work, making our methods more amenable to operational deployment. For example, whereas our work assigns drivers to rides explicitly (and allocates rewards accordingly), previous work performs reward allocations proportionally, in a fluid model. In our model, when two drivers compete for a single ride from i to j, one of the two driver gets the ride, gets paid, and ends up in j after a travel time t. In the representative fluid model of Lin *et al.* [57], both drivers get half of the payment, and two halves of drivers (conceptually) transit to j in a fixed time step, regardless of the distance traveled. Although a fluid model such as this is tractable to solve for and optimize around, it has strong implications on the solution space, as it notably removes time-dependent and driver-dependent features from the model. Issues such as studying variance of driver earnings are not possible in these models, as all drivers starting at the same time and place will end up with identical (quantized) trajectories and earnings.

We also note that in our framework, a wait action for an individual driver is only successful if the driver is present in the same zone as the ride request. Hence, our framework cannot result into the *Wild Goose Chase (WGC)* phenomenon described by Castillo *et al.* [13], in which high demand causes depletion of idle drivers on the streets, leading to suboptimal FCFS matches where drivers spend a significantly higher duration of time en route to pick up passengers.

5.2 Learning Framework

In this section, we describe our approach for solving the MaxEarnings problem. Our method is a *model-based reinforcement learning* approach, and its description is

provided in Algorithm 3.

5.2.1 Model-based Reinforcement Learning Algorithm

As with any reinforcement learning approach, we train our model by allowing the drivers to repeatedly interact with an environment in form of the city's ride demand data from a representative day. Each interaction, which is T timesteps long, constitutes an *episode* of the training process. Each episode constitutes of 3 phases described below.

ALGORITHM 3: Model-based Reinforcement Learning Algorithm

```
1  Initialization Q_I(t,h,a) ← 0, Q_C(t,h,a) ← 0, ξ(t,h) ← 0;
2  for each episode e = 1, ··· , E
3      for each time step t = 1, ··· , T
4          for each driver i = 1, ··· , n
5              Generate two random numbers η_0, η_1 ∈ [0,1];
6              if η_0 ≤ ε
7                  Choose exploratory action;
8              else
9                  if η_1 ≤ ξ(t,h_i)
10                     a = Independent action a* from Q_I;
11                 else
12                     a = Coordinated action a^c from Q_C;
13             Receive reward E(t,h_i,a);
14             Compute rebalance matrix Z;
15     for each zone h ∈ H
16         ∀t,a update Q_I(t,h,a) ;
17         ∀t update degree of coordination ξ(t,h) ;
18         ∀t,a update Q_C(t,h,a) ;
```

Exploratory phase (lines 5-7)

During this phase of the algorithm, drivers exhibit an *exploratory behavior* by choosing a pseudo-random action with a probability ϵ. These randomly chosen actions allow the model to explore a larger portion of the policy space, preventing its pol-

icy from converging to a local minimum. This is similar to the ϵ-greedy behavior of Q-learning [80].

Exploitative phase (lines 9-12)

During this phase of the algorithm, the rest of the drivers exhibit an *exploitative behavior* using the policy learned up until the previous episode of training. The policy recommends exploitative actions to individual drivers based upon the time of the day and their locations, independently of each other, henceforth referred to as *independent actions*. However, certain recommended actions may result in supply-demand imbalances when a large number of drivers relocate to the same city zone with an insufficient demand, or too few of them relocate to a zone with excess demand. We postulate that explicit coordination is essential to prevent such supply-demand imbalances from occurring. Hence, we introduce the *degree of coordination (ξ)* - a probabilistic value that signifies the extent to which drivers located in the same city zone need to coordinate their actions. Whenever a zone has a positive degree of coordination, the exploitative actions recommended to a ξ fraction of drivers in the zone are derived from solving a reward-maximizing linear program, henceforth referred to as *coordinated actions*.

It should be noted that it is the explicit criterion for recommending a coordinated action that sets our approach apart from recent works in the field of deep reinforcement learning across different applications and domains.

Learning phase (lines 15-18)

Actions recommended in the exploratory and exploitative phases of the episode result in drivers picking up passengers or relocating themselves to different city zones, thereby observing rewards of their actions (line 13). The learning phase of the algorithm computes a rebalancing matrix (line 14) to use in conjunction with the observed

rewards to further improve upon the policy.

Having developed an intuition for the major building blocks of Algorithm 3, we now explain these phases in greater detail.

5.2.2 Exploratory Phase

Over the course of training, when a driver located in zone h chooses to explore, we model the probability of driver's exploratory ride distance using a Gaussian function with a random variable $K_{\geq 0}$. Specifically, the probability that a driver relocates to a zone at distance $k \geq 0$ is given by: $\Pr[K = k] = be^{-\frac{k^2}{2c^2}}$.

After sampling an exploration distance k, the driver chooses the actual destination by sampling uniformly at random from all zones at distance k. When $k = 0$, the driver chooses to wait in the current zone, while for $k > 0$, the driver chooses a relocate action. The experiments in this chapter were all conducted using $b = 0.7$ and $c = 1$ (chosen via grid-search), allowing explorations up to 3 hexagonal zones away. In contrast, [57] restricts drivers to single zone distance relocations, reducing their ability to learn policies that mitigate supply-demand imbalances by relocating supply from zones further away in a single timestep. Over the course of training, ϵ is annealed exponentially from 1 to 0, thereby outputting an entirely exploitative model at the end of the training.

5.2.3 Exploitative Phase

Exploitative behavior is manifested in the form of independent actions (line 10) and coordinated actions (line 12) when the degree of coordination is positive. We detail these next.

Choice of independent action

For each independent action chosen by a driver, we record the reward earned. This reward is then used to update the *value* of the action for the next episode, based on the learning rate (α) and the discount factor (γ). These values are stored in a Q-table denoted by $Q_I \in \mathbb{R}^{T\times m\times|\mathbf{A}|}$. For each zone h, at time t, the best independent action (a^*) is chosen by (line 10, Algorithm 3)

$$a^*(t,h) = \arg\max_{a\in\mathbf{A}_h} Q_I(t,h,a),$$

where $\mathbf{A}_h$ refers to the h-th row of $\mathbf{A}$.

Independent learning

Based upon the observations of drivers undertaking independent actions (both exploratory and exploitative), we update the independent learning matrix (Q_I) as described below.

Updating Q_I for wait actions

Let $\mathcal{W}_{(h,h')}$ denote the number of drivers choosing to wait in zone h at time t, and ending up in zone h'. A successful wait generates net earnings $E\big(t,h,a(h,h)\big) = r(h,h')$ and consumes a travel time $\tau(h,h')$, while an unsuccessful wait i.e., $h'=h$, generates zero net earnings and consumes one timestep. The utility of the wait action is therefore

$$\mathcal{U}_{(t,h,h)} = \sum_{h'} \mathcal{W}_{(h,h')}\Big[E\big(t,h,a(h,h)\big) + \gamma Q_I(t',h',a^*(t',h'))\Big]$$

where $t' = t + \tau(h, h')$ and we discount the future rewards with a factor γ. We use the utility of the wait action to update the entry $Q_I(t, h, h)$ as follows:

$$Q_I(t, h, h) \leftarrow (1 - \alpha) Q_I(t, h, h) + \frac{\alpha}{\sum_{h'} \mathcal{W}_{(h,h')}} \mathcal{U}_{(t,h,h)}. \tag{5.2}$$

Normalizing the update term by the number of drivers choosing the wait action captures the average utility of the wait action. The term $Q_I(t, h, h)$ on the right hand side of the equation denotes the values learned upto the previous episode of the training, and α is the learning rate.

Updating Q_I for relocate actions

Let $\mathcal{R}_{(h,h')}$ denote the number of drivers relocating from zone h to zone h'. The utility of such relocation is given by

$$\mathcal{U}_{(t,h,h')} = \mathcal{R}_{(h,h')} \Big[E(t, h, a(h, h')) + \gamma Q_I(t', h', a^*(t', h')) \Big],$$

where $t' = t + \tau(h, h')$. We use the utility of the relocate actions to update the entry $Q_I(t, h, h')$ of the independent table as follows:

$$Q_I(t, h, h') \leftarrow (1 - \alpha) Q_I(t, h, h') + \frac{\alpha}{\mathcal{R}_{(h,h')}} \mathcal{U}_{(t,h,h')}. \tag{5.3}$$

Using Equations (5.2) and (5.3), the Q_I matrix is updated in line 16 of Algorithm 3 using the evidence obtained via simulations in form of utilities $\mathcal{U}_{(t,h,h')}$ of both the wait and relocate actions.

Choice of coordinated action

The choice of coordinated action is more intricate and non-standard, and we next explain it in detail. To guide the coordinated behavior of drivers in line 12 of Algorithm 3, we solve a reward-maximizing rebalancing operation between city zones

experiencing supply-demand imbalances. There are two principal components driving the coordinated behavior: *degree of coordination* (ξ) which controls the need of coordination in a particular zone at a time, and *coordinated learning matrix* (Q_C) which determines the choice of action as a response to the need of coordination. Thus, each coordinated action is associated with a probability for it to participate in the rebalancing operation that is stored in the matrix Q_C. Note that Q_C contains learned probabilities, as against the usual action-value nature of Q_I.

Let the policy learned at the end of k-th episode during training be denoted by π_k. Following this policy induces a driver supply S^{π_k} during the $(k+1)$-th episode of training. For each zone h, at time t, the coordinated action (a^c) in line 12 of Algorithm 3 is obtained by uniformly sampling from the probability vector $Q_C(t,h)$.

Imbalance matrix (Δ)

A matrix $\Delta \in \mathbb{R}^{|\mathcal{T}| \times m}$ such that each entry $\delta(t,h)$ denotes the supply-demand imbalance experienced at zone h at time t during the $(k+1)$-th episode. Specifically, each entry of the imbalance matrix can be given by, $\delta(t,h) = S^{\pi_k}(t,h) - \sum_{h'} d_t(h,h')$. We mask the imbalance matrix using an imbalance threshold parameter Λ such that,

$$\delta(t,h) = \begin{cases} \delta(t,h) & \text{if } \left|\delta(t,h)\right| \geq \Lambda \\ 0 & \text{otherwise.} \end{cases}$$

Using this parameter allows us to control the level of imbalances that the framework should attempt to mitigate.

Rebalancing graph ($\mathcal{G}$)

Based upon the supply-demand imbalance matrix induced at the end of an episode, we create the *rebalancing graph* $\mathcal{G} = (V, E)$ consisting of imbalanced zones as nodes and edges as corresponding relocation actions between them. This is a bipartitle graph

with nodes $V = \{V_+ \cup V_-\}$ where V_+ is the set of nodes with excess supply, i.e., $\delta(t,h) > 0$ and V_- is the set of nodes with supply deficit, i.e., $\delta(t,h) < 0$. Thus each node $v_i \in V$ in the rebalancing graph is associated with three attribues: *imbalanced zone* (v_i^h), *time of imbalance* (v_i^t) and *magnitude of imbalance* $\big(\delta(v_i^t, v_i^h)\big)$. The edge set E consists of directed edges from the nodes in V_+ to nodes in V_- and they model feasible relocations. Thus: $E = \{e_{ij} : v_i \in V_+, v_j \in V_-, v_i^t + \tau(v_i^h, v_j^h) \leq v_j^t\}$. The travel time constraint filters out edges where a relocating driver from an excess supply node cannot reach the deficit node in time. Each edge e_{ij} is associated with utility:

$$\mathcal{U}_{(i,j)} = \underbrace{Q_I(v_j^t, v_j^h, v_j^h)}_{\text{wait action at } v_j^h} - \underbrace{\text{cost}(v_i^h, v_j^h)}_{\text{relocation cost}} - \underbrace{Q_I(v_i^t, v_i^h, v_i^h).}_{\text{wait action at } v_i^h}$$

Thus, the utility of an edge measures the net value for a driver relocating along it during coordinated behavior.

Rebalancing operation

Given a rebalancing graph $\mathcal{G}$, we wish to relocate drivers from supply excess zones to supply deficit zones. We aim to find a matching that maximizes the net reward of all relocations, in order to maximize the driver earnings. Such a rebalancing operation can be achieved by solving a *Minimum Cost Flow* problem expressed in the form of the linear program below.

$$\begin{aligned}
\text{maximize} \quad & \textstyle\sum_{e_{ij} \in E} f_{ij} \times \mathcal{U}_{(i,j)} \\
\text{s.t.,} \quad & \\
\forall e_{ij} \in E, \quad & f_{ij} \geq 0 \\
\forall v_i \in V_+, \quad & \textstyle\sum_{v_j \in V_-} f_{ij} \leq \delta(v_i^t, v_i^h) \\
\forall v_j \in V_-, \quad & \textstyle\sum_{v_i \in V_+} f_{ij} \leq |\delta(v_j^t, v_j^h)|
\end{aligned}$$

Here, we calculate the number of excess drivers who should relocate from an excess node to a deficit node and store it in the form of a flow vector $f \in \mathcal{R}^{|E|}$ indexed along the edges set such that f_{ij} denote the flow from v_i to v_j.

If the platform aims to maximize demand fulfillment, we can formulate it as a *Maximum Flow* problem by setting the utility associated with each edge $\mathcal{U}_{(i,j)} = 1$.

As the constraint matrices – in both problems – are unimodal, the solutions of the linear programs are integral flow vectors and are thus optimal. Note that the size of the constraint matrix increases with a decrease in the Λ parameter. However, we can greatly reduce the sizes of corresponding linear programs and hence the computation time by solving a set of smaller linear programs; one for each connected component of the rebalancing graph.

Coordinated learning

Based upon the computed imbalance matrix (Δ) and the solution to the rebalancing operation above, we are now in a position to update the coordinated learning matrix (Q_C) and the degrees of coordination (ξ) as described below. It should be noted that while the choice of coordinated action from the matrix Q_C is influenced by the reward-maximizing rebalancing described above, the degree of coordination ξ is merely influenced by the supply-demand imbalances induced as a result of the policy learnt so far.

Updating Q_C for rebalancing operation

We capture the rebalancing operation in form of a rebalance matrix $\mathbf{Z} \in \mathbb{R}^{|\mathcal{T}| \times m \times m}$ where each entry $\zeta(t, h, h')$ denotes a probability of a rebalancing relocation from zone

h to zone h' being required at time t. For every edge $e_{ij} \in E$, we update $\mathbf{Z}$ as follows,

$$\begin{aligned} \zeta(v_i^t, v_i^h, v_i^h) &= \frac{\delta(v_i^t, v_i^h) - \sum_{v_j \in V_-} f_{ij}}{\delta v_i^t, v_i^h} \\ \zeta(v_i^t, v_i^h, v_j^h) &= \frac{f_{ij}}{\delta(v_i^t, v_i^h)}. \end{aligned}$$

Using the rebalance matrix, we update Q_C in line 18 of Algorithm 3 as follows,

$$Q_C(t, h, h') \leftarrow (1 - \alpha) Q_C(t, h, h') + \alpha \zeta(t, h, h'). \tag{5.4}$$

Updating degree of coordination (ξ)

At the end of each training episode $(k+1)$, we use the realized imbalance matrix (Δ) to determine the degree of coordination required within each zone at every time step. We update the degree of coordination as follows:

$$\xi_{k+1}(t, h) = (1 - \alpha)\xi_k(t, h) + \alpha \mu(t, h), \tag{5.5}$$

where the rebalancing raio μ is computed as:

$$\mu(t, h) = \begin{cases} \frac{\delta(t,h)}{S^{\pi_k}(t,h)} & \text{if } \delta(t, h) > 0. \\ \frac{\left|\delta(t,h)\right|}{\sum_{h'} d_t(h,h')} & \text{if } \delta(t, h) < 0 \text{ and } \xi_k(t, h) > 0. \end{cases}$$

While the former condition encourages driver relocations in zones with supply excess, the latter condition discourages it in zones with supply deficit. Thus, we use Equation (5.5) to update the degree of coordination for each zone in line 17 of Algorithm 3.

Discussion

We conclude this section by highlighting some of the features of our approach that make it appealing to use in practice.

First of all, over the course of training, our approach learns by trialing driver

actions over historically observed demand data and recommends strategic relocations to drivers when there is enough evidence to do so. This is done *proactively*, i.e., with the goal of preventing such an imbalance from actually occurring. One should contrast this with other reinforcement learning based approaches [52, 57] that try to resolve the imbalance issues *ex-post*, or dynamic pricing based approaches which assume full knowledge of future demand [58].

On deployment, our model relies only on trends learned from the historical data. This design decision is motivated by the empirically observed strong periodicity in demand. It makes our model relatively parameter-free, thereby providing robustness to demand perturbations. This decision is validated by the model generalizability experiment in Section 5.3.2, where we show that the recommendations made by our algorithm are robust to the presence of perturbations in the demand. In contrast, deep reinforcement learning based approaches [57, 63, 81, 89, 91] require as inputs full knowledge of supply and demand distribution during deployment. The stochastic gradient-descent algorithm used during training of networks uniformly samples experiences observed under previous training policies, making it impossible to reliably trace back and explain the actions recommended to the drivers. Furthermore, this increases the sensitivity of model performance to the tuning of numerous hyperparameters, making them difficult to deploy in the real world.

A key characteristic of our approach is the explicit coordination achieved by solving a *minimum cost flow* problem, allowing all our recommendations to be easily traced back and explained.

5.3 Data and Experiments

In this section, we begin by describing the pre-processing we did in order to use the New York City Yellow taxi rides public dataset and then we evaluate our framework.

5.3.1 Data Pre-processing

To train our model, we need to construct the time-evolving city matrices - **D**, **R**, and **T** described in Section 5.1.

Hexagonal binning of New York City

We employ the popular methodology of hexagonal binning to discretize the city into a set $\mathcal{H}$ of 250 non-overlapping uniform-sized hexagonal zones. The distance from the center of a zone to its vertices is about 1 mile.

Forming time-evolving matrices

We begin with the New York City Taxi dataset (2015), which contains street-hail records of over 200,000 taxi rides per day with information regarding pickup and dropoff locations and times, fare, trip distances, etc., from before the significant confounding effects of ride-sharing platforms like Uber, Lyft, etc. For each ride in the dataset, we evaluate its pickup and dropoff zones based on location coordinates. Assuming that passengers do not hail a taxi for short distances, we ignore a small percentage of rides which begin and end within the same zone.

We discretize a 24-hour day into 288 time-slices of duration 5 minutes each, indexed by their start time. Thus, to populate the entries of the matrices **D**, **R** and **T** at time t, we use the rides from the dataset in the 5 minutes time-slice beginning at time t. Due to variations in the popularity of particular pickup and dropoff zones at specific times of the day, the **R** and **T** matrices obtained using this method are sparse. However, to compute the best policies, our framework requires the availability of complete information regarding rewards and travel times. Hence, we estimate the missing values in these matrices using linear regression models including fixed-effects for the time of the day, the source and destination zones To compute the travel time

entry $\tau(i,j)$ at time t, we fit a linear regression model

$$\tau(i,j,t) = \overline{\beta_0} X_{i,j,t} + \beta_1 \alpha_i + \beta_2 \alpha_j + \beta_3 \alpha_t + \epsilon_{i,j,t}$$

where $X_{i,j,t}$ are the time-variant predictors, the α_i, α_j, and α_t are time-invariant fixed-effects for source, destination and time of the day respectively, while $\epsilon_{i,j,t}$ is standard normal error. The performance of our model is not sensitive to the choice of a specific linear regression modeling technique.

5.3.2 Experimental Results

Settings

For all experiments, we use a multiprocessor implementation of our algorithm on a 24-core 2.9 GHz Intel Xeon E5 processor with 512 GB memory. The model training time for 100 episodes of training takes less than an hour. The model testing time is less than 5 minutes. Our code has been made publicly available for reproducibility purposes [34]. All our experiments use learning rate $\alpha = 0.01$ and discount factor $\gamma = 0.99$. During independent learning, the exploration factor (ϵ) used in ϵ-greedy Q-learning decreases exponentially as the training progresses. Unless mentioned otherwise, we train 5,000 drivers over 200 episodes and set the imbalance threshold (Λ) to 2. Experimental results presented in this chapter are obtained by training models over a representative day viz., first Monday of September 2015 with a demand of over 232,000 rides. However, our results generalize to any day.

Model performance

First, we address the question: *how well does our reinforcement learning-based algorithm learn the driver dispatch policy?* In Figure 5·1, we observe the improvement in mean driver earnings and demand fulfillment as the training progresses. We split the 200 training episodes into independent learning episodes ($E_{IL} = 160$) and coor-

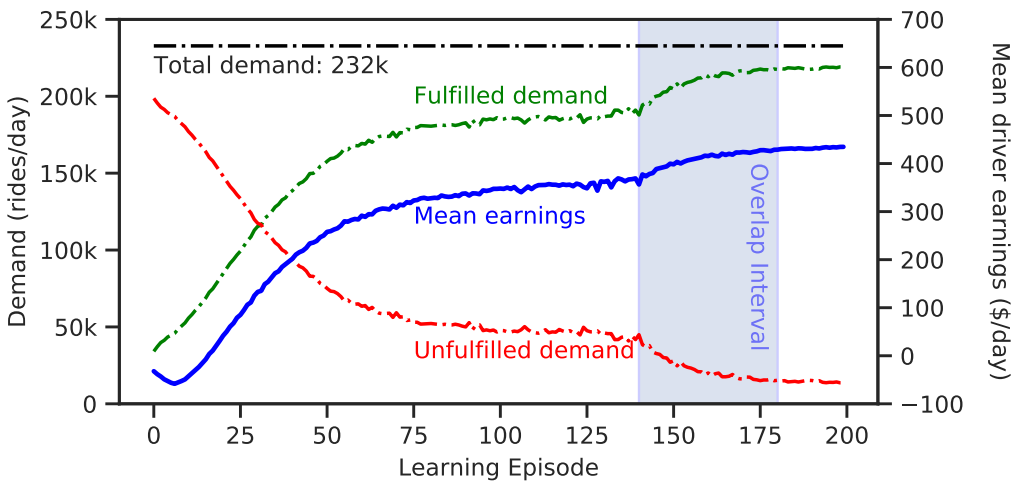

Figure 5·1: A representative illustration of improvement in mean driver earnings during training; x-axis: learning episode, y-axis($l$eft): Number of ride requests (dotted black, green and red lines), y-axis($r$ight): average earnings (solid blue line) of the simulated drivers over an entire day, *overlap interval* refers to episodes where both independent and coordinated behavior learning are being learned.

dinated learning episodes ($E_{CL} = 60$). This can be achieved by setting the degree of coordination (ξ) to 1 until episode number $E - E_{CL}$ on line 9 of Algorithm 3. Consequently, episodes $[140, 160]$ utilize both independent and coordinated learning. In Figure 5·1, we observe a significant improvement in the objective in the interval denoted by a shaded region. As expected, coordinated learning appropriately relaxes some of the constraints imposed by single-agent MDP and leads to significantly better performance.

In Figure 5·2, we plot the total demand at various times in the day, along with its fulfilled and unfulfilled portions by drivers following our policy. About 95% of the total demand during the day is satisfied with our framework. We consider a ride request fulfilled if an idle driver is present in the same zone at the time of the request. We find that 10% of unfulfilled demand can be fulfilled by nearby drivers by adding 10 minutes of passenger wait, and 75% of unfulfilled demand with 15 minutes

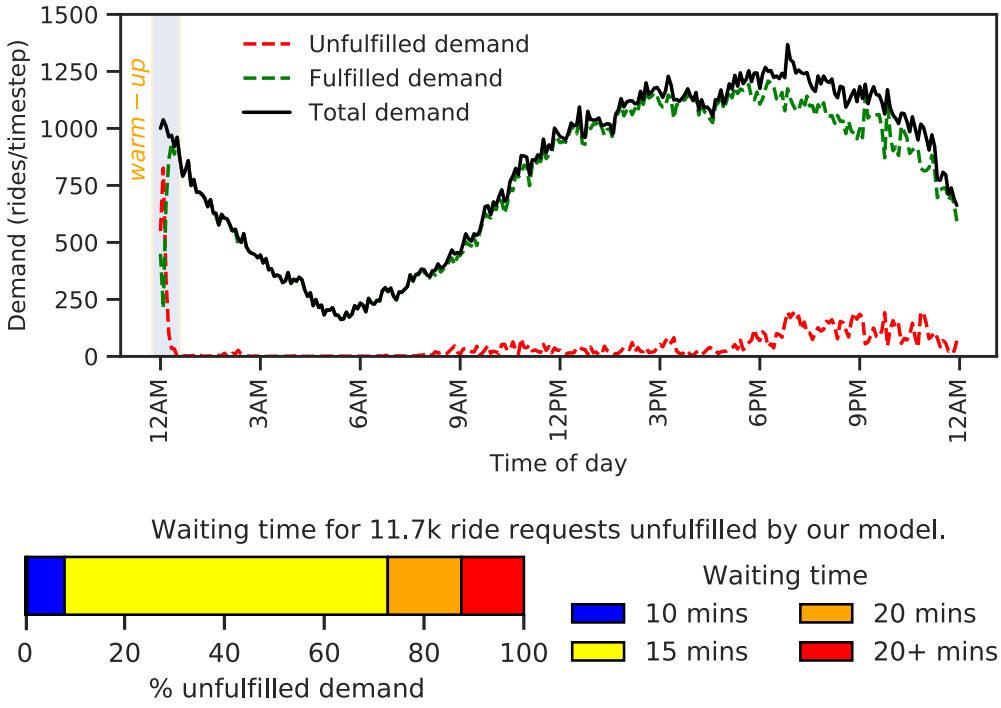

Figure 5·2: Top: Demand fulfillment by a trained policy at different times on a representative day; x-axis: time of the day, y-axis: number of ride requests. Bottom: Bar-plot for waiting times for demand not immediately fulfilled by the model.

of wait. At the beginning of a day, for lack of better alternative, we initialize drivers uniformly across the city zones. Hence, our model requires a "warm-up" time for the drivers to reposition themselves in order to fulfill the demand. This warm-up interval contributes significantly to the unfulfilled demand at the beginning of the day from 12AM-1AM. One may left-pad the training interval to alleviate this issue.

The explicit coordination in our model allows us to visualize the market conditions in which it is utilized. In Figure 5·3, we plot snapshots of coordination in form of a heatmap with probabilities of coordinated wait actions i.e. $Q_C(t, h, h)$ at 6 AM during the early morning commute and at 6 PM during the evening commute[1]. Without

[1]More detailed visualizations depicting evolution of coordinated actions and degree of coordination across the city and through the time of the day are available at [34].

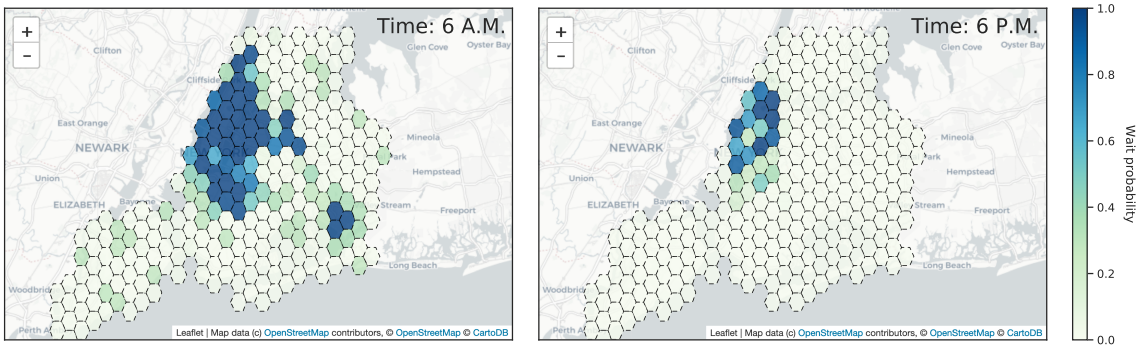

Figure 5·3: Heatmaps of probability of coordinated wait i.e., $Q_C(t, h, h)$ during morning commute at 6 A.M. (left) and evening commute at 6 P.M. (right).

coordination, we would expect all the drivers in the city to relocate to Manhattan in order to satisfy the extremely high volume of demand during the morning commute. However, as observed in Figure 5·3, our model recommends a certain proportion of drivers to wait in the outer boroughs of New York City for the early morning commute to Manhattan. Notably, the model is able to learn demand trends in time-dependent hotspots such as the J.F.K. airport to the south-east of the city. In contrast, during the evening commute to outer boroughs, the model strongly recommends that the drivers wait inside Manhattan.

Impact of independent and coordinated learning

The overlap between the independent and the coordinated learning during training is a crucial aspect of our framework. In this section, we address the question: *how do we determine the appropriate number of independent learning and coordinated learning episodes during training?* Given a fixed number of training episodes E, we assume that our model trains the initial E_{IL} episodes with independent learning and the final E_{CL} episodes with coordinated learning. When $E_{IL} + E_{CL} \geq E$, we have $E_{IL} + E_{CL} - E$ episodes of overlap between independent and coordinated learning. In Figure 5·4, we use 200 episodes of training, and we vary the values of E_{IL} and E_{CL} in

the range $[20, 200]$ to achieve various overlaps[2]. We then plot the mean driver earnings for each learned policy. We show that for a large interval of values of E_{IL} and E_{CL}, our framework provides stable and high performance with up to \$535 mean earnings per day when $E_{IL} = 60$ and $E_{CL} = 160$, denoted by a green marker in the figure. This observation supports our claim that our framework is robust to imperfections in hyperparameter tuning. Note that we have used different values of E_{IL} and E_{CL} in Figure 5·1 in order to clearly portray the incremental impact of coordinated learning on mean driver earnings per day.

Impact of Driver Supply

We next answer the question: *what is an appropriate number of drivers to fulfill the ride demand?* To study this question, we vary the driver supply in the range $[1000, 6000]$, where the units are individual drivers. Given a fixed supply size, we plot the ratio of the number of successful driver waits resulting into passenger rides to the number of unsuccessful driver waits while taking into account the overall demand fulfillment. When the number of drivers is small compared to the demand, the drivers should have an easier time finding a passenger. On the other hand, a city saturated with drivers should result in a higher number of unsuccessful driver waits. In Figure 5·5, we observe that the framework validates our expectations. The "warm-up" period described in Figure 5·2 causes underestimation of demand fulfillment while it simultaneously causes overestimation of the number of unsuccessful driver waits. Excluding the warm-up interval, this experiment provides evidence that over 96% demand of New York City can be fulfilled by about 5,000 drivers. Note that in September 2015, New York City had over 13,500 operational taxicab medallions [93]. It also justifies our decision to use 5,000 drivers in most of our experiments.

[2]Note that there is no overlap between the independent and the coordinated learning phases in the lower triangle of Figure 5·4 when $E_{IL} + E_{CL} < E$.

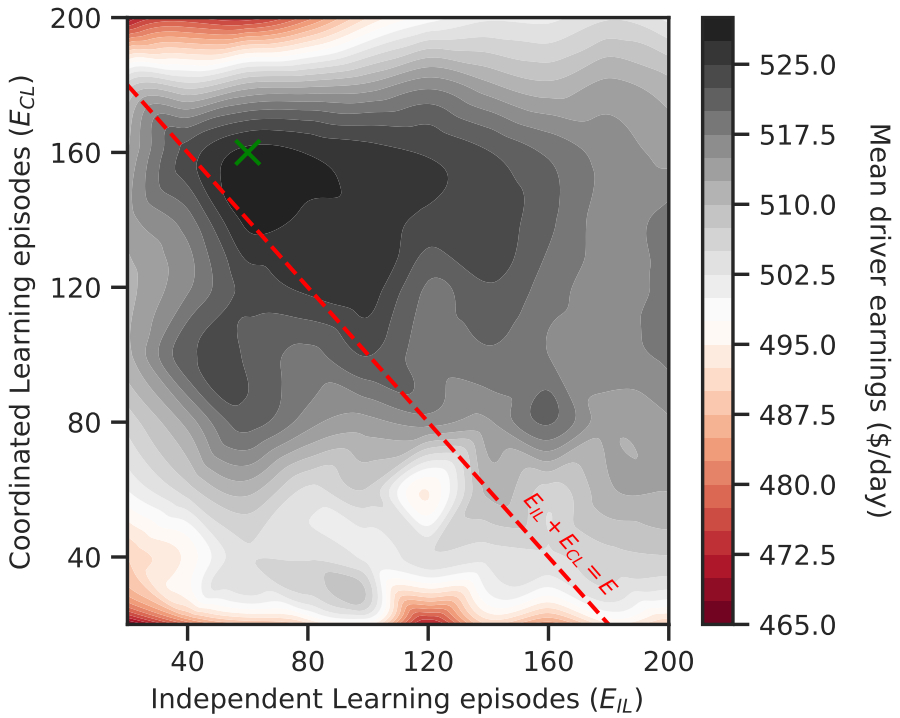

Figure 5·4: Performance stability over a wide range of overlaps between the independent and the coordinated learning; x-axis: number of independent learning episodes, y-axis: number of coordinated learning episodes. The color of the contour indicates average earnings of simulated drivers over an entire day, dotted red line refers to the main diagonal where $E_{IL} + E_{CL} = E$, points above the main diagonal refer to training schemes where there is an overlap between independent and coordinated learning.

Impact of platform objectives

So far, our experiments focused on the MaxEarnings problem. A natural question is: *should a platform optimize driver dispatches to maximize their earnings or to maximize demand fulfillment?* Note that while maximizing the demand fulfillment might help retain customers over a longer-term, it can be detrimental to drivers' earnings. To solve MaxFulfillment (see Section 5.1), the framework rewards (resp. penalizes) a successful passenger pickup (resp. unsuccessful wait) by +1 (resp. -1) net reward. Figure 5·6 depicts that mean driver earnings per passenger ride can be over

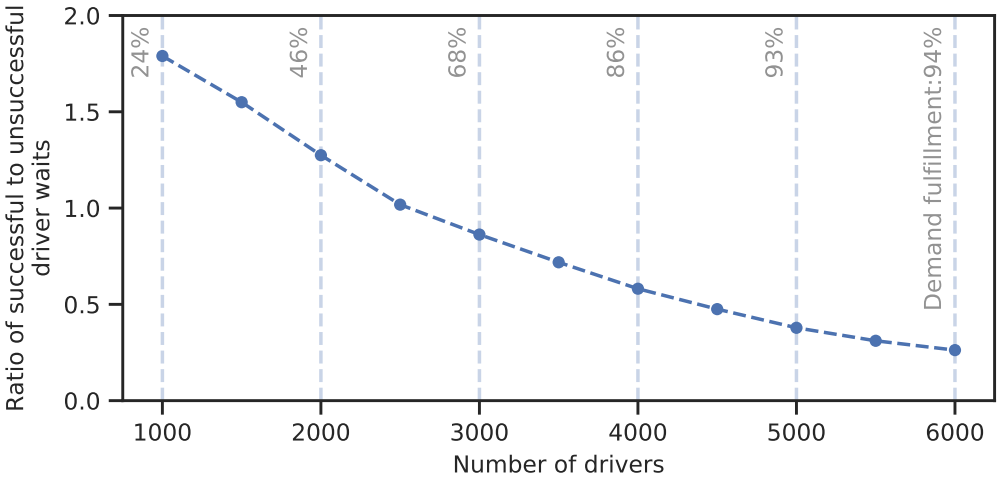

Figure 5·5: Impact of supply size on the ease of finding a passenger on a representative day; x-axis: number of drivers in the fleet, y-axis: ratio of successful wait actions to unsuccessful ones, vertical dotted lines indiciate the total demand fulfilled by the fleet of corresponding size.

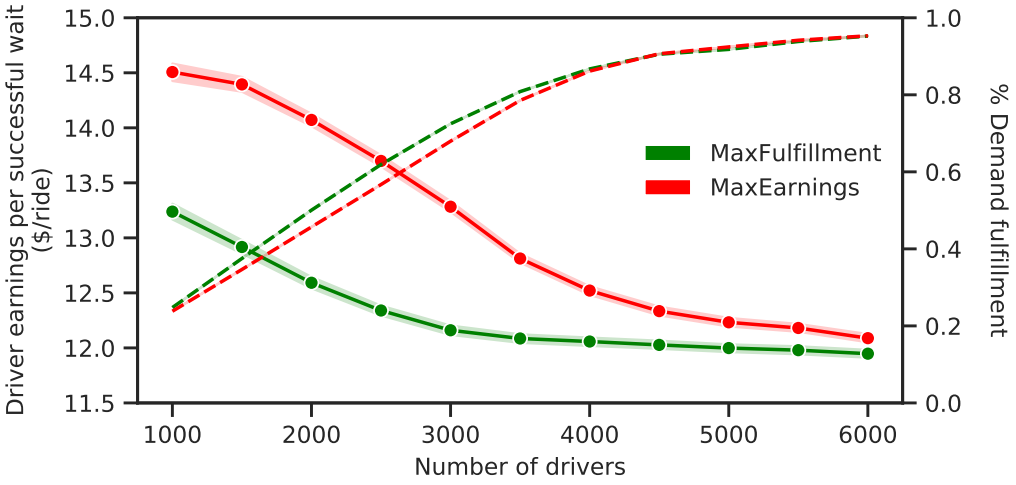

Figure 5·6: Differential impact of platform objectives; x-axis: number of drivers in the fleet, y-axis($l$eft): average driver earnings per passenger ride with 90% confidence interval (solid red and green lines), y-axis($r$ight): % demand fulfillment (dotted red and green lines) for both objectives on a representative day.

a $1 lower in a policy optimized for maximizing demand fulfillment relative to one optimized for earnings. The additional rides covered by the solution to MaxFulfillment may direct drivers to sub-optimal locations and compromise their future earnings for the day. As the supply increases over the minimum number of required drivers, the two objectives converge while a statistically significant difference in the driver earnings per ride persists. Note that higher rewards/penalties while solving MaxFulfillment result in larger divergence between the two objectives.

Advantage of strategic behavior

Next, we address the question: *does our model provide consistently higher earnings for all the drivers?* To explore this, we model the taxi driver population of the city as comprised of strategic drivers who follow the model recommendations and naive drivers who act upon heuristics learned via experience. We expect the mean earnings of drivers to decrease as the number of strategic drivers on the platform increases. While modeling a naive driver, we assume that taxi drivers, over time, learn the popular spots in the city. If they are unable to locate passengers reasonably quickly in other parts of the city, they head back to the popular spots. We designate 15 zones as popular zones based on the historical demand data. Furthermore, we assume that an idle naive driver looking for a passenger decides to head back to one of the popular zones with a fixed probability of 0.25. Upon choosing to relocate, the naive driver picks the target popular zone with a probability inversely proportional to its distance from the current location.

In Figure 5·7, we plot the earnings of the two categories of drivers while varying the percentage of strategic drivers. The lower and upper edges of the boxes indicate quartiles Q1 and Q3 respectively, and the length of whisker is 1.5 times IQR. As expected, an increase in the number of strategic drivers causes their individual earnings to decline. Overall, the strategic drivers not only earn more than the naive drivers,

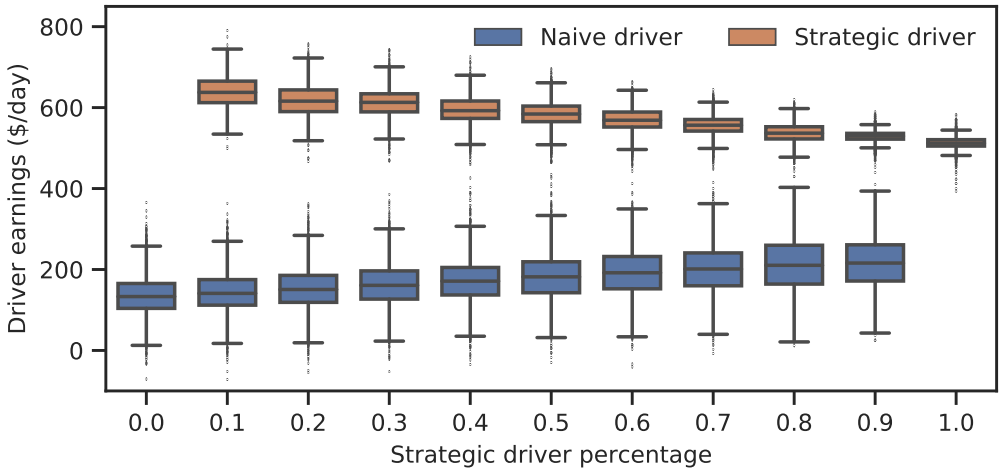

Figure 5·7: Earnings advantage of the strategic drivers over the naive drivers on a representative day; x-axis: % of fleet comprised of strategic drivers, y-axis: earnings of simulated drivers over an entire day.

but also the variance in their earnings is significantly lower. Thus, our framework is *envy-free*, i.e., drivers at the same location and time do not envy each other's future earnings.

Model generalizability

In this section, we explore the question of model generalizability: *does our model perform well when deployed on days with considerably different supply-demand conditions compared to the day it was trained on?* We cross-validate our model by evaluating the policy of a trained model on different days.

For illustrative purposes, we choose as baseline – m_0 – a model trained to satisfy the demand of 288,000 rides observed on the fourth Tuesday of September using 7,000 drivers. We test the policy $\pi(m_0)$ recommended by our baseline model by deploying it on other Tuesdays of the month. Note that the observed demand, as well as the number of active drivers might vary on other Tuesdays compared to our baseline model. To capture this potential for change in supply, we vary the number

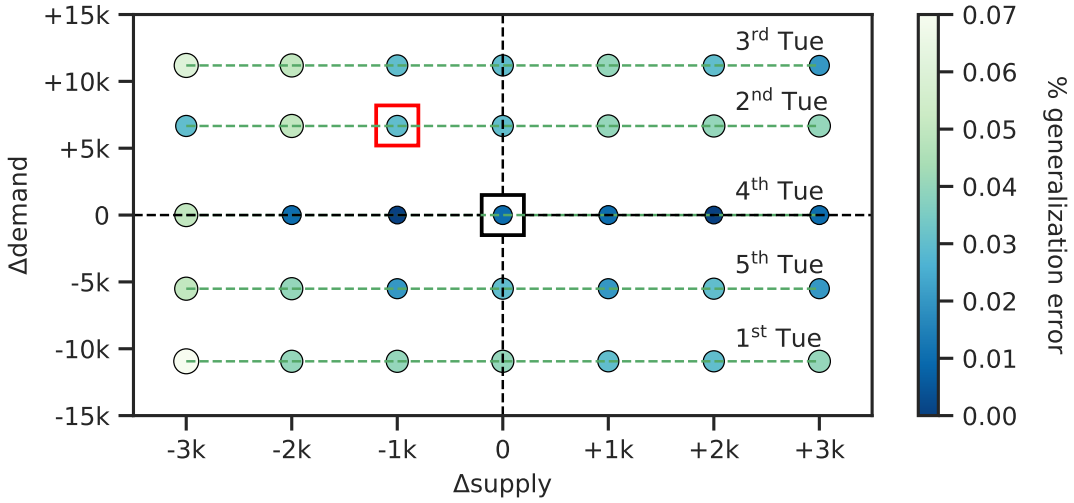

Figure 5·8: Model robustness: Baseline model (enclosed within a black box at origin) is deployed on other Tuesdays; x-axis: difference in fleet size between training and testing datasets, y-axis: difference in number of ride requests between training and testing datasets.

of simulated drivers during testing in the range $[4000, 10000]$. In Figure 5·8, enclosed within a red square box is an illustration of the generalization error associated with deploying our baseline model's recommended policy on the second Tuesday of the month with just 6,000 drivers. Importantly, the policy $\pi(m_0)$ now attempts to fulfill an increased demand of about 7,000 extra rides ($\Delta demand$) using 1,000 fewer drivers ($\Delta supply$) than it was trained for. To evaluate its performance in this task, we compare it with a model m_* which was explicitly trained to fulfill the demand of the second Tuesday with exactly 6000 drivers. Thus, we compute the baseline policy's generalization error as

$$\%\text{generalization error} = \frac{\mathcal{F}^\pi(m_*) - \mathcal{F}^\pi(m_0)}{\mathcal{F}^\pi(m_*)},$$

where $\mathcal{F}^\pi(m)$ denotes the demand fulfillment of model m.

Figure 5·8 shows that our framework generalizes well to perturbations in both supply and demand. We also observe that decreasing the number of drivers excessively

impacts harms its generalization performance. As a result, we recommend deploying models trained with a reasonably higher number of drivers than minimally required so that they generalize better in cases of varying demand. For brevity, we have presented a single illustrative example here; the generalizability result holds true across all the models.

Comparison with baselines

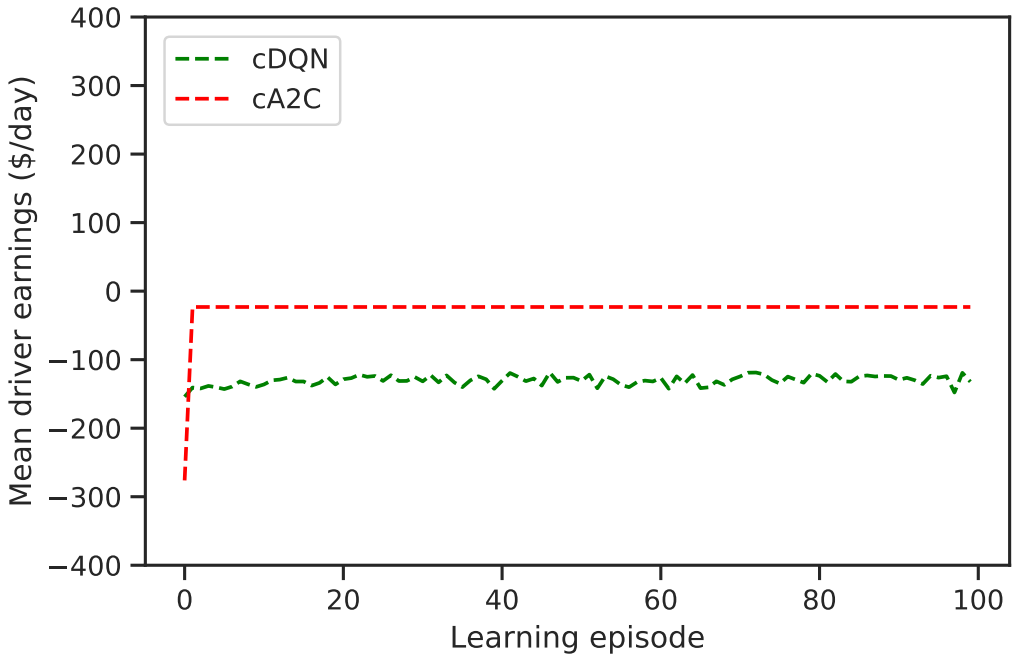

Figure 5·9: Performance of cDQN and cA2C deep-learning approaches; x-axis: training episode, y-axis: average earnings of the simulated drivers over an entire day.

A major challenge in comparative studies in this domain is the lack of reproducibility due to proprietory datasets and simulators. To the best of our knowledge, although Lin *et al.* [57] uses coordinated deep reinforcement learning approach, it is most similar to ours with respect to modeling assumptions. In the absence of the Didi Chuxing's proprietary driver simulator and datasets, direct comparison of our works

is impossible. We make an effort to compare our approaches by re-implementing their deep reinforcement learning based algorithms (cDQN and cA2C) with minimal modifications to fit our setting which computes future driver distributions based on simulating passenger pickups and dropoffs, instead of predicting them using proprietary models.

In [57], the authors do not train the neural network from its randomly initialized state. Instead, they bootstrap the network based on a *pre-trained value networks based on historical means* from the aforementioned simulator. As a direct and fair comparison with our model which does not rely on external pre-trained inputs, our implementations of their algorithms also attempt to learn *from scratch*.

Figure 5·9 shows mean driver earnings per day over the course of model training. Even after extensive hyperparameter tuning, the baselines failed to learn meaningful strategies, with driver earning net negative rewards of -$20 over a day. In the absence of a pre-trained value network, the algorithms proposed in [57] are unable to explore the policy space effectively. Moreover, the reward sharing assumption used in [57] results in a superficial coordination behavior which fails to learn in a more realistic scenario such as ours, which simulates actual passenger pickups and dropoffs. Our implementations of contextual DQN (cDQN) and contextual actor-critic (cA2C) algorithms are publicly available at [34].

Chapter Summary

In this chapter, we considered the problem of capacity repositioning on a ride-hailing platform to maximize welfare. We proposed a robust, explainable, and scalable framework that combines simple combinatorial techniques with traditional tabular Q-learning based RL algorithms. We perform a thorough experimental evaluation of the dynamics of the fleet-management system, its effectiveness, robustness to imperfect hyperparameter tunings, and generalizability in the presence of extraneous

perturbations to the input data. In course of this work, we also developed an OpenAI gym environment[3] named `nyc-yellow-taxi-v0` available at [34] so that the efficacy of any future multiagent reinforcement learning algorithm can be easily tested upon this problem.

[3]OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms [9].

Chapter 6

Conclusion

In recent years, Mobility-on-Demand systems have radically transformed the paradigm of urban mobility. Successful operations of MoD systems rely on solving a wide variety of problems from across multiple disciplines of sciences such as Computer Science, Statistics, Economics, Operations Research, etc. In this dissertation, we focus on few such problems of theoretical and practical interest.

In the first part of the dissertation, we introduce the problem of Markov Chain Monitoring: given a distribution of items over a Markov chain, we aim to perform a limited number of monitoring operations to predict the position of items on the chain after one transition step with least uncertainty. We study variants of this problem and provide efficient algorithms to solve them. Our experimental evaluation demonstrates the superiority of the proposed algorithms compared to baselines and the practical utility of our algorithms in a fleet monitoring operation for a bike-sharing MoD.

A natural extension of this work is to select monitoring operations under incomplete information for initial item distribution – which would allow the operations to be deployed in perpetuity. Another future-work direction is to consider different types of monitoring operations, such as those that combine knowledge of item placement on nodes and edges. Finally, we can also consider more complex traffic models (e.g., involving queuing and different transition delays between nodes [27]).

Next, we shift our attention to ride-hailing services and focus on the problem of maximizing a driver's individual earnings on ride-hailing platforms. Our work con-

firms the power of strategic driving behavior using data-driven projections of ridership in the New York City area. Our first key takeaway is that a naive driver, armed with no data, and driving a 9-5 random walk schedule, is leaving roughly a 50% pay raise on the table by not driving more strategically. In contrast, a data-savvy driver armed with good historical data can build a forecast and optimal contingency driving plans with relatively little computational overhead using our dynamic programming algorithms that have provable resilience to input uncertainty. Our experimental results yield insights into the structure of highly optimized schedules, including relatively frequent relocation, working at specific peak periods, and taking advantage of surges when the time is ripe.

An obvious limitation of this approach is that it is tailored to the setting when self-interested individual drivers employ the methods. If a significant percentage of the labor supply employs sophisticated optimization methods for driving, one needs to consider different strategies to achieve equilibria or other global objectives. Indeed, in the long run, as drivers for ride-hailing platforms like Uber and Lyft are put out of work by fleets of autonomous vehicles, the formulation and solution of new sets of optimization problems along those lines are likely to become relevant as well.

In the final part of this dissertation, to alleviate the limitation of the approach above, we study the problem of maximizing the earnings of an entire fleet of drivers employed by ride-hailing platforms. Our work confirms that even in a high-dimensional and big data domain such as ride-sharing, the inherent structure of the data can be leveraged to develop a simple, interpretable, fair, and highly efficient framework to achieve this goal. Extensive simulations based on New York City taxi datasets show that our framework is easy to calibrate due to its robustness to imperfections in hyperparameter tuning. Our experiments provide evidence for the differential impact of the platform's objectives on driver earnings. Finally, we demonstrate that our model

generalizes well to fluctuations in supply and demand. We also make available an OpenAI gym environment for comparative studies.

As a final remark, as automated electric vehicle technology continues to advance, we anticipate facing MoD operations problems related to the impact of autonomous electric vehicles in the future. In such settings, the optimal strategies to match vehicles to ride requests would have to consider additional factors such as the state of the power grid, cost of electricity, and the inherent slow battery charging processes. Moreover, it is unlikely that the entire human workforce would be replaced by automated vehicles instantly. In light of this, we believe that explainable algorithms and fair compensation strategies for human drivers alongside the automated fleets of MoDs will be exceedingly important. Based on the works in this dissertation, we believe that the path of utilizing a diverse tool-set consisting of classic techniques like approximation algorithms, combinatorial optimization methods, etc., alongside modern machine learning-based approaches such as reinforcement learning shows promise in being able to tackle the multi-faceted problem of optimizing Mobility-on-Demand systems.

References

[1] William A. Bailey and Thomas D. Clark. A Simulation Analysis of Demand and Fleet Size Effects on Taxicab Service Rates. In *Proceedings of the 19th Conference on Winter Simulation*, WSC '87, pages 838–844, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0911801324. doi: 10.1145/318371.318705. URL https://doi.org/10.1145/318371.318705.

[2] Siddhartha Banerjee, Ramesh Johari, and Carlos Riquelme. Pricing in Ride-Sharing Platforms: A Queueing-Theoretic Approach. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, EC '15, page 639, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334105. doi: 10.1145/2764468.2764527. URL https://doi.org/10.1145/2764468.2764527.

[3] Mike Benchimol, Pascal Benchimol, Benot Chappert, Arnaud de la Taille, Fabien Laroche, Frdric Meunier, and Ludovic Robinet. Balancing the stations of a self service "bike hire" system. *RAIRO: Recherche Oprationnelle*, 45(1):37–61, 01 2011. doi: 10.1051/ro/2011102. URL https://doi.org/10.1051/ro/2011102.

[4] Omar Besbes, Francisco Castro, and Ilan Lobel. Surge pricing and its spatial supply response. *Management Science*, 67(3):1350–1367, 2021. doi: 10.1287/mnsc.2020.3622. URL https://doi.org/10.1287/mnsc.2020.3622.

[5] Kostas Bimpikis, Ozan Candogan, and Daniela Saban. Spatial pricing in ride-sharing networks. *Operations Research*, 67(3):744–769, 2019. doi: 10.1287/opre.2018.1800. URL https://doi.org/10.1287/opre.2018.1800.

[6] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing Social Influence in Nearly Optimal Time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 946–957, USA, 2014. Society for Industrial and Applied Mathematics. ISBN 9781611973389. doi: 10.5555/2634074.2634144. URL https://doi.org/10.5555/2634074.2634144.

[7] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001. doi: 10.1080/0022250X.2001.9990249. URL https://doi.org/10.1080/0022250X.2001.9990249.

[8] Ulrik Brandes and Daniel Fleischer. Centrality Measures Based on Current Flow. In *Proceedings of the 22nd Annual Conference on Theoretical Aspects of Com-*

puter Science, STACS'05, pages 533–544, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540249982. doi: 10.1007/978-3-540-31856-9_44. URL https://doi.org/10.1007/978-3-540-31856-9_44.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. URL https://arxiv.org/abs/1606.01540.

[10] Teobaldo Bulhes, Anand Subramanian, Güneks Erdoğan, and Gilbert Laporte. The static bike relocation problem with multiple vehicles and visits. *European Journal of Operational Research*, 264(2):508–523, 2018. ISSN 0377-2217. doi: 10.1016/j.ejor.2017.06.028. URL https://doi.org/10.1016/j.ejor.2017.06.028.

[11] Business Traveller. Global ride sharing industry valued at more than $61 Billion, January 2019. URL www.businesstraveller.com/business-travel/2019/01/04/value-of-global-ride-sharing-industry-estimated-at-more-than-61-billion/.

[12] Colin Camerer, Linda Babcock, George Loewenstein, and Richard Thaler. Labor Supply of New York City Cabdrivers: One day at a Time. *The Quarterly Journal of Economics*, 112(2):407–441, 1997. doi: 10.1162/003355397555244. URL https://doi.org/10.1162/003355397555244.

[13] Juan Camilo Castillo, Dan Knoepfle, and Glen Weyl. Surge Pricing Solves the Wild Goose Chase. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, EC '17, pages 241–242, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450345279. doi: 10.1145/3033274.3085098. URL https://doi.org/10.1145/3033274.3085098.

[14] Harshal A. Chaudhari, John W. Byers, and Evimaria Terzi. Online addendum: Putting Data in the Driver's Seat. https://www.bu.edu/cs/groups/dblab/ride-hailing, 2017.

[15] Daniel Chemla, Frédéric Meunier, Thomas Pradeau, Roberto Wolfler Calvo, and Houssame Yahiaoui. Self-service bike sharing systems: simulation, repositioning, pricing. 2013. URL https://hal.archives-ouvertes.fr/hal-00824078/document.

[16] H Chen, Y Jiao, Z Qin, X Tang, H Li, B An, H Zhu, and J Ye. 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)*, pages 61–70, 2019. doi: 10.1109/ICDM.2019.00016. URL https://doi.org/10.1109/ICDM.2019.00016.

[17] Haoyang Chen, Wei Wang, Kåre Kjelstrøm, and Emily Reinhold. Gaining Insights in a Simulated Marketplace with Machine Learning at Uber. https://eng.uber.com/simulated-marketplace/, June 2019.

[18] Le Chen, Alan Mislove, and Christo Wilson. Peeking Beneath the Hood of Uber. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, pages 495–508, New York, USA, 2015. Association for Computing Machinery. ISBN 9781450338486. doi: 10.1145/2815675.2815681. URL https://doi.org/10.1145/2815675.2815681.

[19] M. Keith Chen. Dynamic Pricing in a Labor Market: Surge Pricing and Flexible Work on the Uber Platform. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, page 455, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450339360. doi: 10.1145/2940716.2940798. URL https://doi.org/10.1145/2940716.2940798.

[20] Joseph Y.J. Chow and Hamid R. Sayarshad. Symbiotic network design strategies in the presence of coexisting transportation networks. *Transportation Research Part B: Methodological*, 62:13–34, 2014. ISSN 0191-2615. doi: 10.1016/j.trb.2014.01.008. URL https://doi.org/10.1016/j.trb.2014.01.008.

[21] Mauro Dell'Amico, Eleni Hadjicostantinou, Manuel Iori, and Stefano Novellani. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega*, 45:7–19, 2014. ISSN 0305-0483. doi: 10.1016/j.omega.2013.12.001. URL https://doi.org/10.1016/j.omega.2013.12.001.

[22] Dóra Erdös, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. A Divide-and-Conquer Algorithm for Betweenness Centrality. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 433–441, 2015. doi: 10.1137/1.9781611974010.49. URL https://doi.org/10.1137/1.9781611974010.49.

[23] Gne Erdoğan, Gilbert Laporte, and Roberto Wolfler Calvo. The static bicycle relocation problem with demand intervals. *European Journal of Operational Research*, 238(2):451–457, 2014. ISSN 0377-2217. doi: 10.1016/j.ejor.2014.04.013. URL https://doi.org/10.1016/j.ejor.2014.04.013.

[24] Iris A. Forma, Tal Raviv, and Michal Tzur. A 3-step math heuristic for the static repositioning problem in bike-sharing systems. *Transportation Research Part B: Methodological*, 71:230–247, 2015. ISSN 0191-2615. doi: 10.1016/j.trb.2014.10.003. URL https://doi.org/10.1016/j.trb.2014.10.003.

[25] Christine Fricker and Nicolas Gast. Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity. *EURO Journal on Transportation and Logistics*, 5(3):261–291, 2016. ISSN 2192-4376. doi: 10.1007/s13676-014-0053-5. URL https://doi.org/10.1007/s13676-014-0053-5.

[26] Sainyam Galhotra, Akhil Arora, Srinivas Virinchi, and Shourya Roy. ASIM: A Scalable Algorithm for Influence Maximization under the Independent Cascade Model. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 35–36, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334730. doi: 10.1145/2740908.2742725. URL https://doi.org/10.1145/2740908.2742725.

[27] Robert G Gallager. *Discrete Stochastic Processes*, volume 321. Springer Science & Business Media, 2012. doi: 10.1007/978-1-4615-2329-1. URL https://doi.org/10.1007/978-1-4615-2329-1.

[28] Yong Gao, Dan Jiang, and Yan Xu. Optimize taxi driving strategies based on reinforcement learning. *International Journal of Geographical Information Science*, 32(8):1677–1696, 2018. doi: 10.1080/13658816.2018.1458984. URL https://doi.org/10.1080/13658816.2018.1458984.

[29] Nandani Garg and Sayan Ranu. Route Recommendations for Idle Taxi Drivers: Find Me the Shortest Route to a Customer! In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1425–1434, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220055. URL https://doi.org/10.1145/3219819.3220055.

[30] Nikhil Garg and Hamid Nazerzadeh. Driver Surge Pricing. In *Proceedings of the 21st ACM Conference on Economics and Computation*, EC '20, page 501, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379755. doi: 10.1145/3391403.3399476. URL https://doi.org/10.1145/3391403.3399476.

[31] David K. George and Cathy H. Xia. Fleet-sizing and service availability for a vehicle rental system via closed queueing networks. *European Journal of Operational Research*, 211(1):198–207, 2011. ISSN 0377-2217. doi: 10.1016/j.ejor.2010.12.015. URL https://doi.org/10.1016/j.ejor.2010.12.015.

[32] Aristides Gionis, Evimaria Terzi, and Panayiotis Tsaparas. Opinion Maximization in Social Networks. In *Proceeding of the 2013 SIAM International Conference on Data Mining (SDM)*, pages 387–395, 2013. doi: 10.1137/1.9781611972832.43. URL https://doi.org/10.1137/1.9781611972832.43.

[33] Aristides Gionis, Michael Mathioudakis, and Antti Ukkonen. Bump Hunting in the Dark: Local Discrepancy Maximization on Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29(3):529–542, 2017. doi: 10.1109/TKDE.2016.2571693. URL https://doi.org/10.1109/TKDE.2016.2571693.

[34] GitHub Repository. Learn to Earn. https://chdhr-harshal.github.io/learn-to-earn/, 2020.

[35] Amit Goyal, Wei Lu, and Laks V.S. Lakshmanan. SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model. In *2011 IEEE 11th International Conference on Data Mining*, pages 211–220, 2011. doi: 10.1109/ICDM.2011.132. URL https://doi.org/10.1109/ICDM.2011.132.

[36] The Rideshare Guy. Advice For New Uber Drivers- Don't Chase The Surge! http://maximumridesharingprofits.com/advice-new-uber-drivers-dont-chase-surge/, 2016.

[37] Jonathan V. Hall and Alan B. Krueger. An Analysis of the Labor Market for Uber's Driver-Partners in the United States. Technical Report No. w22843, National Bureau of Economic Research, 2016. URL https://doi.org/10.1177/0019793917717222.

[38] Waster Hudson. Chasing the Surge: 3 Tips for Maximizing Uber Earnings, 2016. URL https://www.pjmedia.com/lifestyle/2016/07/19/chasing-the-surge-3-tips-for-maximizing-uber-earnings-n164052.

[39] Mimi Hwang, James Kemp, Eva Lerner-Lam, Nancy Neuerburg, Paula E Okunieff, et al. Advanced Public Transportation Systems: the State of the Art. *United States. Department of Transportation. Federal Transit Administration*, 2006. URL https://rosap.ntl.bts.gov/view/dot/16448.

[40] Vatche Ishakian, Dóra Erdös, Evimaria Terzi, and Azer Bestavros. A Framework for the Evaluation and Management of Network Centrality. In *Proceedings of the 2012 SIAM International Conference on Data Mining (SDM)*, pages 427–438, 2012. doi: 10.1137/1.9781611972825.37. URL https://doi.org/10.1137/1.9781611972825.37.

[41] Jiarui Jin, Ming Zhou, Weinan Zhang, Minne Li, Zilong Guo, Zhiwei Qin, Yan Jiao, Xiaocheng Tang, Chenxi Wang, Jun Wang, Guobin Wu, and Jieping Ye. CoRide: Joint Order Dispatching and Fleet Management for Multi-Scale Ride-Hailing Platforms. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, pages 1983–1992, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369763. doi: 10.1145/3357384.3357978. URL https://doi.org/10.1145/3357384.3357978.

[42] Jaeyoung Jung, R. Jayakrishnan, and Ji Young Park. Design and Modeling of Real-time Shared-taxi Dispatch Algorithms. In *Proceedings of Transportation Research Board 92nd Annual Meeting*, 2013. URL https://trid.trb.org/view/1241180.

[43] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the Spread of Influence through a Social Network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. Association for Computing Machinery. ISBN

1581137370. doi: 10.1145/956750.956769. URL https://doi.org/10.1145/956750.956769.

[44] Christian Kloimüllner and Günther R. Raidl. Full-Load Route Planning for Balancing Bike Sharing Systems by Logic-Based Benders Decomposition. *Networks*, 69(3):270–289, May 2017. ISSN 0028-3045. doi: 10.1002/net.21736. URL https://doi.org/10.1002/net.21736.

[45] S. Kullback, M. Kupperman, and H. H. Ku. Tests for Contingency Tables and Markov Chains. *Technometrics*, 4(4):573–608, 1962. doi: 10.1080/00401706.1962.10490041. URL https://doi.org/10.1080/00401706.1962.10490041.

[46] Prem Kumar and Michel Bierlaire. Optimizing locations for a vehicle sharing system. In *Swiss Transport Research Conference*, 2012. URL https://core.ac.uk/download/pdf/148001815.pdf.

[47] Gilbert Laporte, Frédéric Meunier, and Roberto Wolfler Calvo. Shared mobility systems: an updated survey. *Annals of Operations Research*, 271(1):105–126, 2018. doi: 10.1007/s10479-018-3076-8. URL https://doi.org/10.1007/s10479-018-3076-8.

[48] Der-Horng Lee, Hao Wang, Ruey Long Cheu, and Siew Hoon Teo. Taxi Dispatch System Based on Current Demands and Real-Time Traffic Conditions. *Transportation Research Record*, 1882(1):193–200, January 2004. ISSN 0361-1981. doi: 10.3141/1882-23. URL https://doi.org/10.3141/1882-23.

[49] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014. URL http://snap.stanford.edu/data.

[50] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-Effective Outbreak Detection in Networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 420–429, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936097. doi: 10.1145/1281192.1281239. URL https://doi.org/10.1145/1281192.1281239.

[51] Bin Li, Daqing Zhang, Lin Sun, Chao Chen, Shijian Li, Guande Qi, and Qiang Yang. Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset. In *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 63–68, 2011. doi: 10.1109/PERCOMW.2011.5766967. URL https://doi.org/10.1109/PERCOMW.2011.5766967.

[52] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient Ridesharing Order Dispatching with Mean Field

Multi-Agent Reinforcement Learning. In *Proceedings of the 28th International Conference on World Wide Web*, WWW '19, pages 983–994, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313433. URL https://doi.org/10.1145/3308558.3313433.

[53] Xiaopeng Li, Jiaqi Ma, Jianxun Cui, Amir Ghiasi, and Fang Zhou. Design framework of large-scale one-way electric vehicle sharing systems: A continuum approximation model. *Transportation Research Part B: Methodological*, 88:21–45, 2016. ISSN 0191-2615. doi: 10.1016/j.trb.2016.01.014. URL https://doi.org/10.1016/j.trb.2016.01.014.

[54] Yanfeng Li, W.Y. Szeto, Jiancheng Long, and C.S. Shui. A multiple type bike repositioning problem. *Transportation Research Part B: Methodological*, 90:263–278, 2016. ISSN 0191-2615. doi: 10.1016/j.trb.2016.05.010. URL https://doi.org/10.1016/j.trb.2016.05.010.

[55] Jenn-Rong Lin and Ta-Hui Yang. Strategic design of public bicycle sharing systems with service level constraints. *Transportation research part E: Logistics and Transportation review*, 47(2):284–294, 2011. doi: 10.1016/j.tre.2010.09.004. URL https://doi.org/10.1016/j.tre.2010.09.004.

[56] Jenn-Rong Lin, Ta-Hui Yang, and Yu-Chung Chang. A hub location inventory model for bicycle sharing system design: Formulation and solution. *Computers & Industrial Engineering*, 65(1):77–86, 2013. doi: /10.1016/j.cie.2011.12.006. URL https://doi.org/10.1016/j.cie.2011.12.006.

[57] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1774–1783, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219993. URL https://doi.org/10.1145/3219819.3219993.

[58] Hongyao Ma, Fei Fang, and David C. Parkes. Spatio-Temporal Pricing for Ridesharing Platforms. *SIGecom Exchange*, 18(2):53–57, December 2020. doi: 10.1145/3440968.3440975. URL https://doi.org/10.1145/3440968.3440975.

[59] Michal Maciejewski and Kai Nagel. Simulation and dynamic optimization of taxi services in MATSim. *Transportation Science*, 2013.

[60] Luis M. Martinez, Lus Caetano, Toms Eir, and Francisco Cruz. An Optimisation Algorithm to Establish the Location of Stations of a Mixed Fleet Biking System: An Application to the City of Lisbon. In *Proceedings of EWGT2012 - 15th Meeting of the EURO Working Group on Transportation, September 2012, Paris*, volume 54,

pages 513–524, 2012. doi: 10.1016/j.sbspro.2012.09.769. URL https://doi.org/10.1016/j.sbspro.2012.09.769.

[61] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of Influence Networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 529–537, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450308137. doi: 10.1145/2020408.2020492. URL https://doi.org/10.1145/2020408.2020492.

[62] Charalampos Mavroforakis, Michael Mathioudakis, and Aristides Gionis. Absorbing random-walk centrality: Theory and algorithms. In *2015 IEEE International Conference on Data Mining*, pages 901–906. IEEE, 2015. doi: 10.1109/ICDM.2015.103. URL https://doi.org/10.1109/ICDM.2015.103.

[63] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. December 2013. URL https://arxiv.org/abs/1312.5602v1.

[64] Rahul Nair and Elise Miller-Hooks. Equilibrium network design of shared-vehicle systems. *European Journal of Operational Research*, 235(1):47–61, 2014. ISSN 0377-2217. doi: 10.1016/j.ejor.2013.09.019. URL https://doi.org/10.1016/j.ejor.2013.09.019.

[65] Arnab Nilim and Laurent El Ghaoui. Robustness in Markov Decision Problems with Uncertain Transition Matrices. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, pages 839–846, Cambridge, MA, USA, 2003. MIT Press. doi: 10.5555/2981345.2981450. URL https://doi.org/10.5555/2981345.2981450.

[66] Jorge Nunes, Luís Matos, and António Trigo. Taxi pick-ups route optimization using genetic algorithms. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 410–419. Springer, 2011. doi: 10.1007/978-3-642-20282-7_42. URL https://doi.org/10.1007/978-3-642-20282-7_42.

[67] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-Ichi Kawarabayashi. Fast and Accurate Influence Maximization on Large Networks with Pruned Monte-Carlo Simulations. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 138–144. AAAI Press, 2014. doi: 10.5555/2893873.2893897. URL https://doi.org/10.5555/2893873.2893897.

[68] Erhun Özkan and Amy R Ward. Dynamic matching for real-time ride sharing. *Stochastic Systems*, 10(1):29–70, 2020. doi: 10.1287/stsy.2019.0037. URL https://doi.org/10.1287/stsy.2019.0037.

[69] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999. URL http://ilpubs.stanford.edu:8090/422/. Previous number = SIDL-WP-1999-0120.

[70] Julius Pfrommer, Joseph Warrington, Georg Schildbach, and Manfred Morari. Dynamic Vehicle Redistribution and Online Price Incentives in Shared Mobility Systems. *IEEE Transactions on Intelligent Transportation Systems*, 15(4):1567–1578, 2014. doi: 10.1109/TITS.2014.2303986. URL https://doi.org/10.1109/TITS.2014.2303986.

[71] Zhiwei Qin, Hongtu Zhu, and Jieping Ye. Reinforcement Learning for Ridesharing: A Survey. 2021. URL https://arxiv.org/pdf/2105.01099.pdf.

[72] Tal Raviv and Ofer Kolka. Optimal inventory management of a bike-sharing station. *IIE Transactions*, 45(10):1077–1093, 2013. doi: 10.1080/0740817X.2013.770186. URL https://doi.org/10.1080/0740817X.2013.770186.

[73] Matteo Riondato and Evgenios M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery*, 30 (2):438–475, 2016. doi: 10.1007/s10618-015-0423-0. URL https://doi.org/10.1007/s10618-015-0423-0.

[74] Huigui Rong, Xun Zhou, Chang Yang, Zubair Shafiq, and Alex Liu. The Rich and the Poor: A Markov Decision Process Approach to Optimizing Taxi Driver Revenue Efficiency. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 2329–2334, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340731. doi: 10.1145/2983323.2983689. URL https://doi.org/10.1145/2983323.2983689.

[75] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966. doi: 10.1007/BF02289527. URL https://doi.org/10.1007/BF02289527.

[76] J. Schuijbroek, R.C. Hampshire, and W.-J. van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, 257(3):992–1004, 2017. ISSN 0377-2217. doi: 10.1016/j.ejor.2016.08.029. URL https://doi.org/10.1016/j.ejor.2016.08.029.

[77] K T Seow, N H Dang, and D Lee. A Collaborative Multiagent Taxi-Dispatch System. *IEEE Transactions on Automation Science and Engineering*, 7(3):607–616, July 2010. ISSN 1545-5955. doi: 10.1109/TASE.2009.2028577. URL https://doi.org/10.1109/TASE.2009.2028577.

[78] Ying Shi and Zhaotong Lian. Optimization and strategic behavior in a passengertaxi service system. *European Journal of Operational Research*, 249(3):1024–1032, 2016. ISSN 0377-2217. doi: 10.1016/j.ejor.2015.07.031. URL https://doi.org/10.1016/j.ejor.2015.07.031.

[79] Tom Sühr, Asia J. Biega, Meike Zehlike, Krishna P. Gummadi, and Abhijnan Chakraborty. Two-Sided Fairness for Repeated Matchings in Two-Sided Markets: A Case Study of a Ride-Hailing Platform. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 3082–3092, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330793. URL https://doi.org/10.1145/3292500.3330793.

[80] Richard S Sutton and Andrew G Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 9780262193986. doi: 10.5555/3312046. URL https://doi.org/10.5555/3312046.

[81] Xiaocheng Tang, Zhiwei (Tony) Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. A Deep Value-Network Based Approach for Multi-Driver Order Dispatching. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1780–1790, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330724. URL https://doi.org/10.1145/3292500.3330724.

[82] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence Maximization: Near-Optimal Time Complexity Meets Practical Efficiency. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 75–86, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450323765. doi: 10.1145/2588555.2593670. URL https://doi.org/10.1145/2588555.2593670.

[83] The New York Times. An App That Helps Drivers Earn the Most From Their Trips. https://www.nytimes.com/2015/05/10/technology/a-dashboard-management-consultant.html, 2015.

[84] The New York Times. How Uber Uses Psychological Tricks to Push Its Drivers' Buttons, 2017. URL https://www.nytimes.com/interactive/2017/04/02/technology/uber-drivers-psychological-tricks.html.

[85] Tanvi Verma, Pradeep Varakantham, Sarit Kraus, and Hoong Chuin Lau. Augmenting Decisions of Taxi Drivers through Reinforcement Learning for Improving Revenues. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, pages 409–418. AAAI Press, 2017.

[86] James Vincent. AI systems should be accountable, explainable, and unbiased, says EU. *TheVerge*, April 2019. URL https://theverge.com/2019/4/8/18300149/eu-artificial-intelligence-ai-ethical-guidelines-recommendations.

[87] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2. URL https://doi.org/10.1038/s41592-019-0686-2.

[88] Patrick Vogel, Bruno A. Neumann Saavedra, and Dirk C. Mattfeld. A Hybrid Metaheuristic to Solve the Resource Allocation Problem in Bike Sharing Systems. In *Hybrid Metaheuristics*, pages 16–29, Cham, 2014. Springer International Publishing. ISBN 978-3-319-07644-7. doi: 10.1007/978-3-319-07644-7_2. URL https://doi.org/10.1007/978-3-319-07644-7_2.

[89] Z Wang, Z Qin, X Tang, J Ye, and H Zhu. Deep Reinforcement Learning with Knowledge Transfer for Online Rides Order Dispatching. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 617–626, November 2018. doi: 10.1109/ICDM.2018.00077. URL https://doi.org/10.1109/ICDM.2018.00077.

[90] Ariel Waserhole and Vincent Jost. Pricing in vehicle sharing systems: optimization in queuing networks with product forms. *EURO Journal on Transportation and Logistics*, 5(3):293–320, 2016. ISSN 2192-4376. doi: 10.1007/s13676-014-0054-4. URL https://doi.org/10.1007/s13676-014-0054-4.

[91] Jian Wen, Jinhua Zhao, and Patrick Jaillet. Rebalancing shared mobility-on-demand systems: A reinforcement learning approach. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems*, pages 220–225, 2017. doi: 10.1109/ITSC.2017.8317908. URL https://doi.org/10.1109/ITSC.2017.8317908.

[92] Wikipedia contributors. Skellam distribution. https://en.wikipedia.org/w/index.php?title=Skellam_distribution, 2017.

[93] Wikipedia contributors. Taxicabs of New York City. https://en.wikipedia.org/w/index.php?title=Taxicabs_of_New_York_City, 2019.

[94] Wikipedia contributors. Bicycle-sharing system. https://en.wikipedia.org/w/index.php?title=Bicycle-sharing_system, 2021.

[95] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data*

Mining, KDD '18, pages 905–913, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219824. URL https://doi.org/10.1145/3219819.3219824.

[96] Chiwei Yan, Helin Zhu, Nikita Korolko, and Dawn Woodard. Dynamic pricing and matching in ride-hailing platforms. *Naval Research Logistics (NRL)*, 67(8): 705–724, 2020. doi: 10.2139/ssrn.3258234. URL https://doi.org/10.2139/ssrn.3258234.

[97] Hai Yang, S.C. Wong, and K.I. Wong. Demandsupply equilibrium of taxi services in a network under competition and regulation. *Transportation Research Part B: Methodological*, 36(9):799–819, 2002. ISSN 0191-2615. doi: 10.1016/S0191-2615(01)00031-5. URL https://doi.org/10.1016/S0191-2615(01)00031-5.

[98] Jing Yuan, Yu Zheng, Liuhang Zhang, XIng Xie, and Guangzhong Sun. Where to Find My next Passenger. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 109–118, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306300. doi: 10.1145/2030112.2030128. URL https://doi.org/10.1145/2030112.2030128.

[99] N J Yuan, Y Zheng, L Zhang, and X Xie. T-Finder: A Recommender System for Finding Passengers and Vacant Taxis. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2390–2403, October 2013. ISSN 1041-4347. doi: 10.1109/TKDE.2012.153. URL https://doi.org/10.1109/TKDE.2012.153.

[100] Lingyu Zhang, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, and Jieping Ye. A Taxi Order Dispatch Model Based On Combinatorial Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '17, pages 2151–2159, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098138. URL https://doi.org/10.1145/3097983.3098138.

[101] Rick Zhang and Marco Pavone. Control of robotic mobility-on-demand systems: A queueing-theoretical perspective. *The International Journal of Robotics Research*, 35(1–3):186–203, 2016. doi: 10.1177/0278364915581863. URL https://doi.org/10.1177/0278364915581863.

[102] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. 2020. URL https://arxiv.org/abs/2010.09776.

Curriculum Vitae