TECHNIQUES FOR EFFICIENT AND RESPONSIBLE OPERATION
OF MOBILITY SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Matthew Wu Tsao
June 2022
© 2022 by Matthew Wu Tsao. All Rights Reserved.
Re-distributed by Stanford University under license with the author.

This work is licensed under a Creative Commons Attribution-3.0 United States License.
http://creativecommons.org/licenses/by/3.0/us/

This dissertation is online at: https://purl.stanford.edu/cq614xb8649
I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Marco Pavone, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Stephen Boyd

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dorsa Sadigh

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.
Abstract

Transportation is a necessary resource for many societies around the world. While advances in data science provide promising tools for personalized, adaptive and more efficient mobility services, they also bring new challenges in equal measure. In this dissertation I will discuss algorithm design for two such challenges faced by modern mobility services. First, I will discuss techniques for operating ridehailing and ridesharing systems in settings with incomplete information, which often arise due to the on-demand nature of such services. In particular, I will show both in theory and in practice how ideas from model predictive control, online optimization and machine learning can be used to effectively serve existing customers while also adequately preparing for unknown future demand. Second, I will highlight some privacy concerns that arise from the sharing of mobility data that is often required for modern data-driven algorithms. To address some of these concerns, I present techniques based on multiparty computation and differential privacy to effectively use location data to improve routing services in a privacy-preserving way.
Acknowledgments

A university environment provides a unique set of resources that aren’t found in many other places in the world. Throughout the course of my graduate studies I have shared a campus with colleagues and mentors who are not only experts in their fields of study, but also passionate educators who make learning a fun, fulfilling, and inspiring process. Before we begin I would like to spend a few moments to acknowledge the people who have helped me on my journey through graduate school.

First, I would like to thank my academic advisors Marco Pavone and Amin Saberi. Marco has an excellent ability to find opportunities for his students, encouraging and empowering them to become the best versions of themselves. Through his mentorship I’ve had the opportunity to work with companies like the Toyota Motor Company, Toyota Research Institute, NTT DOCOMO, and Grab Holdings, which included two trips to Japan to deploy some of our algorithms on real transportation systems. He’s taught me that research isn’t done in a vacuum, and has encouraged me to think about the bigger picture impact and implication of our research. Under Marco’s careful guidance I have learned how to effectively formulate, communicate, present and share ideas. Thanks to the myriad of experiences and opportunities I’ve had as Marco’s collaborator, I’ve developed confidence in my ability to learn and adapt to the unknown tasks and challenges that the future will bring, and for this I am incredibly grateful.

Amin’s mentorship helped me develop an ability to distil main ideas from complicated papers and to ask interesting technical questions. His high standards motivated me to be thorough in both problem formulation and algorithm design, and his great insight and intuition were there to help me whenever I got stuck. Amin is a very patient advisor who always valued quality over quantity and always encouraged me to take my time to develop the best possible result I could. I am truly proud of the work we did together. I’ve also had the great pleasure of taking several of Amin’s classes. Even in the pandemic, Amin did a great job in creating forums for discussion in class and encouraged people to speak and foster interactive learning. He conveys complicated ideas in a clear and elegant way, and my ability to appreciate and enjoy theoretical research benefited greatly both from his courses as well as our research meetings.

Next, I would like to thank the other members of my defense committee, Professors Mac Schwager, Dorsa Sadigh, and Stephen Boyd. In addition to attending and evaluating my defense, they
have all been role models for me during my time at Stanford.

I have had the pleasure of working with many wonderful collaborators over the years. A special thanks goes to Ramón Iglesias for showing me the ropes when I was just starting out, and for all of the long nights working on the OUI and MUI for the Toyota Ha:mo system. Though our time together was brief, I would like to thank Dejan Milojevic and Jannik Zgraggen for being excellent collaborators during their 6 month visits to Stanford. I would like to thank Kiril Solovey for many fruitful and fun discussions and collaborations on motion planning projects. Even through the COVID-19 pandemic, I was able to maintain productivity and focus because I had dedicated and flexible collaborators like Maximilian Schiffer, Kaidi Yang, Devansh Jalota, Justin Luke and Karthik Gopalakrishnan who were willing to meet at a moment’s notice to discuss ideas. I am very glad to have met each and every one of you, I and wish you the best of luck wherever your journeys may take you next.

I would like to thank the many instructors I’ve had during my time in graduate school. Learning a new topic can sometimes feel daunting, but these people fostered welcoming environments that encourage curiosity and venturing beyond one’s comfort zone. So a big thanks goes to John Duchi, Stephen Boyd, Mac Schwager, Mykel Kochenderfer, Marco Pavone, Aaron Sidford, Moses Charikar, Gregory Valiant, Itai Ashlagi, Amin Saberi, Dan Boneh, David Tse, Alex Ozdemir, Saba Eskandarian and Riad Wahby.

I would like to thank the many industry collaborators that I’ve worked with during my Ph.D. A special thanks goes to Stephen Zoepf from Lacuna Technologies who has been a valuable colleague on multiple projects spanning optimization, privacy, and equity in mobility systems. I am particularly grateful for the many valuable discussions we had on the current state of mobility policy between municipal authorities and mobility providers. I am very thankful for the opportunity to work with Chris Sholley, Keshav Puranmalka and Tim Siauw at Lyft on algorithms for high capacity shared rides. I would like to thank Stephen Hughes and Masanori Yamato from the Toyota Research Institute for working with us on a project to improve vehicle availability for Toyota’s Ha:mo carsharing service in Japan. Thanks also go to Shin Ishiguro from NTT DOCOMO for the many fruitful discussions on the role of cell phone data in modern mobility systems. I would also like to thank Xin Xu and Gracie Lin from Grab Holdings for working with us on a project exploring the efficacy of mobility systems with a mix of human-driven and autonomous vehicles.

I am very grateful for all of the friends that I’ve met during graduate school who helped me grow both professionally and personally. I’d like to thank Federico Rossi and Sumeet Singh for being role models whom I could go to for advice. I’d like to thank Alex Su, Ajay Mandlekar, Vin and Vig Sachidananda, Andrew Bylard, Benoit Landry, Gioele Zardini, Nicolas Lanzetti, and Riccardo Bonalli for all of the great memories. I’d also like to give thanks to all of my friends and mentors from the University of Illinois at Urbana-Champaign and the Illinois Mathematics and Science Academy. A special thanks goes out to Bill, Holly, and Ben Yue for being there for me when I stumbled and
needed a hand.

Finally, I would like to acknowledge the unwavering and unconditional support from my family, without which none of this would have been possible.

To my father Mark Cao, thank you for being the best mentor I could have possibly asked for. Thank you for teaching me the values of diligence, commitment and hard work. You introduced me to so many ideas and activities, ranging from swimming, ice skating, hiking, cooking, chess, the rubik’s cube, zome, and so many others. You always went out of your way to give me wonderful experiences through our family vacations, the books we would read together, and all of the hikes we went on together. Thank you for sharing your love and passion for mathematics with me. Thinking through and solving puzzles with you are among the most fulfilling and memorable moments in my life. It’s thanks to you that I developed a habit and lifestyle of pursuing learning, exploration, and personal growth. The acorn that was once so rough around the edges is starting to look a bit smoother. The way you conduct yourself and your dedication to those you love has become a central cornerstone to who I am and what I believe in, and I am truly and infinitely grateful to have you as my father.

To my mother Jane Wu, thank you for being my lighthouse that helps me find my way no matter how stormy the seas of life may get. You taught me the importance of the little things like eating well, staying healthy, maintaining a positive outlook on life, being organized, and having an open mind for unfamiliar ideas. While they may seem minuscule, they are still very important, for a journey of a thousand miles begins with a single step. You taught me that no matter how much one prepares, life is imperfect, and there is always a chance you will need to deal with unfortunate or unforeseen circumstances. Our path to this point has had its fair share of difficulties, but you held our family together in our greatest time of need. Thinking back on it, there were a million ways that things could have gone wrong, but your emotional strength, your patience, your initiative and your leadership weathered the storm and secured a brighter future for our family. I honestly believe that there aren’t many people in the world who could have overcome the hardships that you faced. I’ve learned so much about emotional strength from you and I am so proud to have you as my mother.

To my brother Felix Tsao, thank you for being my closest confidant. In the same way that no two snowflakes are ever exactly the same, the diversity of peoples’ life experiences means that no two people will have exactly the same world view. This diversity of life experiences can make finding someone who can truly relate to what you are experiencing a difficult task. In this sense I feel very fortunate to have you as a confidant. Our similar upbringings enables us relate to each others’ perspectives and offer helpful advice. On the other hand, we are different enough that we will sometimes arrive at different conclusions and challenge each other to grow. I have always been impressed by your strong sense of individuality and how you made decisions that you wanted, rather than decisions that others wanted you to want. Yours has been a truly inspirational journey that has really changed how I see the world. Because of you, I have a stronger sense of self and a better
grasp on what truly matters in life. Thank you for being my brother, my confidant, and my role model.
# Contents

Abstract iv  
Acknowledgments v  

## I Overview  

### 1 Introduction  

#### 1.1 Background and Motivation  

1.1.1 Modern Transportation Services  
1.1.2 Challenges in Modern Transportation Services  

#### 1.2 Statement of Contributions and Dissertation Outline  

#### 1.3 Publication List  

1.3.1 Dissertation Publications  
1.3.2 Additional Publications  

## II Fleet Control for On-Demand Mobility Systems  

2 Fleet Control for On-Demand Ridehailing Systems  

2.1 Introduction  

2.1.1 Statement of Contributions  
2.1.2 Organization  

2.2 Related Work  

2.3 Model & Problem Formulation  

2.3.1 Model  
2.3.2 Problem Formulation  
2.3.3 Model Discussion  

2.4 Sample Average Approximation  

2.4.1 Sample Average Approximation for AMoD Control  


<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.2</td>
<td>Convergence Properties of SAA</td>
<td>18</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Computational Complexity</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>Scalable Integer Solutions via Totally Unimodular Linear Relaxations</td>
<td>21</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Bipartite Matching for Servicing Waiting Customers</td>
<td>21</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Network Flow Optimization for Rebalancing Vehicles</td>
<td>22</td>
</tr>
<tr>
<td>2.6</td>
<td>Stochastic Optimization for Model Predictive Control of AMoD Systems</td>
<td>23</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Algorithm Discussion</td>
<td>23</td>
</tr>
<tr>
<td>2.7</td>
<td>Numerical Experiments</td>
<td>24</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Scenarios</td>
<td>24</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Experimental Design</td>
<td>25</td>
</tr>
<tr>
<td>2.7.3</td>
<td>Forecasting</td>
<td>26</td>
</tr>
<tr>
<td>2.7.4</td>
<td>Results</td>
<td>26</td>
</tr>
<tr>
<td>2.8</td>
<td>Discussion</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>Fleet Control for On-Demand Ridesharing Systems</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>30</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Statement of Contributions</td>
<td>31</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Organization</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>Related Work</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Flow Optimization Model for Double-Occupancy RAMoD</td>
<td>32</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Modeling the Road Network</td>
<td>32</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Integer Network Flow Model for RAMoD Systems</td>
<td>33</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Model Discussion</td>
<td>36</td>
</tr>
<tr>
<td>3.4</td>
<td>A Real-Time RAMoD MPC Algorithm</td>
<td>37</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Real-Time RAMoD MPC Algorithm</td>
<td>37</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Algorithm Discussion</td>
<td>38</td>
</tr>
<tr>
<td>3.5</td>
<td>Numerical Experiments</td>
<td>39</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Scenario</td>
<td>39</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Simulation Environment</td>
<td>39</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Experimental Design</td>
<td>40</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Results</td>
<td>40</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Remarks and Observations</td>
<td>41</td>
</tr>
<tr>
<td>3.6</td>
<td>Discussion</td>
<td>43</td>
</tr>
<tr>
<td>III</td>
<td>Online Matching for Ridehailing and Ridesharing Systems</td>
<td>44</td>
</tr>
<tr>
<td>4</td>
<td>Online Matching for Ridesharing</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>45</td>
</tr>
</tbody>
</table>
## 4.1 Overview of Results
- 4.1.1 Overview of Results
- 4.1.2 Organization

## 4.2 Related Work

## 4.3 Model
- 4.3.1 Offline Variants
- 4.3.2 Approximation Ratios and Competitive Ratios

## 4.4 Algorithms and Hardness for Utility Maximization
- 4.4.1 Upper Bounds
- 4.4.2 Algorithms

## 4.5 Algorithms and Hardness for Cost Minimization
- 4.5.1 Cost Assumptions on Edge Weights
- 4.5.2 The \( k = 2 \) Case
- 4.5.3 The \( k > 2 \) Case

## 4.6 Discussion

## 5 Online Matching with Reservations
- 5.1 Introduction
- 5.1.1 Statement of Contributions
- 5.1.2 Organization
- 5.2 Model
- 5.2.1 Notation
- 5.3 Related Work
- 5.4 The Reservation Priority Algorithm
- 5.4.1 Performance Guarantees for Reservation Priority
- 5.4.2 Reservation-Priority with Randomization
- 5.5 Discussion

## IV Privacy-Preserving Analysis and Computation on Mobility Data

## 6 Private Location Sharing for Routing services
- 6.1 Introduction
- 6.1.1 Statement of Contributions
- 6.1.2 Organization
- 6.2 Related Work
- 6.3 Model
- 6.3.1 Transportation Network
- 6.3.2 Users and Traffic Counts
- 6.3.3 Communication Model
## 7 Conclusions

### 7.1 Summary of Contributions
- Contributions from Part II
- Contributions from Part III
- Contributions from Part IV

### 7.2 Future Directions

## A Background: Hypergraph Matching and Set Cover

### A.1 Hypergraph Matching

### A.2 Set Cover

## B Proofs for Section 4.4

### B.1 Proof of Lemma 6

### B.2 Visualizing the Reduction from $d$-ASP to $\text{Online-}(k,d)$-Max-Matching

### B.3 Proof of Theorem 2

## C Proofs for Section 4.5

### C.1 Proof of Lemma 10

### C.1.1 Maximal Paths of Length 1
C.1.2 Type 1 Paths .................................................. 119
C.1.3 Even Length Paths: Base Case .................................. 119
C.1.4 Induction Step on the Path Length ............................ 121
C.1.5 Base Case for Type 2 Paths of Length at least 3 .......... 122

D Proofs for Section 5 ............................................. 126
D.1 Proof of Theorem 7 .............................................. 126
   D.1.1 Setup ......................................................... 126
   D.1.2 Proof ......................................................... 126
D.2 Proof of Lemma 12 .............................................. 128
D.3 Proof of Theorem 8 .............................................. 130
   D.3.1 Constructing a Worst Case Instance for Reservation-Priority 130
   D.3.2 Performance of Reservation-Priority on G .................. 131

E Proofs for Section 6 ............................................. 135
E.1 Proof of Theorem 9 .............................................. 135
   E.1.1 Proving Lemma 14 in Case 1 ............................... 135
   E.1.2 Proving Lemma 14 in Case 2 ............................... 136

F Proofs for Section 2 ............................................. 139
F.1 Proof of Lemma 1 ............................................... 139
F.2 Proof of Lemma 2 ............................................... 139
F.3 Proof of Lemma 3 ............................................... 142
F.4 Proof of Lemma 4 ............................................... 143

Bibliography ...................................................... 145
List of Tables

2.1 Wait times for the DiDi scenario (seconds) ........................................ 27
2.2 Wait times for the NYC scenario. (seconds) ...................................... 27

3.1 Performance of the Controllers .................................................. 41

6.1 Demand scenarios used in our simulation ....................................... 98
6.2 Performance measures for $\epsilon = 0.01$ ....................................... 99
6.3 Performance measures for $\epsilon = 0.1$ ......................................... 100
List of Figures

2.1 A heatmap displaying the surplus and deficit of Grab service vehicles during the evening rush hour in Singapore. Vehicle imbalance is evident from the fact that the northern part of Singapore experiences vehicle deficit (red regions), whereas the south sees significant vehicle surplus (purple regions). ................................. 13

2.2 Total demand for the DiDi ridehailing service between January 1st and January 21st in 2016. Each color represents a different day. ...................................................... 13

2.3 The average demand for the DiDi ridehailing service between January 1st and January 21st (red) compared to the daily values (black). The DiDi demand data exhibits significant variance. .................................................... 14

2.4 The top plot shows the number of waiting customers as a function of time for both our controller (MPC-LSTM-SAA) in blue and the controller from [44] in green. Under the operation of our controller, there are significantly fewer waiting customers throughout the day, which is reflected by the waiting times. Unexpectedly, as a price of this improved service, our controller issues more rebalancing requests, since it is planning for many outcomes, as shown by the middle plot. Looking at the middle and bottom plots together, we see that our controller does additional rebalancing precisely when there is significant variance in the demand. ........................................ 27

2.5 The top plot shows the number of waiting customers as a function of time for several controllers. As expected, compared to the reactive controller, the predictive controllers have more customers wait at the beginning of the simulation in order to better prepare for the customers appearing later. Leveraging stochastic optimization, MPC-LSTM-SAA outperforms MPC-LSTM-LP and Reactive in terms of mean waiting time. As a tradeoff, the bottom plot shows that the reactive controller issues the least amount of rebalancing tasks, while MPC-LSTM-SAA issues the most. .... 28

3.1 Schematic representation of a double-occupancy RAMoD system. Customers entering the system (downward-facing arrows) are collected and dropped off (upward-facing arrows) in a first-in-first-out fashion. ...................................................... 31
3.2 Visualization of the San Francisco experiment with AMoDeus [42]. Green, blue, orange and red dots represent empty, rebalancing, on pick-up, and customer carrying vehicles respectively. Since the road graph models each station as a single node, it does not capture the structure within stations. A lower level controller is used to dispatch vehicles within the station.

3.3 Relative performance difference with respect to the proposed RAMoD-MPC algorithm.

3.4 Number of waiting customers for the controllers tested as a function of time (upper plot) and distance traveled by the fleet per unit time (lower plot).

4.1 An example of batching for the online matching problem. The time horizon is partitioned into batches of length \( d \), and an offline algorithm is used to compute a matching (blue hyperedges) for each batch.

4.2 Consider a deterministic batching algorithm with \( z = 0 \). The batching algorithm is unable to select the red edges, since they are not visible with respect to \( z = 0 \). If the red edges each have weight 1 and all other edges have weight \( \epsilon \), then the weight of the matching obtained by batching is at most \( \epsilon |E| \), whereas the optimal solution has weight at least 3. Hence the competitive ratio of batching is at most \( \frac{\epsilon |E|}{3} \), which converges to zero as \( \epsilon \to 0 \).

4.3 The minimum cost matching is highlighted in green and blue respectively for instances \( H_1 \) and \( H_2 \).

4.4 The observed graph when \( A \) becomes critical under both \( H_1 \) and \( H_2 \).

4.5 A visualization of the graph in Example 1. \( A \) expires before \( C \) arrives, so an algorithm needs to decide between matching \( A \) to \( B \) to collect an immediate reward, or keep \( B \) unmatched in case \((B, C)\) turns out to be a more favorable match.

5.1 A plot of achievable competitive ratios in various different reservation models as a function of the fraction of reservations \( p \). The black curve shows the best possible reservation-agnostic algorithm, RANKING [51]. Red shows the performance of [54] under an reservation model where a \( p \) fraction of online vertices are adversarially chosen to be reserve vertices. Blue shows the performance of Reservation-Priority in the OBMRR model.

5.2 In most routing services, users give their location data in exchange for route recommendations. Routing services often sell this data in data marketplaces. Third parties who buy location data from these routing services will be able to infer preferences, habits, and schedules of users who frequently interact with routing services.
6.2 A visualization of the routing service protocol described in Algorithm 8. Users traveling in the transportation network share their location data in a privacy-preserving way to estimate the traffic counts in a decentralized manner (upper left). These counts are then used to estimate travel times (right). When a user requests a route from the routing service, a shortest path is computed using the estimated travel times (lower left). ............................................................. 81

6.3 Distribution of δ-critical counts for all roads in the network for δ = 0.1. The dashed red line denotes the threshold above which the road travel time estimate will have an error less than 10%. 90% of the time to give a privacy level of ϵ = 0.2. ............. 99

B.1 An example of a shareability hypergraph G ∈ G evolving in time. This example has d = 9. Vertices arrive one per timestep on the right, and depart one per timestep from the left. A vertex is red during the last timestep it is in the system. For each t, e_t is revealed at time t + d − 1, when its last vertex appears. It also becomes critical at this time, since t ∈ e_t is about to leave the system. Therefore, from timestep d − 1 onward, one edge, along with its weight, is revealed each timestep, and can only be included into the matching at that timestep. Furthermore, at most one edge can be in the matching, since all edges contain the node d − 1 = 8. Hence, the family G encodes the d-ASP problem. ............................................................. 113

D.1 A visualization of the graph G. A_1, A_2, B_1, B_2 each have m vertices while A_3, B_3 each have \( \left( \frac{p}{1-p} + \sqrt{\frac{4 \log m}{(1-p)^3} m} \right) m \) vertices. The light gray area represents the perfect matching between A and B. The dark gray areas represent the fully connected subgraphs between (A_2, B_1) and (A_3, B_2). ............................................................. 132
Part I

Overview
Chapter 1

Introduction

1.1 Background and Motivation

The transportation industry is one among many fields that are being revolutionized by advances in information technology related to data collection, processing, and storage. In particular, mobile devices offer a fast, ubiquitous and convenient way for customers to communicate their preferences and needs to mobility service providers in real-time. As a result, the past decade has seen a significant increase in the number of transportation systems that now provide on-demand service, meaning that transportation and delivery are available to customers at a moment’s notice, wherever and whenever they may need it [47, 49, 46, 45].

1.1.1 Modern Transportation Services

Among the plethora of new transportation and delivery systems, the types of services that have gained the most adoption include ridehailing, ridesharing, food delivery, carsharing, bikesharing and routing services.

**Ridehailing services** coordinate a fleet of (possibly crowd-sourced) service vehicles to provide an on-demand taxi service wherein customers can hail a ride from their phone and be promptly picked up and delivered to their destination by one of the fleet’s service vehicles. Examples of ridehailing systems include Uber, Lyft, DiDi and Grab.

**Ridesharing services** are a special type of ridehailing service wherein a service vehicle can carry more than one customer at a time. Ridesharing typically improves the utilization rate of the service fleet by carrying multiple customers within each service vehicle. Customers of ridesharing services can experience longer wait times or detours in their route, but the fleet’s increased efficiency can often translate to lower prices for such users when compared to
CHAPTER 1. INTRODUCTION

ridehailing services. Many ridehailing services including, but not limited to Uber, Lyft, and Grab, also have ridesharing offerings.

Food delivery services function similarly to ridehailing in the sense that they coordinate a fleet of vehicles to provide a transportation service. However, instead of transporting customers to their desired locations, they deliver food from restaurants or supermarkets to the homes of the customers. Examples of food delivery services include DoorDash, InstaCart, and Grubhub. Ridehailing services sometimes double as food delivery services in the case of UberEats and GrabFood, which are part of Uber and Grab respectively.

Vehicle sharing services offer convenient vehicle rentals. This is typically done by building stations in many popular areas of a city. Vehicles parked at a station are available for rent, and rented vehicles can be returned at any station so long as there is a vacant parking spot. The convenience of vehicle sharing services thus depends on the density of stations within a city. The most popular forms of vehicle sharing include carsharing, bikesharing, and scootersharing. Zipcar is a commonly used carsharing service, whereas Citi Bike is a popular bikesharing service in the New York area.

Routing services provide routing recommendations to transportation queries. The recommendations are often based on estimated congestion levels of the roads within the transportation network so that new recommendations will steer users away from heavily congested areas. Users of routing services (i.e., those who submit transportation queries to the routing service) include owners of personal vehicles as well as ridehailing and ridesharing services that want the best route to take customers to their desired destinations. Examples of routing services include Google Maps, Apple Maps, and Waze.

1.1.2 Challenges in Modern Transportation Services

While advances in information technology have endowed modern transportation services with new functionalities, the optimization of these new functionalities introduces several new and important challenges.

Challenge #1: Vehicle imbalance in ridehailing and vehicle sharing. Ridehailing systems often experience a spatial imbalance in the pickup locations and dropoff locations requested by customers. If left unattended, this will lead to an imbalance in the supply of service vehicles where popular dropoff locations will have a surplus of vehicles, and popular pickup locations will experience vehicle scarcity. Such a situation would make it difficult for future customers to get service. Such imbalance happens in practice, as operational data from Grab illustrates in Figure 2.1. Carsharing and bikesharing services also suffer from demand imbalance in practice. With this issue in mind, the service fleet should effectively serve current
customers while also rebalancing its vehicles to better align them with future demand so that future customers can find service quickly.

**Challenge #2: Real-time rider-rider matching in ridesharing systems.** Ridesharing systems aim to increase vehicle utilization by having service vehicles carry multiple customers at a time. As such, a ridesharing system must be able to identify ridesharing groups, i.e., groups of customers that can be efficiently served by a single vehicle. This task is complicated by the fact that ridesharing is an on-demand service, meaning that riders expect quick responses to their travel requests. This need for quick responses limits the ridesharing service's ability to collect information about future demand, i.e., a ridesharing system must quickly assign incoming riders into ridesharing groups while also adequately preparing for unknown future demand. In the face of this challenge, a ridesharing service must properly balance the merits of (a) assigning ridesharing groups quickly to minimize user waiting time, and (b) discovering more cost-efficient groupings by waiting for more riders to appear (since arrival of new riders increases the set of ridesharing groups that the system can choose from).

**Challenge #3: Privacy in the sharing of mobility data.** Data-driven methodologies offer many exciting upsides, but they also introduce new challenges, particularly in the realm of user privacy. Specifically, the way data is collected can pose privacy risks to end users. Many routing services use location data to estimate congestion levels in the transportation network in order to effectively provide route recommendations. In such a setup, a single entity (e.g., the routing service provider) collects and manages user trajectory data. When it comes to user privacy, these systems have a central point of failure since users have to trust that this entity will not sell, distribute or use their data to infer sensitive private information. This issue motivates the exploration of aggregation techniques to estimate total network congestion without exposing individual level location data for any user. To avoid single points of failure, the aggregation procedure should be decentralized. Fundamental tradeoffs between privacy, utility, and computational complexity should be studied to identify application instances (if any) where both privacy and utility can be obtained.

### 1.2 Statement of Contributions and Dissertation Outline

This thesis presents several techniques for addressing the three challenges described in Section 1.1.2. Part I presents methods to address Challenge #1 for ridehailing and ridesharing systems in settings where a fleet operator has control over the actions of the entire fleet. As such, the operator can issue rebalancing tasks to the vehicles to have them reposition themselves to better align with future demand. Such a setting is realistic when a ridehailing or ridesharing service employs paid contractors which will follow the commands issued by the operator, and in a futuristic setting where the service vehicle fleet is comprised of autonomous vehicles.
In Section 2, we present a ridehailing fleet controller which uses both on receding horizon optimization and demand forecasting to balance considerations for existing customers and future customers. Existing work on ridehailing fleet control either aim to maintain uniform availability through the network [68] or use point estimates of future demand to determine rebalancing actions [76, 44]. However, ridehailing demand has significant variance in practice (see Figures 2.2 and 2.3) so we developed a control algorithm [83] which predicts a distribution for future demand rather than predicting point estimates, and then uses stochastic control techniques to efficiently incorporate the predicted demand distribution into a receding horizon control loop. Numerical experiments show that the uncertainty quantified by predicting demand distributions leads to more robust behavior when compared to control algorithms that rely on point estimates for future demand.

In Section 3, we generalize the control algorithm presented in Section 2 to enable 2-capacity ridesharing, meaning that each service vehicle can carry up to 2 customers at a time. The main challenge involved in the generalization is that the capabilities of a service vehicle depend on its occupancy. Indeed, a vehicle carrying 2 customers cannot pick up any more customers until it drops off its current customers, whereas an empty vehicle can pick up 2 customers. To address this challenge, we present a ridesharing fleet controller [84] which models the vehicle fleet as a collection of subfleets where each subfleet represents the set of vehicles with a certain occupancy level (e.g., 0, 1, or 2). This methodology can also be used to operate fleets of heterogeneous vehicles, which may be of independent interest. Existing work in ridesharing fleet control either do not enable rebalancing idle vehicles to better align with future demand [78] or decouple the rebalancing from the process that assigns riders to drivers, meaning that future demand is not considered when making rider to driver assignments [5]. The proposed control algorithm computes rider to driver assignments and rebalancing actions together, and we show through simulation experiments that it performs better than existing methods which don’t account for future demand.

Part III presents techniques from matching theory for operation of ridesharing and ridehailing systems. In particular, Chapter 4 presents hypergraph matching algorithms to address Challenge #2, and Chapter 5 shows how rider reservations can improve the efficiency of rider-driver matching in ridehailing systems. The motivation to study Challenge #2 comes from the fact that the ridesharing fleet controller presented in 3 can become computationally expensive when deployed on transportation networks with high spatial and temporal resolution, or when considering large vehicle capacities. This is due to the joint optimization of rider to driver assignments and rebalancing actions. One way to avoid such computational difficulties is to decouple the computations of driver to rider assignment and rebalancing actions. To this end, a decoupled strategy would assign riders to drivers by computing a collection of ridesharing groups and use an existing algorithm [68, 76, 44, 83] to handle vehicle rebalancing.

In Section 4, we study the problem of rider to driver assignment in ridesharing settings through the lens of hypergraph matching. We represent a ridesharing system as a hypergraph where vertices
CHAPTER 1. INTRODUCTION

represent rider requests and hyperedges represent ridesharing groups. This representation is designed so that every feasible collection of ridesharing groups (i.e., a collection where each customer is part of at most one group) corresponds to a matching in the hypergraph. In this way, the rider to driver assignment problem can be cast as an online hypergraph matching problem. We present algorithms for both utility maximization and cost minimization settings [66, 67], where the ridehailing system’s objective is to maximize its utility or minimize its operation costs respectively. Existing work explores a 2-capacity utility maximization problem from a worst-case perspective, motivated by ridesharing and kidney exchange applications [9]. Other works study 2-capacity matching under specific stochastic arrival and departure processes [3, 7]. We chose to follow the worst-case approach of [9] because rider arrival and departure processes of real-world ridesharing systems is complex and does not have the i.i.d. or stationary structures assumed in [3, 7].

In Section 4.4 we generalize the results of [9] from 2-capacity to $k$-capacity ridesharing settings where service vehicles can carry up to $k$ riders at a time. This is done by representing the ridesharing setting with a hypergraph rather than a graph. Therein we present a randomized batching algorithm and prove that it has the optimal competitive ratio for the online hypergraph matching problem while also running in polynomial time. In Section 4.5 we study online hypergraph matching with a cost minimization objective. Under the assumption that the cost of hyperedges satisfies a sublinear property, we present an optimal deterministic algorithm in the $k = 2$ setting, and we present a randomized batching algorithm in the $k > 2$ setting whose competitive ratio is within a constant factor of the optimal competitive ratio achievable in polynomial time.

In Chapter 5 we present a technical note on improving worst-case performance guarantees in ridehailing systems by incorporating some side information about the demand. One critique for worst-case analysis is that worst-case instances may not fully capture domain knowledge or structure that is present in typical real-world problem instances. To address this concern, we study a rider-driver matching problem for ridehailing systems where the system obtains partial information on the future demand through rider reservations. When reserving their ride, a rider gives their travel preferences to the system ahead of time, which we show can be leveraged to achieve a more efficient outcome than cases where no side information is available (e.g., no riders make any reservations). This rider reservation setup is an interpolation between classical online models (e.g., no reservations are made) and classical offline models where the demand is fully known (e.g., all riders make reservations), and thus provide performance guarantees that are less conservative than those of online model while relaxing the information requirement of offline models.

Part IV presents techniques to address Challenge #3 in the context of routing services. Routing services require estimates on network congestion to recommend routes that steer users away from congested areas. However, most existing approaches estimate congestion by having a single entity collect individual level location and trajectory data from users. Repeated sharing of this data through many interactions with a routing service can reveal habits, preferences, and schedules of
users.

Many existing works that attempt to provide privacy guarantees in mobility applications propose differentially private mechanisms to anonymize mobility data sets, thereby providing anonymous, general purpose data for public use \[4, 37, 59\]. However, these approaches suffer from poor accuracy since differential privacy was not designed to be used in this way. In fact, \[25\] shows that anonymization comes at a cost: there will always be simple properties of a data set that will be erased by anonymization procedures. Differential privacy was designed for use in interactive protocols where scientists submit questions to a data center, and the data center provides privacy-preserving responses \[25\]. The main idea in differential privacy is for the data center to inject carefully chosen query-specific noise to answer queries as accurately as possible while still hiding user data at the individual level.

In Section \[6\] we propose a decentralized location sharing protocol where users on the road will periodically compute and announce the traffic counts (i.e., approximate number of vehicles traveling on each road) of a transportation network in a decentralized and privacy-preserving manner \[85\]. The protocol uses differentially private mechanisms, but does not suffer from the poor accuracy issues experienced by \[4, 37, 59\] because the protocol’s only objective is to estimate traffic counts, rather than trying to preserve all attributes of the location data. Since only the total number of vehicles on each road is announced, the location of individual users is not discernible by observers, which is contrary to many current location sharing setups where users give their individual location data directly to routing services. With this protocol, user privacy does not rely on a trusted data custodian, and there are no single points of failure. The protocol is computationally efficient and does not require specialized hardware; all it needs is GPS, which is included in most mobile devices. Furthermore, assuming the roads in the network are sufficiently large, we can prove that the travel time estimates produced by the protocol will be close to the estimates produced by the ground truth with high probability. We corroborate this insight using numerical experiments which show that the protocol provides a privacy-preserving routing service with minimal overhead to the travel time of users.

1.3 Publication List

1.3.1 Dissertation Publications

The contributions of this dissertation outlined in Section \[1.2\] covers results from the following publications:

\[83\] Stochastic model predictive control for autonomous mobility on demand (ITSC18)

Matthew Tsao, Ramon Iglesias, and Marco Pavone.
Model predictive control of ride-sharing autonomous mobility on demand systems (ICRA19)
Matthew Tsao, Dejan Milojevic, Claudio Ruch, Mauro Salazar, Emilio Frazzoli, and Marco Pavone.

Online Hypergraph Matching with Delays (WINE20)
Marco Pavone, Amin Saberi, Maximilian Schiffer, and Matthew Tsao.

Online Hypergraph Matching with Delays (Operations Research 2022)
Marco Pavone, Amin Saberi, Maximilian Schiffer, and Matthew Tsao.

Trust but verify: Cryptographic data privacy for mobility management (TCNS22)
Matthew Tsao, Kaidi Yang, Stephen Zoepf, and Marco Pavone.

Private location sharing for decentralized routing services (ITSC22)
Matthew Tsao, Kaidi Yang, Karthik Gopalakrishnan, and Marco Pavone.

1.3.2 Additional Publications
Additional works published during my Ph.D are listed below.

Robust and Adaptive Planning under Model Uncertainty (ICAPS19)
Apoorva Sharma, James Harrison, Matthew Tsao and Marco Pavone.

A Model Predictive Control Scheme for Intermodal Autonomous Mobility-on-Demand (ITSC19)
Jannik Zgraggen, Matthew Tsao, Mauro Salazar, Maximilian Schiffer and Marco Pavone.

A Congestion-aware Routing Scheme for Autonomous Mobility-on-Demand Systems (ECC19)
Mauro Salazar, Matthew Tsao, Izabel Aguiar, and Marco Pavone.

Sample Complexity of Probabilistic Roadmaps via Epsilon-nets (ICRA20)
Matthew Tsao, Kiril Solovey, and Marco Pavone.
Real-Time Control of Mixed Fleets in Mobility-on-Demand Systems (ITSC21)
Kaidi Yang, Matthew Tsao, Xin Xu and Marco Pavone.

Balancing Fairness and Efficiency in Traffic Routing via Interpolated Traffic Assignment (AAMAS22)
Devansh Jalota, Kiril Solovey, Matthew Tsao, Stephen Zoepf, and Marco Pavone.
Part II

Fleet Control for On-Demand Mobility Systems
Chapter 2

Fleet Control for On-Demand Ridehailing Systems

This section presents a stochastic, model predictive control (MPC) algorithm that leverages short-term probabilistic forecasts for fleet dispatching and rebalancing in ridehailing systems. The algorithm periodically estimates a probability distribution for near-future demand. It then solves a stochastic optimization problem in a receding horizon manner to balance the tasks of serving existing customers and repositioning its vehicles to prepare for future demand. In a simulation using customer data from the ridesharing company DiDi Chuxing, the algorithm presented here exhibits a 62.3 percent reduction in customer waiting time compared to state of the art non-stochastic algorithms.

2.1 Introduction

The last decade has witnessed a rapid transformation in urban mobility. On one hand, Mobility-on-Demand (MoD) services like ridehailing (e.g. Uber and Lyft) and carsharing (e.g. Zipcar, Car2Go) have become ubiquitous due to the convenience and flexibility of their services. On the other hand, the advent of self-driving vehicles promises to further revolutionize urban transportation. Autonomous Mobility-on-Demand is the union of these two technologies where a fleet of self-driving vehicles is used to provide a ridehailing service. Some expect that AMoD will have a profound impact on modern transportation, and in particular could reduce personal vehicle ownership [64].

AMoD systems present a unique opportunity to address the widespread problem of vehicle imbalance: Due to a spatial imbalance in the pickup and dropoff locations requested by customers, ridehailing vehicles tend to accumulate in popular dropoff locations and become depleted near popular pickup locations [32], hampering the quality of service. Figure 2.1 shows vehicle imbalance in Grab’s Singapore ridehailing operations. Unlike existing MoD systems, an AMoD operator can
order a fleet of self driving vehicles to rebalance, i.e., to reposition its vehicles to better align with future demand.

This observation has led to the development of controllers that attempt to optimally rebalance AMoD systems in real-time. However, as we discuss in the literature review, most of the existing controllers either ignore future demand, assume deterministic future demand, or do not scale to large systems. In particular, while travel demand follows relatively predictable patterns, it is subject to significant uncertainties due to externalities such as, e.g., weather and traffic. Indeed, figures 2.2 and 2.3 show that the daily ridehailing demand for DiDi Chuxing can significantly deviate from its mean. Successful AMoD systems must cope with these uncertainties. The goal of this chapter is to propose a stochastic model-predictive control approach for vehicle rebalancing that leverages short-term travel demand forecasts while considering their uncertainty.

### 2.1.1 Statement of Contributions

The contributions of this chapter are threefold. First, we develop a stochastic MPC algorithm that leverages travel demand forecasts and their uncertainties to assign and reposition empty, self-driving vehicles in an AMoD system. Second, we provide high probability bounds on the suboptimality of the proposed algorithm when competing against an oracle controller which knows the true distribution of customer demand. Third, we demonstrate through experiments that the proposed algorithm outperforms the aforementioned deterministic counterparts when the demand distribution has significant variance. In particular, on the same DiDi Chuxing dataset, our controller yields a 62.3 percent reduction in customer waiting time compared to the work presented in [44].

### 2.1.2 Organization

The remainder of the chapter is organized as follows. We introduce the AMoD rebalancing problem in Section 2.3 and we formulate it as an explicit stochastic integer program using a Sample Average Approximation (SAA) approach in Section 2.4. In Section 2.5 we discuss approximation algorithms to rapidly solve such an integer program, while in Section 2.6 we leverage the presented results to design a stochastic MPC scheme for AMoD systems. In Section 2.7 we compare the proposed MPC scheme against state-of-the-art algorithms using numerical simulations. Section 2.8 concludes the chapter with a brief discussion and remarks on future research directions.

### 2.2 Related Work

To keep this chapter concise, we limit our review to work that specifically addresses AMoD systems, although similar ideas can be found in the MoD literature. We categorize prior work in real-time control of AMoD systems in two broad classes: i) reactive control methods that do not make
Figure 2.1: A heatmap displaying the surplus and deficit of Grab service vehicles during the evening rush hour in Singapore. Vehicle imbalance is evident from the fact that the northern part of Singapore experiences vehicle deficit (red regions), whereas the south sees significant vehicle surplus (purple regions).

Figure 2.2: Total demand for the DiDi ridehailing service between January 1st and January 21st in 2016. Each color represents a different day.
assumptions about future demand and ii) Model Predictive Control (MPC) algorithms that are able to leverage signals about future demand. Reactive, time-invariant methods span from simple bipartite matching, to control methods based on fluidic frameworks. A good comparison of different reactive controllers can be found in [99] and [41], where, notably, both studies show that the controller first proposed in [69] performs competitively across tests. However, these controllers do not provide a natural way to leverage travel demand forecasts.

In contrast, time-varying MPC algorithms, such as those proposed in [99, 44, 63], provide a natural way to leverage travel forecasts. However, [99] suffers from computational complexity as the fleet size grows, and [99, 44] do not account for uncertainty on the forecasts. While the authors of [44] show impressive results in their experiments, it can be shown that the difference between the stochastic optimum and the certainty equivalent one can be arbitrarily large. To address stochasticity of demand, [63] proposes a distributionally robust approach leveraging semidefinite programming. However, their model makes a restrictive Markovian assumption that exchanges fidelity for tractability. Moreover, the authors do not address how to recover integer rebalancing tasks from the fractional strategy provided by the controller.

To the best of our knowledge, there is no existing AMoD controller that i) exploits travel demand forecasts while considering its stochasticity, ii) produces actionable integer solutions for real-time control of AMoD systems, and iii) scales to large AMoD systems.

Figure 2.3: The average demand for the DiDi ridehailing service between January 1st and January 21st (red) compared to the daily values (black). The DiDi demand data exhibits significant variance.
2.3 Model & Problem Formulation

In this section, we first present a stochastic, time-varying network flow model for AMoD systems that will serve as the basis for our control algorithms. Unlike in [44], the model does not assume perfect information about the future, instead it assumes that customer travel demand follows an underlying distribution, which we may estimate from historical and recent data. Then, we present the optimization problem of interest: how to minimize vehicle movements while satisfying as much travel demand as possible. Finally, we end with a discussion on the merits and challenges of the model and problem formulation.

2.3.1 Model

Let $G = (V, E)$ be a weighted graph representing a road network, where $V$ is the set of discrete regions (also referred to as stations), and the directed edges $E$ represent the shortest routes between pairs of stations. We consider $G$ to be fully connected so there is a path between any pair of regions. Accordingly, let $n = |V|$ denote the number of stations. We represent time in discrete intervals of fixed size $\Delta t$. The time it takes for a vehicle to travel from station $i$ to station $j$, denoted $\tau_{ij}$, is an integer multiple of $\Delta t$ for all pairs $i, j \in V$.

At time $t$, we consider a planning horizon $\mathcal{T}$ of $T$ consecutive time intervals, i.e. $\mathcal{T} = [t + 1, t + 2, \ldots, t + T]$. For notational convenience and without loss of generality, we will always assume that the beginning of the planning horizon is at time $t = 0$. For each time interval in $\mathcal{T}$, $\lambda_{ijt}$ represents the number of future passengers that want to go from station $i$ to station $j$ at time interval $t$. However, the travel demand is a random process. Thus, we assume that the travel demand $\Lambda = \{\lambda_{ijt}\}_{i,j \in V, \forall t \in \mathcal{T}}$ within the time window $\mathcal{T}$ is characterized by a probability distribution $P$. Additionally, $\lambda_{ij0}$ denotes the number of outstanding passengers who have already issued a request to travel from $i$ to $j$ some time in the past but have not yet been serviced. Note that it is safe to assume that $\lambda_{ij0}$ is always known (since keeping track of waiting customers is relatively trivial) and, therefore, deterministic.

Within the same time window, there are $m$ self-driving vehicles which are either idling, serving a customer, or executing a rebalancing task. Thus, the availability of these vehicles is location and time-dependent. Specifically, $a_i$ is the number of idle vehicles at the beginning of the time window at station $i$, and $v_{it}$ the number of vehicles which are currently busy, but will finish their current task and become available at time $t$ at station $i$. Thus, the total number of available vehicles in the system as a function of location and time is given by

$$s_{it} := \begin{cases} a_i + v_{it} & \text{if } t = 1, \\ v_{it} & \text{if } t > 1. \end{cases}$$

Vehicle movements are captured by $x$, i.e., $x_{ijt}$ is the number of cars, rebalancing or serving customers, which are departing from $i$ at time $t$ and traveling to $j$. Note that vehicles must satisfy
flow conservation, such that, the number of vehicles arriving at a station at a particular time equals
the number of departing vehicles. Formally:

$$\sum_{j=1}^{n} x_{ijt} = s_{it} + \sum_{j=1}^{n} x_{ji(t-\tau_{ji})}, \forall i \in \mathcal{V}, t \in \mathcal{T}. \quad (2.1)$$

Finally, \( w \) captures outstanding customers, such that \( w_{ijt} \) is the number of outstanding customers
who waited until time \( t \) to be transported from station \( i \) to station \( j \). All outstanding customers
must be served within the planning horizon:

$$\sum_{t=0}^{T} w_{ijt} = \lambda_{ij0} \forall i, j \in \mathcal{V}. \quad (2.2)$$

2.3.2 Problem Formulation

Our objective is to minimize a combination of i) the operational cost based on vehicle movement,
ii) the waiting time for outstanding customers and iii) the expected number of customers who upon
arrival do not find an available vehicle in their region. Given \( (\zeta)_+ := \max\{0, \zeta\} \) and a vehicle
availability state \( \{s_{it}\}_{i \in \mathcal{V}, t \in \mathcal{T}} \), the goal is to solve the following optimization problem:

$$\min_{x, w} c_T x + c_w^T w + \mathbb{E}_P \left[ \sum_{ijt} c_{\lambda,ijt}(\lambda_{ijt} + w_{ijt} - x_{ijt})_+ \right], \quad (2.3)$$

s.t (2.1), (2.2).

\( w, x \in \mathbb{N}^{n^2 T} \)

The first term in the objective, where \( c_x \) := \( \{c_{x,ijt}\}_{i,j \in \mathcal{V}, t \in \mathcal{T}} \), is the operational cost, i.e. the
cost of operating the fleet (including, e.g., fuel, maintenance, depreciation) in proportion to total distance
traveled. Similarly, the second term \( c_w^T w \) penalizes customer waiting times by a cost vector \( c_w \), where
\( c_{w,ijt} \) is the cost of making an outstanding customer wanting to travel between stations \( i \) and \( j \) wait
until time interval \( t \) to be served. The last term penalizes the expected mismatch between customer
demand and the vehicle supply, that is \( c_{\lambda,ijt} \) is the cost of not being able to serve a customer wanting
to travel between \( i \) and \( j \) at time \( t \). Finally, in addition to the previously mentioned constraints, \( x \)
and \( w \) must be positive integers since fractional vehicles and customers are non-physical.

2.3.3 Model Discussion

There are two key challenges in solving (2.3). First, \( P \) is a time varying high dimensional probability
distribution which is generally not known. Hence, one cannot evaluate the objective function explicitly.
Secondly, due to the integer constraints on \( x, w \), (2.3) is an instance of integer programming
which is NP-hard, such that no known polynomial time algorithms exist and the problem remains computationally intractable for large inputs.

In the following sections, we present a series of relaxations that allow us to efficiently obtain solutions to a surrogate problem that approximates (2.3). Specifically, to address the unknown distribution in the objective function, we fit a conditional generative model on historical data to predict future demand given recent realizations of demand. To address the computational complexity of integer programming, we perform several relaxations to arrive at a linear programming surrogate problem. Finally, we present bounds on the optimality gap induced by making these relaxations.

### 2.4 Sample Average Approximation

Since $P$ is an unknown, time varying distribution, we cannot explicitly evaluate the objective in (2.3). To address this issue, we present a SAA problem whose objective function approximates the objective of (2.3) in section 2.4.1. In section 2.4.2 we give sufficient conditions under which the solution to the SAA problem from 2.4.1 is near optimal for the original problem. We address the trade-off between solution accuracy and problem complexity in section 2.4.3.

#### 2.4.1 Sample Average Approximation for AMoD Control

Despite not knowing $P$, nor being able to sample from it, we have historical data from $P$ that we use to train a conditional generative model $\hat{P}$ to mimic the behavior of $P$. With a generative model in hand, one can consider solving (2.3) with $\hat{P}$ instead of $P$.

However, in many cases solving a stochastic optimization problem exactly is not possible if the underlying distribution does not have a computationally tractable form. Many popular probabilistic generative models, such as Bayesian networks and Bayesian neural networks fall into this category. To overcome this issue, we can sample from the generative model and replace expectations with Monte Carlo estimates to get approximate solutions, a method commonly referred to Sample Average Approximation (SAA) [22][12]. To this end we generate $K$ samples $\{\{\lambda_{ijt}^k\}_{i,j\in[n],t\in[T]}\}_{k=1}^K \sim \hat{P}$ and approximate expectations under $\hat{P}$ with Monte Carlo estimates, i.e.

$$
\mathbb{E}_{\hat{P}} \left[ \sum_{ijt} (\lambda_{ijt} + w_{ijt} - x_{ijt})_+ \right] \approx \frac{1}{K} \sum_{k=1}^K \sum_{ijt} (\lambda_{ijt}^k + w_{ijt} - x_{ijt})_+ .
$$

Using this approximation, we consider the following SAA surrogate problem:

$$
\min_{x,w} \ c_x^T x + c_w^T w + \frac{c_x}{K} \sum_{k=1}^K \sum_{ijt} (\lambda_{ijt}^k + w_{ijt} - x_{ijt})_+ .
$$
CHAPTER 2. FLEET CONTROL FOR ON-DEMAND RIDEHAILING SYSTEMS

\[ \text{s.t. } \sum_{i \in T} w_{ijt} = \lambda_{ij0} \quad \forall i, j \in [n] \]

\[ \sum_{j=1}^{n} x_{ijt} - x_{j(i-t+ij)} = s_{it} \quad \forall i \in [n], t \in T \]

\[ u_{ijt}^k \geq 0 \quad \forall k \in [K], i, j \in [n], t \in T \]

\[ u_{ijt}^k \geq \lambda_{ijt}^k + w_{ijt} - x_{ijt} \quad \forall k \in [K], i, j \in [n], t \in T \]

\[ \{ w_{ijt}^k \}_{k=1}^K, x, w \in \mathbb{N}^{n^2 T} \quad \forall k \in [K], i, j \in [n], t \in T, \]

where, in addition to the Monte Carlo estimate, we include a series of inequalities to make the objective function linear. Specifically, minimizing \((x)_+\) is equivalent to minimizing \(u\) with the constraints \(u \geq 0, u \geq x\). The surrogate SAA problem (2.4) is directly solvable by off-the-shelf mixed integer linear programming (MILP) solvers.

2.4.2 Convergence Properties of SAA

In this section, we compare the quality of the solutions to (2.3) and (2.4) when evaluated by the objective in (2.3). Specifically, we present a result stating that if \( \hat{P} \) is close to \( P \) in an appropriate sense and we use enough samples for the SAA in (2.4), then the obtained solution is with high probability, provably near optimal for the original problem in (2.3) that we would have solved had we known \( P \). Such a result is called an oracle inequality. Using the notation

\[ F(x, w) := c_\lambda \mathbb{E}_P \left[ \sum_{ijt} (\lambda_{ijt}^* + w_{ijt}^* - x_{ijt}^*)_+ \right] \]

\[ \hat{F}_K(x, w) := \frac{c_\lambda}{K} \sum_{k=1}^K \left[ \sum_{ijt} (\lambda_{ijt}^k + w_{ijt} - \hat{x}_{ijt})_+ \right], \]

the difference between the objectives in (2.3) and (2.4) is \( F(x, w) - \hat{F}_K(x, w) \). Consider the following lemma:

**Lemma 1** (\( \| \cdot \|_\infty \)-continuity of function minima). Let \( f, g : \mathcal{X} \to \mathbb{R} \) denote two real valued functions that have finite global minima, i.e., both \( x_f \in \arg \min_{x \in \mathcal{X}} f(x) \) and \( x_g \in \arg \min_{x \in \mathcal{X}} g(x) \) exist. Then,

\[ f(x_g) \leq f(x_f) + 2 \sup_{x \in \mathcal{X}} |f(x) - g(x)|. \]
See Appendix F.1 for a proof of Lemma 1. Applying this idea to the AMoD setting, let \((x^*, w^*)\) be a solution to (2.3), and \((\hat{x}, \hat{w})\) a solution to (2.4). If \(
abla_{x,w} F(x, w) \preceq F_K(x, w) < \epsilon\) is small, then \((\hat{x}, \hat{w})\) will be at most \(2\epsilon\) worse than \((x^*, w^*)\) when evaluated by \(F\). It is then of interest to understand the conditions for which \(F_K\) will be uniformly close to \(F\). Since \(F_K\) is a random object, its error in estimating \(F\) has two contributors: stochastic error and model error. Specifically, the stochastic error is due to the error induced by estimating expectations under \(P\) using SAA, and the model error is the error incurred when estimating the true distribution \(P\) using \(\hat{P}\). For the analysis, we will need the following definition.

**Definition 1. Sub-exponential Random Variables**

A random vector \(X \in \mathbb{R}^d\) is sub-exponential with parameters \(\sigma^2, b < \infty\) if, for any \(v \in \mathbb{R}^d\) satisfying \(||v||_2 \leq b^{-1}\), the following inequality holds:

\[
\log \mathbb{E} \left[ e^{v^T(X - \mathbb{E}X)} \right] \leq \frac{||v||_2^2 \sigma^2}{2}.
\]

Intuitively, a random variable is sub-exponential if its tails decay at least as fast as that of an exponential random variable.

**Lemma 2 (Uniform Convergence for SAA).** Let \(P\) be the true distribution of customer demand, \(\hat{P}\) be the distribution of predicted customer demand and let \(P_{ijt}, \hat{P}_{ijt}\) be the distribution of \(\lambda_{ijt}\) under \(P, \hat{P}\) respectively. Assuming that \(\lambda \sim \hat{P}\) is \((\sigma^2, b)\) sub-exponential, then for any \(\delta > 0\), with probability \(1 - \delta\), the following holds:

\[
\max_{x,w} |F(x, w) - \hat{F}_K(x, w)| \leq \frac{2\sigma}{\sqrt{K}} \left( n^2T \log(m) + \log \frac{1}{\sqrt{\delta}} \right) + ||\chi(\hat{P}||P)||_2 \sqrt{\text{Var}_P(||\lambda||_2)},
\]

where \(\chi(\hat{P}||P) \in \mathbb{R}^{n^2T}_+, \chi(\hat{P}||P)_{ijt} = \chi(\hat{P}_{ijt}||P_{ijt})\) and \(\chi^2(\cdot||\cdot)\) represents the \(\chi^2\)-divergence between probability distributions which is non-negative and zero if and only if its arguments are equal.

See Appendix F.2 for a proof of Lemma 2. Note that the assumption of sub-exponential \(\lambda\) is not very restrictive. Indeed, many common distributions including gaussian, Poisson, chi-squared, exponential, geometric, and any bounded random variables are all sub-exponential [87]. If we denote
the solution to (2.3) as \((x^*, w^*)\) and the solution to (2.4) as \((\tilde{x}, \tilde{w})\), then applying lemmas 1 and 2, the following happens with probability at least \(1 - \delta\).

\[
\frac{1}{2} (F(\tilde{x}, \tilde{w}) - F(x^*, w^*)) \leq 2\sigma \sqrt{\frac{n^2T \log(m) + \log \frac{1}{\sqrt{\delta}}}{K}} + \frac{||\chi(\hat{P}|P)||_2 \sqrt{\text{Var}_P(||\lambda||_2)}}{\sqrt{\delta}}.
\]

Stochastic Error

Model Error

This result implies that, for a desired accuracy \(\epsilon > 0\), if we fit a generative model \(\hat{P}\) satisfying \(||\chi(\hat{P}|P)||_2 \leq 0.25\epsilon \text{Var}_P(||\lambda||_2)^{-1/2}\) and we use at least \(K \geq 64\epsilon^2 - 1/2\) samples for the SAA in (2.4), then the solution to (2.4) will, with probability at least \(1 - \delta\), be at most \(\epsilon\) worse than the optimal solution to (2.3) with known \(P\).

2.4.3 Computational Complexity

As shown in lemma 2, the sampling error of (2.4) is \(O(K^{-1/2})\), where \(K\) is the number of samples used to form the SAA objective. On the other hand, the computational complexity of (2.4) is an increasing function of \(K\), so in this section we discuss how the problem size of (2.4) depends on \(K\). A naive implementation of (2.4) would allocate \(Kn^2T\) decision variables for \(\{u^k_{ijt}\}_{k=1}^K\), and a linear dependence on \(K\) which would lead to scalability issues since integer programming is NP hard in the worst case. However, note that in an optimal solution, we will have \(u^k_{ijt} = (\lambda^k_{ijt} + w_{ijt} - x_{ijt})_+\). Thus if for some \(k, l\) we have \(\lambda^k_{ijt} = \lambda^l_{ijt}\), then the optimal solution has \(u^k_{ijt} = u^l_{ijt}\). In this case, solving (2.4) with the additional constraint of \(u^l_{ijt} = u^l_{ijt}\) will still yield the same optimal value while reducing the number of decision variables by one. Therefore, for each trip type \((i, j, t)\), instead of needing \(K\) decision variables \(\{u^k_{ijt}\}_{k=1}^K\), we only need \(c\) decision variables, where \(c\) is the number of unique values in the set \(\{\lambda^k_{ijt}\}_{k=1}^K\). The following lemma demonstrates the reduction in complexity achievable by this variable elimination procedure.

Lemma 3 (SAA Problem Size for Subexponential Demand). Assume that \(\lambda \sim \hat{P}\) is sub-exponential with parameters \(\sigma^2, b\). For any \(\delta > 0\), with probability at least \(1 - \delta\), the number of distinct realizations of the customer demand is no more than \(O \left( n^2T \min \left( \log \frac{Kn^2T}{\delta}, K \right) \right)\). Thus, as long as \(n^2T\) is not exponentially larger than \(K\), a variable elimination procedure ensures that the number of decision variables scales as \(O(\log K)\), as opposed to the linear scaling \(O(K)\) that the naive implementation would lead one to believe.

See Appendix F.3 for a proof of Lemma 3. Thus with high probability the number of decision variables will be logarithmic in \(K\), which is an exponential improvement over the linear dependence
that the naive implementation proposes. This is especially important since using large $K$ gives an objective function with less variance.

## 2.5 Scalable Integer Solutions via Totally Unimodular Linear Relaxations

Recall from (2.5) that increasing the number of samples $K$ used for Monte Carlo reduces the standard deviation of the random objective in (2.4), thereby increasing the quality of the algorithm’s output. While we showed that the number of decision variables is only logarithmic in the sample size $K$, the problem is still NP-hard. Thus, increasing the number of samples used in (2.4) may not be tractable in large scale settings. In this section, we propose a modified algorithm that solves a convex relaxation of (2.4), which is scalable to large problem sizes.

Our relaxation separately addresses the tasks of servicing existing customers and rebalancing vacant vehicles that are jointly solved in (2.4). Note that information about future customers can affect scheduling of waiting customers and vice versa in the optimal solution. In such a situation, servicing existing customers and rebalancing vacant vehicles with two separate algorithms prevents the sharing of information and can lead to suboptimal solutions. Nevertheless, this procedure runs in polynomial time, as opposed to integer programming. It is important to note, however, that solutions to convex relaxations of combinatorial problems need not be integral, and in this case naive rounding techniques can lead to violations of the network flow constraints. We obtain integer solutions by showing that our convex relaxations are totally unimodular linear programs. A linear program being totally unimodular means that it always has optimal solutions that are integer valued [2], and can thus be obtained using standard interior point optimization methods.

Network flow minimization problems are linear programs with constraints of the form (2.1), and preserve total unimodularity. However, in the case of problem (2.4), the inclusion of the constraints (2.2) break this totally unimodular structure, and hence solving a relaxation of (2.4) with the $x, w \in \mathbb{N}^{n_T}$ constraint removed is not guaranteed to return an integer solution. Alternatively, if we first assign vehicles to service existing customers, then the problem of rebalancing the empty vehicles no longer has constraints of type (2.2), and becomes totally unimodular. Inspired by this fact, in section 2.5.1 we discuss a bipartite matching algorithm we use to assign vacant vehicles to waiting customers, and in Section 2.5.2 we solve a totally unimodular version of (2.4) to determine rebalancing tasks.

### 2.5.1 Bipartite Matching for Servicing Waiting Customers

We use a bipartite matching algorithm to pick up waiting customers in a way that minimizes the total waiting time. Specifically, the current state of the system is $z, d \in \mathbb{N}^n$, where $z_i$ is the number
of vehicles currently available at station \( i \), and \( d_i \) is the number of outstanding customers at station \( i \). The decision variable is a vector \( x \in \mathbb{R}^{n^2} \) where \( x_{nt+j} \) is the number of vehicles sent from station \( i \) to station \( j \). Let \( A := 1_n^T \otimes I_n - I_n \otimes 1_n^T \), where \( 1_n \) is the vector of all 1’s in \( \mathbb{R}^d \), \( I_n \) is the identity matrix of size \( n \times n \) and \( \otimes \) is the matrix Kronecker product. Using this notation, the resulting state of taking action \( x \) in vehicle state \( z \) is simply \( z + Ax \). To satisfy the customers, we want \( Ax + z \geq y \) elementwise. If this is not possible, we will pay a cost of \( c_\lambda \) for every customer we do not pick up. To capture this, we define a drop vector \( u = (y - Ax - z)_+ \). The cost vector \( c \in \mathbb{R}^{n^2} \) is defined so that \( c_i + n_j \) is the travel time between \( i, j \). Thus, the optimal solution to the bipartite matching problem is obtained by solving the following linear program:

\[
\begin{align*}
\min_{x,u} & \quad c^T x + 1_n^T u \\
\text{subject to} & \quad u \geq 0 \\
& \quad u \geq y - (Ax + z) \\
& \quad x \in \mathbb{R}^{n^2}, u \in \mathbb{R}^n.
\end{align*}
\] (2.6)

It can be shown that bipartite matching has the totally unimodular property, and, therefore, will return integer solutions when the constraints are also integer.

### 2.5.2 Network Flow Optimization for Rebalancing Vehicles

To rebalance vacant vehicles in anticipation of future demand, we now solve (2.4) with \( w = 0 \) to obtain a rebalancing flow. We have \( w = 0 \) because the task of picking up outstanding customers is given to a bipartite matching algorithm, and hence does not need to be considered here. In this case, we can relax the integer constraints on \( x \) to obtain a totally unimodular linear program according to Lemma 4.

**Lemma 4** (Totally Unimodular SAA Rebalancing Problem). Consider the following convex relaxation of (2.4) where \( w = 0 \):

\[
\begin{align*}
\min_{x,w} & \quad c^T x + \frac{1}{K} \sum_{k=1}^{K} \sum_{ij} u_{ij}^k \\
\text{subject to} & \quad \sum_{j=1}^{n} x_{ij} - x_{ji}(t - \tau_{ji}) = s_{it} \text{ for all } i \in [n] \text{ and } t_0 < t \leq t_0 + T \\
& \quad u_{ij}^k \geq 0 \quad \forall k \in [K], i, j \in [n], t \in T, \\
& \quad u_{ij}^k \geq \lambda_{ij} - x_{ij} \quad \forall k \in [K], i, j \in [n], t \in T
\end{align*}
\] (2.7)
\{u_k\}_{k=1}^K, x \in \mathbb{R}^{n^2T}, \forall k \in [K], i, j \in [n], t \in T.

This problem is totally unimodular.

See Section F.4 for a proof.

Thus, in the setting where \( w = 0 \), the convex relaxation from (2.4) to (2.7) is tight in the sense that the solution to the latter is feasible and optimal for the former. For practical use the control strategy is to perform the tasks specified by the solutions to (2.6) and (2.7). The main strength of this approach is that both optimizers efficiently solve linear programs, as opposed to integer programs like (2.4) which can take orders of magnitude longer to solve in practice.

### 2.6 Stochastic Optimization for Model Predictive Control of AMoD Systems

When controlling an autonomous fleet of cars in real time, using a receding horizon framework allows the controller to take advantage of new information that is observed in the system. We propose a model predictive control approach to control AMoD systems online whereby a controller periodically issues commands obtained from solutions to optimization problems. Algorithm 1 outlines the details of the MPC controller for one timestep. Every \( \Delta t \) minutes, the controller queries the system to obtain information about the current state \( \{s_{it}\}_{i \in V, t \in T} \), the current number of waiting customers \( \lambda_0 \), and recent demand measurements \( \rho \). The controller then draws \( K \) samples from \( \hat{P}(\lambda|\rho) \) and uses those samples to solve a stochastic optimization problem. The solve mode \( I \) specifies if a solution results from integer programming (cf. 2.4.1) or linear programming (cf section 2.5). Specifically, if \( I = 1 \), the controller solves the integer program specified by (2.4), otherwise it solves the convex relaxation specified by (2.6) and (2.7). The controller executes the plan resulting from the optimization for the next \( \Delta t \) seconds after which it repeats this process with updated information.

#### 2.6.1 Algorithm Discussion

While total unimodularity is a convenient tool for getting tractable surrogates for integer programming, the conditions needed for it are fragile. Total unimodularity does not hold for several important extensions for AMoD control. [76] considers congestion effects and route rebalancing vehicles with traffic considerations in mind. [74] considers joint optimization between a fleet of electric vehicles with the smartgrid. In both of these extended models, total unimodularity does not hold, so these works rely on integer optimization. As more features are taken into consideration, using integer optimization will become more costly, thus in this section, we introduce a rounding scheme to produce integer solutions from convex relaxations of integer programs. In particular, we will be considering convex relaxations of (2.4). Consider the relaxed linear program obtained from (2.4) by removing
Algorithm 1: Model Predictive Control for AMoD systems using Stochastic Optimization

1. **Stochastic AMoD Control** \((\mathcal{I}, s, \lambda_0, \rho)\);
2. **Parameters:** Road Network \(G = (V, E)\), Conditional generative demand model \(\hat{P}\);
3. **Input:** Solve mode \(\mathcal{I}\), System state \(\{s_{it}\}_{i \in V, t \in T}\), Waiting customers \(\lambda_0\), recent demand \(\rho\).
4. **Output:** Control action \(x\).

3. Sample \(\{\lambda^k\}_{k=1}^K \overset{\text{i.i.d.}}{\sim} \hat{P}(\lambda|\rho)\);
4. if \(\mathcal{I} = 1\);
5. Obtain \(\{x_{saa}(t)\}_{t \in T}\) by solving (2.4) with samples \(\{\lambda^k\}_{k=1}^K\);
6. return \(x_{saa}(1)\);
7. else
8. Obtain \(\{x_{bm}(t)\}_{t \in T}\) by solving (2.6) for waiting customers \(\lambda_0\);
9. Obtain \(\{x_{saa}(t)\}_{t \in T}\) by solving (2.7) with samples \(\{\lambda^k\}_{k=1}^K\);
10. return \(x_{bm}(1), x_{saa}(1)\).

integral constraints on \(x, w\). Let \(x^*, w^* \in \mathbb{R}^{n \times T}\) be the solution to the above problem. It is then possible to construct transition matrices of an integer valued Markov process \(Y \in \mathbb{N}^n\) from \(x^*, w^*\) with the property that \(E[Y(t)] = x(t)\) for all \(t\) within the planning horizon, which can be used as a control decision. However, there are no estimates on the variance of \(Y\) or how it will affect performance.

2.7 Numerical Experiments

In this section we simulate the operation of an AMoD system where a controller implementing Algorithm 1 serves trip requests from several transportation datasets. Simulations in recent work [44] suggest that substantial improvements to operation cost and service quality are possible by using AMoD over current non-centralized implementations of ridesharing where drivers operate for their own self benefit. Our goal here is to present substantial improvements to their results, further highlighting the potential of AMoD. To this end, we evaluate the performance of Algorithm 1 in a MPC framework. We simulate the operation of an AMoD system servicing requests from the two different datasets and compare performance to recent state of the art algorithms. The AMoD system services trip requests in Hangzhou, China from a DiDi Chuxing ridesharing company dataset in the first experiment, and requests from the New York City Taxi and Limousine Commission dataset in the second experiment.

2.7.1 Scenarios

For Hangzhou, we leveraged a dataset provided by the Chinese ridesharing company Didi Chuxing. The dataset contains all trips requested by users from January 1 to January 21, 2016, resulting in a total of around eight million trips. The dataset separates Hangzhou into 793 discretized regions.
However, the dataset contains only trips that started in a core subset consisting of 66 regions. For simplicity, we disregard trips that do not start and end in this core subset (approximately one million trips). For each trip, the records provide origin region, destination region, a unique customer ID, a unique driver ID, the start timestamp and the price paid. The dataset contains neither geographic information about the location of the individual districts, nor information on the duration of the trip. Thus, we used RideGuru [73] to estimate the travel time of each trip from the trip price, which in turn allowed us to infer average travel times between regions. For the simulation, we used the first 15 days to train the forecasting model, and the last day to test in simulation by “playing back” the historical demand.

The second scenario is based on the well-known New York City Taxi and Limousine Commission dataset[1]. It contains, among others, all yellow cab taxi trips from 2009 to 2018 in New York City. For each trip, the start and end coordinates and timestamps are provided. For our simulation, we looked only into the trips that started and ended in Manhattan. Additionally, we partitioned the island into 50 regions. We used the trips between December 22, 2011 and February 29, 2012 to train the forecasting model, and used the evening rush hour (18:00-20:00) of March 1, 2012 for testing in simulation.

2.7.2 Experimental Design

For each scenario, we simulate the operation of an AMoD system by “playing back” the historical demand at a 6 second resolution. That is, vehicle and customer states get updated in 6 second timesteps. If on arrival, a customer arrives to a region where there is an available vehicle, the customer is assigned to that vehicle. Otherwise, the customer joins the region’s customer queue. A customer’s trip duration corresponds to the travel time recorded in the dataset. However, vehicle speeds are such that travel time between any two region centroids corresponds to the average travel time between those respective regions in the dataset.

Every $\Delta t = 5$ minutes, the simulation invokes an AMoD controller. The controller returns the rebalancing tasks for each region. These tasks, in turn, are assigned to idle vehicles as they become available. After $\Delta t$ minutes, unused tasks are discarded, and the controller is invoked again. We tested the following controllers:

- **Reactive** is a time-invariant reactive controller presented in [69] which rebalances vehicles in order to track uniform vehicle availability at all stations.

- **MPC-LSTM-MILP** is the model predictive controller presented in [44] which relies on point forecasts and mixed integer linear programming.

- **MPC-LSTM-LP** is a relaxation of the MPC-LSTM-MILP controller attained by the ideas described in Section 2.5 by running two linear programs.

• **MPC-LSTM-SAA** is the controller implementing Algorithm 1 with \( I = 0 \) and \( K = 100 \) samples.

• **MPC-Perfect** is a non-causal golden standard where the MPC controller is given perfect forecasts instead of samples of predicted demand.

All MPC controllers are using a planning horizon of 4 hours. For the coefficients \( c_x, c_w, c_\lambda \), we chose \( c_x \ll c_w \) and \( c_w = c_\lambda / T \) so that the controller prioritizes customer satisfaction.

### 2.7.3 Forecasting

In these experiments, the generative model \( \hat{P} \) for Algorithm 1 first estimates the mean of the future demand using a Long Short Term Memory (LSTM) neural network. The LSTM networks were trained on a subset of the data that does not include the test day. We trained a different network for each of the scenarios. Specifically, the LSTM takes as input the past 4 hours of observed customer demand and then predicts the expected demand for the next 2 hours. We assume that the conditional distribution of demand given the LSTM output is Poisson. Moreover, to account for model uncertainty, we sample from the LSTM with dropout. To generate the forecasts given recent values of demand \( \lambda_0 \), we draw \( K \) samples \( \{\lambda^K_k\}_{k=1}^K \) i.i.d. \( \sim \hat{P}(\cdot|\lambda_0) \) from the LSTM using dropout, and then sample the demand predictions from a Poisson process whose mean is specified by \( \bar{\lambda} \) so that \( \lambda^K_k \sim \text{Poisson}(\bar{\lambda}^k) \). For these experiments, we used \( K = 100 \) because the benefits of more samples was no longer worth the increased computation needed.

### 2.7.4 Results

In the Hangzhou scenario, the MPC-LSTM-SAA controller, based on [1], greatly outperforms the other controllers: it provided a 62.3% reduction in mean customer waiting time over the MPC-LSTM-MILP controller from [44], and a 96.7% reduction from Reactive (see Table 2.1). Qualitatively, Figure 2.4 shows how the MPC-LSTM-SAA controller shows the greatest improvement over MPC-LSTM-MILP in times of the day where there is relatively high variance (in day-to-day travel demand variation). This suggests that the proposed algorithm’s rebalancing strategy is better at handling future demand with high uncertainty than prior work. Naturally, handling uncertainty requires being prepared for a large variety of demand realizations. Thus, it is not unexpected that, as seen in Figure 2.4 and Table 2.1, MPC-LSTM-SAA rebalances slightly more than MPC-LSTM-MILP; nonetheless, it still issues less rebalancing tasks than Reactive.

Moreover, the performance of the MPC-LSTM-MILP and MPC-LSTM-LP controllers are essentially the same, which suggests that the relaxations described in 2.5 yield reliable runtimes without significantly sacrificing performance quality.

The New York City scenario also demonstrates benefits of using stochastic optimization in the control. Table 2.2 summarizes the results of this case study. While the 99 percentile wait time for the deterministic algorithm MPC-LSTM-LP is 32 percent smaller than that of Reactive, its mean waiting
CHAPTER 2. FLEET CONTROL FOR ON-DEMAND RIDEHAILING SYSTEMS

Figure 2.4: The top plot shows the number of waiting customers as a function of time for both our controller (MPC-LSTM-SAA) in blue and the controller from [44] in green. Under the operation of our controller, there are significantly fewer waiting customers throughout the day, which is reflected by the waiting times. Unexpectedly, as a price of this improved service, our controller issues more rebalancing requests, since it is planning for many outcomes, as shown by the middle plot. Looking at the middle and bottom plots together, we see that our controller does additional rebalancing precisely when there is significant variance in the demand.

Table 2.1: Wait times for the DiDi scenario (seconds).

<table>
<thead>
<tr>
<th>Wait Times:</th>
<th>Mean</th>
<th>Median</th>
<th>99 Percentile</th>
<th>Reb. Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>276.284</td>
<td>72.0</td>
<td>1890.0</td>
<td>139927</td>
</tr>
<tr>
<td>MPC-LSTM-MILP</td>
<td>24.149</td>
<td>0.0</td>
<td>582.0</td>
<td>40097</td>
</tr>
<tr>
<td>MPC-LSTM-LP</td>
<td>23.305</td>
<td>0.0</td>
<td>558.0</td>
<td>39687</td>
</tr>
<tr>
<td>MPC-LSTM-SAA</td>
<td>9.0799</td>
<td>0.0</td>
<td>264.0</td>
<td>68150</td>
</tr>
<tr>
<td>MPC-Perfect</td>
<td>5.527</td>
<td>0.0</td>
<td>108.0</td>
<td>32950</td>
</tr>
</tbody>
</table>

time is larger by 16 percent. Leveraging stochastic optimization, MPC-LSTM-SAA further improves the 99 percentile wait time of MPC-LSTM-LP by 17 percent and offers a 22 percent reduction in mean waiting time over Reactive. In summary, MPC-LSTM-SAA outperforms both Reactive and MPC-LSTM-LP in both mean and 99 percentile wait times. As a tradeoff, both MPC-LSTM-LP and MPC-LSTM-SAA issue more rebalancing tasks than Reactive.

Table 2.2: Wait times for the NYC scenario. (seconds)

<table>
<thead>
<tr>
<th>Wait Times:</th>
<th>Mean</th>
<th>Median</th>
<th>99 Percentile</th>
<th>Reb. Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>19.15</td>
<td>0.0</td>
<td>732.0</td>
<td>7196</td>
</tr>
<tr>
<td>MPC-LSTM-LP</td>
<td>22.70</td>
<td>0.0</td>
<td>504.0</td>
<td>7907</td>
</tr>
<tr>
<td>MPC-LSTM-SAA</td>
<td>15.07</td>
<td>0.0</td>
<td>420.0</td>
<td>10952</td>
</tr>
<tr>
<td>MPC-Perfect</td>
<td>10.8</td>
<td>0.0</td>
<td>384.0</td>
<td>8356</td>
</tr>
</tbody>
</table>
Figure 2.5: The top plot shows the number of waiting customers as a function of time for several controllers. As expected, compared to the reactive controller, the predictive controllers have more customers wait at the beginning of the simulation in order to better prepare for the customers appearing later. Leveraging stochastic optimization, MPC-LSTM-SAA outperforms MPC-LSTM-LP and Reactive in terms of mean waiting time. As a tradeoff, the bottom plot shows that the reactive controller issues the least amount of rebalancing tasks, while MPC-LSTM-SAA issues the most.

2.8 Discussion

In this chapter we presented an algorithm to rebalance vacant service vehicles to better align with future demand in ridehailing systems. The algorithm is comprised of two components, the first being a demand forecasting model that predicts a distribution for future demand. The second component is a stochastic receding horizon controller that balances the tasks of serving existing customers and rebalancing the fleet to prepare for future customers. The experimental results show that using distributional estimates for future demand leads to more robust behavior than using point estimates.

One important area for future work is in better models for congestion effects which still lead to tractable optimization problems. Existing works employ very conservative hard constraints on edge capacities to ensure that the resulting network flow optimization problem remains convex \[100\]. In such cases it is not clear how to convert a fractional solution into an integer solution in a principled way that preserves the objective value, though in practice simple rounding procedures are used. More realistic congestion models such as the cell transmission model \[20\] or link transmission model \[95\] lead to non-convex optimization problems, and so one important future direction is how such transmission models can be approximated in a way that the resulting network flow problem is efficiently solvable.

Rebalancing aims to improve the vehicle supply in the proximity of future customers’ pickup locations, and consequentially reduce customer waiting time. A nearly orthogonal way to improve vehicle supply in ridehailing systems is the use of ridesharing where a single vehicle can carry more than one customer, as opposed to the model we studied in this chapter where service vehicles can carry at most one customer. Leveraging higher vehicle capacity would increase system throughput without needing to increase the fleet size, provided that the customer trips can be effectively pooled.
In the next section we describe how the algorithm presented in this section can be augmented to support 2-capacity ridesharing whereby service vehicles can carry up to 2 vehicles at a time.
Chapter 3

Fleet Control for On-Demand Ridesharing Systems

In this chapter we generalize the methodology from Section 2 to include ridesharing services. To this end, we present a model predictive control (MPC) algorithm to optimize vehicle routes for 2-capacity Ridesharing Autonomous Mobility-on-Demand (RAMoD) systems, whereby self-driving vehicles provide coordinated on-demand mobility, possibly allowing two customers to share a ride. We evaluate the proposed algorithm in a case study for the city of San Francisco, CA, by using the microscopic traffic simulator MATSim. The simulation results show that a RAMoD system can significantly improve social welfare with respect to a single-occupancy Autonomous Mobility-on-Demand (AMoD) system, and that the predictive structure of the proposed MPC controller allows it to outperform existing reactive ridesharing coordination algorithms for RAMoD.

3.1 Introduction

AMoD systems have the potential to reduce the social cost for mobility [80], but existing works do not fully study the endogenous effects that AMoD systems could have on traffic congestion or on induced demand. While some works study congestion-aware routing algorithms for AMoD systems [76], there are many social and economic factors that must be thoroughly explored in order to accurately estimate the impact that AMoD systems will have on society. In this chapter we present techniques to incorporate ridesharing into AMoD systems. Ridesharing enables higher vehicle utilization by having a service vehicle carry multiple customers at a time, and could potentially enable an AMoD service to reduce its fleet size without impeding service quality.
CHAPTER 3. FLEET CONTROL FOR ON-DEMAND Ridesharing SYSTEMS

3.1.1 Statement of Contributions

In this chapter we present a MPC algorithm for RAMoD accounting for present and future travel demand. Specifically, the contribution of this chapter is threefold: First, we develop a multi-commodity network flow model capturing the operations of the double-occupancy RAMoD system shown in Fig. 3.1. Second, we devise a MPC algorithm assigning multiple customers to vehicles, designing vehicle routes and rebalancing empty vehicles to anticipate future requests, with the goal to maximize social welfare, namely, a weighted combination of customers’ travel time and vehicles’ mileage. Third, we evaluate the performance of our algorithm against state-of-the-art unit capacity AMoD approaches as well as reactive ridesharing algorithms. Our results show that a RAMoD system can significantly reduce overall costs with respect to a single-occupancy AMoD system, and that its predictive structure allows it to outperform existing reactive ridesharing algorithms.

3.1.2 Organization

The remainder of this chapter is structured as follows: Section 3.3 introduces the multi-commodity flow optimization model for RAMoD. The design and details of the RAMoD MPC algorithm is discussed in Section 3.4. Section 3.5 presents a real-world case-study for San Francisco, CA, where we test our approach and compare it with the state-of-the-art. We conclude the chapter in Section 3.6 with a summary and a discussion on future research.

3.2 Related Work

Our contribution pertains to two main research fields: i) AMoD systems and ii) ridesharing. There are several approaches to analyze and control AMoD systems, such as simulation models [41, 62, 67],
queuing-theoretical models [97, 98] and multi-commodity network flow models [80, 69]. Simulation models describe transportation systems with high precision, but are not amenable to optimization. Queuing-theoretical models capture the stochasticity of the customer requests and can be used for control synthesis, but it is extremely hard to represent external constraints in this framework. Multi-commodity network flow models can be efficiently implemented in optimization frameworks, while still being very expressive and compatible with a variety of constraints. Consequently, they have been applied to a number of problems: from the control of AMoD systems in congested road networks [100, 76], in coordination with the power network [75], and to the design of MPC algorithms [83].

The ridesharing problem has been studied in static and dynamic environments, and with the objective of minimizing the mileage driven and the average travel time, or maximizing the number of customers served [1]. There are a number of contributions to the static ridesharing problem, where all the requests are assumed to be known in advance. A study for the single-driver-single-rider setting is presented in [6]. The dynamic version of the ridesharing problem captures new riders and drivers continuously entering and leaving the system. Ridesharing features exist today in mobility services like Lyft and Uber, and have been studied in [78, 5]. These works, along with ridesharing systems by Lyft and Uber are primarily reactive in the sense that they only consider serving the current demand. Considering the substantial performance gains presented by [83, 44] for single occupancy mobility networks, a natural question is whether ridesharing systems can experience similar benefits by anticipating future customer demand.

3.3 Flow Optimization Model for Double-Occupancy RAMoD

In this section, we present a graphical model representation of road networks. We then pose the problem of coordinating a fleet of vehicles for mobility service with ride-sharing in road networks as an optimization problem. All service vehicles in this model are double-occupancy, i.e., they can carry up to two passengers at any given time, as shown in Fig. 3.1.

3.3.1 Modeling the Road Network

We model the transportation network as a spatio-temporal graph. The road network is partitioned into $n$ stations, which are spatially disjoint regions where customers can request rides to and from. The nodes of the spatio-temporal graph are then $V = [n] \times \mathbb{T}$ so that a node $(i, t) \in V$ specifies a physical location $i$ and a time $t$. We measure time in discrete intervals of $\Delta t$ so that $\mathbb{T} = \Delta t \cdot \mathbb{N}$. Note that the size and number of stations dictate the resolution of the graph: Having small stations increases the granularity, but, as a trade-off, more stations are needed to cover the entire road network.

Each road in the network has a nominal driving speed which may depend on the level of exogenous traffic, and a fixed length. We model congestion as an exogenous influence on the nominal speeds of
roads, but do not model the endogenous congestion effects induced by RAMoD vehicles. Therefore, there is no limit to the number of RAMoD vehicles that can be traveling at the nominal speed on a road at any given time. The travel time to traverse a road is given by its length divided by its nominal speed.

Paths are defined as an ordered sequence of roads, and the travel time of a path is simply the sum of the travel time of its roads. Then, for a given level of exogenous traffic, we denote the time needed to travel from station \(i\) to station \(j\) taking the fastest path as \(\tau_{ij}\). Directed edges in this graph correspond to paths in the road network. For any two nodes \((i, t_1)\), \((j, t_2)\), a directed edge from \((i, t_1)\) to \((j, t_2)\) exists if \(\tau_{ij} = t_2 - t_1\). Because we do not consider endogenous congestion effects from RAMoD vehicles, we only need to consider shortest paths when routing vehicles.

To allow cars to idle, we also include edges from \((i, t)\) to \((i, t + \Delta t)\) for each \(i \in [n], t \in T\). Defining \(E\) to be the set of all such edges, our weighted graph representation of the road network is \(G := (V, E)\) where the weight along an edge is its corresponding travel time. The RAMoD system is endowed with \(M\) self-driving cars whose position can be at stations if they are idling, or on edges if they are in transit. We denote the transportation demand for a set of times \(T \subset T\) as \(\Lambda\), which is a \(n \times n \times |T|\) array, so that \(\Lambda_T(i, j, t)\) is the number of customers that will request a trip from a location in station \(i\) to a location in station \(j\) at time \(t \in T\). We will use the shorthand notation \(\lambda_{ijt} := \Lambda_T(i, j, t)\) when the time set \(T\) is unambiguous.

### 3.3.2 Integer Network Flow Model for RAMoD Systems

We leverage network flow models and Integer Linear Programming (ILP) to devise an algorithm for controlling the double-occupancy RAMoD fleet shown in Fig. 3.1. In accordance with the model presented in Section 3.3.1, this involves specifying actions for all vehicles in the fleet for a horizon of \(T\) time-steps, where each time-step is \(\Delta t\) minutes. At time \(t_0\), the planning horizon \(T(t_0) \subset T\) is then

\[
T(t_0) := \{t_1, t_2, \ldots, t_T\}
= \{t_0 + \Delta t, t_0 + 2 \cdot \Delta t, \ldots, t_0 + T \cdot \Delta t\},
\]

so that \(t_k := t_0 + k \cdot \Delta t\) for \(k \in [T]\).

In this setting, the control algorithm needs to provide instructions to empty vehicles as well as partially occupied vehicles. Since vehicles with different occupancy levels have varying levels of vacancy and commitments to customer destinations, a controller needs to treat different types of vehicles accordingly. Using the graph representation of the road network introduced in Section 3.3.1, we represent the distinction between cars with different occupancy levels by introducing the following decision variables:

- \(r \in \mathbb{N}^{n^2\cdot T}\), where \(r_{ijt}\) represents the number of completely empty vehicles traveling from station
$i \rightarrow j$ at time $t$;

- $\{x(m)\}_{m=1}^{n} \in \mathbb{N}^{nT}$, where $x_{ijt}(m)$ represents the number of vehicles with exactly one passenger whose destination is station $m$, traveling from station $i \rightarrow j$ at time $t$;

- $p \in \mathbb{N}^{nT}$ where $p_{ijkt}$ is the number of cars at station $i$ at time $t$ with two passengers with destinations $j$ and $k$, respectively.

Moreover, in our model, RAMoD vehicles interact with customers at stations in two phases:

- **Phase 1**: When a vehicle arrives at a station, it first delivers any customers onboard whose destination is in this region.

- **Phase 2**: After delivering customers, the vehicle has the option to pick up new customers before leaving the station.

Additionally, we assume that cars at maximum capacity are not controllable: They will drive directly from their current position to their first destination and cannot be assigned a different task until they drop off their first customer. Thus a car of type $p_{ijkt}$ must drive from station $i$ to station $j$.

We use $s_r \in \mathbb{N}^n T$, $s_x \in \mathbb{N}^{n^2 T}$ to encode the current state of RAMoD vehicles in the system so that:

- $s_{r, it}$ specifies how many currently busy vehicles will become available as vacant cars in station $i$ at time $t$;

- $s_{x, it}(m)$ specifies the number of currently busy cars that will become available as partially occupied cars in station $i$ at time $t$ whose on-board customers’ destination is in station $m$.

These variables, however, are not enough to specify an actionable strategy. For example, suppose an algorithm computes $x_{1, 2, t}(2) = 1$, meaning that a car carrying one passenger should embark from station 1 to station 2 at time $t$. This does not tell us, however, whether the car should have picked up a customer, dropped off a customer, or done nothing before leaving station 1. To address these kinds of ambiguities, we introduce book-keeping variables $x^{(zo)}, x^{(so)}, p^{(zo)}, p^{(so)}$ where the superscripts $(zo), (so)$ indicate that the car had zero occupants or a single occupant after finishing phase 1. Now if we have $x^{(zo)}_{1, 2, t}(2) = 1$, then we know that the car at station 1 should pick up a customer before leaving for station 2, since the car had zero occupants after phase 1. On the other hand, if we see $x^{(so)}_{1, 2, t}(2) = 1$, then the car should not pick up a customer before leaving the station, since it has one passenger after phase 1, and leaves with one passenger. Naturally, $x$ and $p$ are the sum of their zero and single occupant components which we enforce using the following constraints.

$$x_{ijt}(m) = x^{(zo)}_{ijt}(m) + x^{(so)}_{ijt}(m) \quad \forall i, j, m \in [n], t \in T(t_0) \quad (3.1)$$
CHAPTER 3. FLEET CONTROL FOR ON-DEMAND RIDE-SHARING SYSTEMS

\[ p_{ijk} = p_{ijk}^{(zo)} + p_{ijk}^{(so)} \quad \forall i, j, k \in [n], t \in \mathcal{T}(t_0) \quad (3.2) \]

Additionally, constraints (3.3) and (3.4) enforce that phase 1 (dropping off current customers) must occur before phase 2 (picking up new customers and leaving the station):

\[ x_{ijt}^{(so)}(i) = 0 \quad \forall i, j \in [n], t \in \mathcal{T}(t_0) \quad (3.3) \]
\[ p_{ijt}^{(so)} = 0 \quad \forall i, j \in [n], t \in \mathcal{T}(t_0) \quad (3.4) \]

We now present the physical constraints that any actionable strategy for the planning horizon $\mathcal{T}(t_0)$ must satisfy:

\[ s_{r,it} + \sum_{j=1}^{n} (r_{jit} + x_{jit}(i) + p_{jit}) = \sum_{j=1}^{n} r_{jit} + \sum_{m=1}^{n} x_{ijt}^{(so)}(m) + \sum_{u=1}^{n} p_{ijt}^{(so)} \quad (3.5) \]

where $t_{jt} = t - \tau_{ji}, \forall i \in [n], t \in \mathcal{T}(t_0)$.

Constraint (3.5) specifies the options available to empty vehicles. Specifically, the left side of (3.5) counts the number of cars that will have zero occupants after arriving at station $i$ at time $t$ and finishing phase 1. These cars can either remain empty, pick up one passenger, or pick up two passengers before leaving the station, corresponding to the terms on the right hand side of (3.5).

\[ s_{x,imt}(m) + \sum_{j=1}^{n} (x_{ji(t-\tau_{ji})}(m) + p_{jim(t-\tau_{ji})}) = \sum_{j=1}^{n} p_{imjt}^{(so)} + x_{ijt}^{(so)}(m) \quad (3.6) \]
\[ \forall i, m \in [n], m \neq i, t \in \mathcal{T}(t_0). \]

Constraint (3.6) specifies the options available to single occupant vehicles. Specifically, the left side of (3.6) counts the number of cars that will have one occupant whose destination is $m$ after arriving at station $i$ at time $t$ and finishing phase 1. These cars can leave with or without picking up another passenger, corresponding to the first and second terms on the right side of (3.6) respectively. Interactions with customers are captured by the following two constraints:

\[ a_{ijt} = \left( \sum_{u=1}^{n} x_{iut}^{(zo)}(j) + p_{iujt}^{(so)} + 1_{j \neq u} \right) \left( p_{iujt}^{(zo)} + p_{ijut}^{(zo)} \right) + 2p_{ijt}^{(zo)} \quad (3.7) \]
\[ \forall i, j \in [n], t \in \mathcal{T}(t_0) \]
CHAPTER 3. FLEET CONTROL FOR ON-DEMAND RIDESHARING SYSTEMS

\[ d_{ijt} = \sum_{\tau=1}^{t} \lambda_{ij\tau} - a_{ij\tau} \] (3.8)

\[ \forall i, j \in [n], t \in T(t_0) \]

The variable \( a \) in (3.7) counts the number of trips from station \( i \) to \( j \) that are serviced at time \( t \). This is because the right side of (3.7) includes all the ways such customers can be picked up. The variable \( d \) in (3.8) counts the number of customers that are waiting for each trip type and time. This is because the right side of (3.8) is the total demand from \( i \) to \( j \) up to time \( t \) minus the number of those customers that have been served up until time \( t \). Finally, because we cannot have fractional vehicles on the road, all of the decision variables must be integer.

\[ r \in \mathbb{N}^{n^2T}, \{x^{(zo)}, x^{(so)}(m)\}_{m=1}^{n} \in \mathbb{N}^{n^2T}, p^{(zo)}, p^{(so)} \in \mathbb{N}^{n^3T}. \] (3.9)

The goal of a RAMoD algorithm is to maximize social welfare. Specifically, we aim at minimizing a weighted combination of total travel time and operational costs represented by

\[ J(r, x, p, d) = V_d \cdot \sum_{i,j=1}^{n} \sum_{t \in T(t_0)} d_{ijt} \] (3.10)

\[ + \sum_{i,j=1}^{n} \sum_{t \in T(t_0)} \tau_{ij} \cdot \left( V_r \cdot r_{ijt} + \sum_{m=1}^{n} V_x \cdot x_{ijt}(m) + V_p \cdot p_{ijmt} \right), \]

where \( V_r, V_x, V_p, V_d \) are tunable coefficients representing the per unit time cost of rebalancing, driving customer carrying vehicles, and delaying customer pickup respectively. The first term of \( J \) measures service quality for customers and the last term represents operation cost of the system.

Our RAMoD algorithm for servicing transportation demands implements the strategy obtained by solving the following integer linear program

\[ \min_{r, x^{(zo)}, x^{(so)}, p^{(zo)}, p^{(so)}} J(r, x, p, d) \] (3.11)

s.t. (3.1) \(-\) (3.9).

3.3.3 Model Discussion

A few comments are in order. Partitioning the road network into stations is a general approach that encapsulates models with a wide range of focuses as special cases including, but not limited to, temporal resolution, spatial resolution, and different modes of transportation [76, 44, 83]. We choose to study double-occupancy fleets because most vehicles have at least two passenger seats that can be utilized for ride-sharing. We avoid studying higher occupancy models due to computational
3.4 A Real-Time RAMoD MPC Algorithm

In this section we use the integer network flow framework discussed in Section 3.3.2 to derive a real-time MPC algorithm for RAMoD that leverages demand forecasts to improve service quality.

3.4.1 Real-Time RAMoD MPC Algorithm

To extend the integer linear programming approach from Section 3.3.2 to an online MPC setting, we implement it in a receding horizon framework. Specifically, every \( \Delta t \) minutes, we collect the system state \( \{s_r, s_x\} \), currently waiting customers \( \Lambda_{(t_0)} \), and travel times \( \{\tau_{ij}\}_{i,j \in [n]} \) computed from the road congestion levels at time \( t_0 \) as input to the ILP. The ILP uses a planning horizon of \( \mathcal{T}(t_0) \) with \( T \) time-steps where \( t_0 \) is the current time. Since the true demand in our planning horizon \( \Lambda_{\mathcal{T}(t_0)} \) is not known, we use a forecaster \( \hat{\Lambda} : \mathbb{T} \rightarrow \mathbb{N}^{n \times n \times T} \) instead so that \( \hat{\Lambda}(t_0) \) is an estimate of \( \Lambda_{\mathcal{T}(t_0)} \).

After executing the first step of the control strategy resulting from the ILP, we update the system state and re-solve the ILP in a receding horizon manner. In general, ILPs are NP-hard. Thus, there is no guarantee that this method will scale to large problem instances. However, as only the first step of the resulting strategy is implemented before the algorithm updates the system state and recomputes, only the decision variables at the first time-step \( t_1 \) need to be integer in order for the algorithm to be actionable. Thus, we can relax the integer constraint in (3.9) on all variables at times \( t_2, t_3, ..., t_T \) to reduce the complexity of the problem while still having an actionable algorithm. With this heuristic in place, the number of integer constrained variables no longer depends on the planning horizon \( T \), allowing for a larger planning horizon to be employed. This relaxation leads to the following mixed integer linear program (MILP):

\[
\begin{align*}
\text{min.} & \quad J(r, x, p, d) \\
\text{s.t.} & \quad (3.1) - (3.8) \\
& \quad r_{ijt_1} \in \mathbb{N} \forall i, j \in [n] \\
& \quad \{x_{ijt_1}^{(zo)}(m)\}_m, \{x_{ijt_1}^{(so)}(m)\}_m, p_{ijmt_1}^{(zo)}, p_{ijmt_1}^{(so)} \in \mathbb{N} \\
& \quad \forall i, j, m \in [n] \\
& \quad r, \{x^{(zo)}(m)\}_m, \{x^{(so)}(m)\}_m, p^{(zo)}, p^{(so)} \succeq 0 \forall m
\end{align*}
\]

In practice, the forecast \( \hat{\Lambda}(t_0) \) will not be a perfect estimate of \( \Lambda_{\mathcal{T}(t_0)} \). An incorrect forecast can cause an algorithm to send a vehicle to a location to pick up a customer when in fact no customer will show up. Another important situation is if the forecaster underestimates demand and dispatches too few vehicles, leading to unserved customers. The first scenario causes fuel inefficiency and the
second degrades service quality. For these reasons it is important for any predictive approach to have robustness to such inaccuracies. To accomplish this, we implement a matching algorithm that can be used to pickup the aforementioned unserved customers with vehicles that are told to idle by the solution of (3.12). Specifically, we choose a reserve fleet size $M_0 < M$ so that, if the solution to (3.12) assigns idle tasks to $M_0 + M_{\text{extra}}$ vehicles, then the matching algorithm is allowed to use $M_{\text{extra}}$ cars to serve waiting customers. In this way, we have a reactive component to serve unexpected demand that will only act if it can guarantee that there will be $M_0$ vehicles available to the controller (3.12) for the next time-step. The purpose of the reserve fleet size $M_0$ is to prevent the matching algorithm from implementing a completely greedy approach, and ultimately striking a balance between prediction and reaction when the forecast is not perfect. Algorithm 2 describes the full receding-horizon algorithm for RAMoD.

### Algorithm 2: Ride-Sharing Autonomous Mobility on Demand

```plaintext
1. **RAMoD** $(\mathcal{G}, T, \hat{\Lambda}, T, M_0)$;

   **Input**: Graph rep. of road network $\mathcal{G} = (V, E)$, operation horizon $T$, number of timesteps
   in the planning horizon $T$, forecaster $\hat{\Lambda}$ and reserve fleet size $M_0$.

   **Output**: Control actions $r, \{x^{(zo)}\}_m, \{x^{(so)}\}_m, p^{(zo)}, x^{(so)}$.

2. for $t_0 \in T$
   3. Collect the vehicle state of the system $s_r, s_x$;
   4. Collect the current demand $\Lambda(t_0)$;
   5. Obtain forecast for the next $T$ timesteps $\hat{\Lambda}(t_0)$;
   6. Solve (3.12) to obtain $r, \{x^{(zo)}\}_m, \{x^{(so)}\}_m, p^{(zo)}, p^{(so)}$;
   7. Implement the $r, \{x^{(zo)}\}_m, \{x^{(so)}\}_m, p^{(zo)}, p^{(so)}$ instructions for the first timestep;
   8. if There are more than $M_0$ idling cars;
   9. then
   10. Pair idling cars with nearby customers until either there are only $M_0$ idling cars or no waiting customers.
```

### 3.4.2 Algorithm Discussion

A few comments are in order. First, by representing the vehicle actions as flows between locations, the number of decision variables in (3.12) does not depend on the number of vehicles or the number of customers, enabling the algorithm to operate effectively in highly populated areas. This is contrary to recent approaches presented in [78, 5] where the problem size increases with the number of cars and the number of customers. Second, the relaxation from (3.11) to (3.12) allows modern branch and bound solvers to find solutions in under one minute, allowing Algorithm 2 to be run in real-time. Third, Algorithm 2 is able to leverage forecasts of future demand into its strategy. Finally by updating the travel times using current congestion values when solving (3.12), we can model congestion effects in a dynamic and time varying manner, despite not directly considering endogenous contributions from the fleet.
3.5 Numerical Experiments

In this section, we present numerical experiments to benchmark the performance of Algorithm\(^2\) when servicing trips from transportation datasets in simulation. The code written for experiments in \([78, 5]\) is protected by copyright, so we attempted to capture its characteristics in our own implementation of a reactive ride-sharing algorithm which we refer to as RAMoD-Reactive. The performance of Algorithm\(^2\) is compared to the real-time MPC algorithm for AMoD in \([44]\), RAMoD-Reactive, and to an existing, high-performing rebalancing heuristic \([68]\).

3.5.1 Scenario

We focus on the transportation network of San Francisco, CA. The Fig. 3.2 shows a sequence of the simulation with the San Francisco map. The study uses a publicly available dataset of taxi traces recorded in the city of San Francisco \([71]\). The traces were recorded between May 17, at 03:00:04 and June 10, 2008, 02:25:34. The dataset contains a total of 464'045 trips for the entire period. We use 15 days from May 20 up to June 9, where the weekends are not included, to fit our model data, and May 19 for the evaluation. The model we fit is for the forecast of customer demand needed by the RAMoD and the AMoD MPC algorithms.

3.5.2 Simulation Environment

We use the AMoDeus \([42]\) simulator to validate the algorithms. It is an open-source simulator to analyze and validate algorithms for mobility-on-demand (MoD) systems. Internally, it uses the agent-based transportation simulator MATSim \([43]\), which includes well-tested high-fidelity simulation of
road dynamics. The simulator is able to represent large MoD systems with unreduced fleet sizes and to compare them directly to existing benchmark algorithms, e.g., [68]. The simulator contains an inner loop to model road network dynamics as well as an outer loop to take into account varying or dynamic demand which may change as a function of the network dynamics. For this validation, we assumed a static demand profile, i.e., we assume that independent of cost and performance of our implemented RAMoD system, the stochastic user equilibrium [92] has been reached and is invariant.

3.5.3 Experimental Design

We simulate the scenario described in Section 3.5.1 testing the following four controllers:

- **RAMoD-MPC**: The controller implementing Algorithm 2.
- **AMoD-MPC**: The controller described in [44] that also leverages short-term forecasts in a MPC manner, but does not have a ride-sharing feature.
- **RAMoD-Reactive**: The RAMoD-MPC algorithm modified to emulate a similar behavior to the methods presented in [78, 5]. We do this by making two modifications: First, we provide no forecast to make the algorithm reactive. Second, we eliminate vehicle paths with detours that cause significant inconveniences to customers, corresponding to the sharability graphs in [78, 5].
- **AMoD-Reactive**: The controller described in [68].

A fleet size of $M = 400$ vehicles is provided to all controllers. The RAMoD-MPC, AMoD-MPC and RAMoD-Reactive controllers use a time horizon of 150 minutes broken up into $T = 10$ time steps, each of length $\Delta t = 15$ minutes. The number of stations used for these controllers is 25. The AMoD-Reactive controller uses 10 stations instead of 25 because using 10 stations leads to better performance. RAMoD-MPC uses a reserve fleet size of $M_0 = 100$. The network is split into stations using a k-means partitioning method on the request origin locations. For the MPC schemes with time horizon of $T = 10$, every 15 minutes the optimization problem is solved and the control inputs for the next 15 minutes in the simulation are applied. For every time in $T(t_0)$, the demand forecast for each origin, destination pair is computed by taking the sample average of the corresponding values from different days in the dataset mentioned in section 3.5.1.

3.5.4 Results

The main results obtained with the four controllers are summarized in Table 3.1. To evaluate the customer satisfaction we measure the waiting times and journey times. To measure the operational cost we use the total distance driven. The mean waiting time corresponds to the average waiting time of all customers during the whole day, while the mean journey time is equivalent to the average
CHAPTER 3. FLEET CONTROL FOR ON-DEMAND RIDESHARING SYSTEMS

41

Relative Difference [%]

AMoD-MPC RAMoD-Reactive AMoD-Reactive

− 20
0
20
40
60

Figure 3.3: Relative performance difference with respect to the proposed RAMoD-MPC algorithm.

time of all journeys starting from the request submission time until the drop-off time at destination. The distance driven describes the sum of all distances traveled through the whole simulation day by all cars including pickup, drop-off, and rebalancing trips.

Table 3.1: Performance of the Controllers

<table>
<thead>
<tr>
<th>Controller</th>
<th>Mean Waiting Time</th>
<th>Mean Journey Time</th>
<th>Distance Driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMoD-MPC</td>
<td>4 min 3 s</td>
<td>22 min 18 s</td>
<td>372'637 km</td>
</tr>
<tr>
<td>AMoD-MPC</td>
<td>4 min 15 s</td>
<td>17 min 30 s</td>
<td>453'450 km</td>
</tr>
<tr>
<td>RAMoD-Reactive</td>
<td>6 min 8 s</td>
<td>24 min 25 s</td>
<td>367'785 km</td>
</tr>
<tr>
<td>AMoD-Reactive</td>
<td>6 min 48 s</td>
<td>20 min 30 s</td>
<td>476'183 km</td>
</tr>
</tbody>
</table>

3.5.5 Remarks and Observations

A few comments are in order. As Fig. 3.3 shows, the ride-sharing algorithms drive 20% less distance than the single occupancy algorithms. Additionally, the algorithms using MPC achieve more than 40% lower mean waiting times than the reactive algorithms by preemptively rebalancing available vehicles to locations where future demand is expected to appear. From this we see that RAMoD-MPC is at least as good and somewhere better in terms of mean waiting time and travel distance compared to the benchmark algorithms. For this case study, the time to solve (3.12) for Algorithm 2 was consistently under 15 seconds.

The number of waiting customers throughout the day is shown in Fig. 3.4. Interestingly, RAMoD-MPC has more waiting customers than AMoD-MPC between 7:00am and 9:00am. We believe this is because RAMoD-MPC waits until there are sufficiently many customers that can be grouped together using ride-sharing before picking up customers. By doing this it able to serve the demand with fewer cars, allowing it to be more prepared for later demand between 9:00am and 2:00pm. The driving distance for both fleets is also shown in Fig. 3.4 showing AMoD-MPC driving substantially more from 7:00am-2:00pm, aligning with our hypothesis.
These advantages, however, come at a price. As Fig. 3.3 shows, ride-sharing increases the average journey time of customers by more than 10%. This is caused by vehicles taking detours to service multiple customers. Despite the extra journey time, we emphasize that the societal and economic advantages of ride-sharing are substantial. Having the service fleet driving less reduces CO$_2$ emissions and fuel costs. In this particular case study, the $RAMoD$-$MPC$ drives 80'000 km less than the $AMoD$-$MPC$ while having a similar mean wait time. Using the standard CO$_2$ emission rate for cars equipped with gasoline engines of $25$ kg/100 km [38, Chpt. 1], in this particular case ride-sharing would reduce CO$_2$ emissions by 7'300 metric tons every year. In terms of fuel cost, considering a standard gasoline consumption of $8$ L/100 km [38] would lead to 2.3 M L gasoline saved annually, resulting in savings exceeding 2 M $ per year [28].
3.6 Discussion

In this chapter we presented a model predictive control algorithm, called RAMoD-MPC, to coordinate a fleet of self-driving vehicles for servicing travel requests in a ridesharing setting. We compared RAMoD-MPC to ridehailing approaches in numerical simulations to quantify the value of ridesharing. Our experiments showed that RAMoD-MPC algorithm outperforms the state-of-the-art unit-capacity mobility algorithms in terms of total driving distance and reactive ride-sharing algorithms in terms of mean wait time. In particular, by slightly increasing the total trip length for customers, the RAMoD-MPC algorithm is able to significantly reduce the distance traveled by mobility providers.

There are several important directions for future work. First, we note that the geographic discretization used in our experiments (shown in 3.2) is coarser than industry standard which is typically Geohash 6. Thus one important next step is to increase the spatial resolution that is used by the control algorithm. A second important direction is generalization to higher vehicle capacity. This work utilizes vehicles that can carry up to 2 customers at a time, but service vehicles often have at least 3 passenger seats in practice.

Generalizing RAMoD-MPC for higher spatial resolution and larger vehicle capacities would likely require an increase in the computational power needed to ensure that the algorithm runs in real-time. This is largely because RAMoD-MPC is responsible for both rebalancing actions and rider-driver assignment. One way to accommodate higher spatial resolution and large vehicle capacities is to decouple these responsibilities, i.e., use an existing algorithm [68, 98, 44, 83] to determine rebalancing actions, and develop a separate algorithm which will decide which riders should share a ride.

With such a decomposition in mind, we will study the problem of deciding which riders should share a ride in the next section. We will approach the problem from a matching theoretic perspective, where riders and drivers are represented as vertices in a graph, and edges represent promising or feasible matches between a rider and a driver (in the ridehailing setting) or between several riders (in the ridesharing setting). Within such models, the set of feasible actions for the system correspond to the space of all matchings within the graph, and tools from matching theory can then be used to compute efficient matchings between riders and drivers.
Part III

Online Matching for Ridehailing and Ridesharing Systems
Chapter 4

Online Matching for Ridesharing

In this chapter we move away from fleet coordination and focus on the problem of matching riders in a ridesharing service. Matching riders in a ridesharing service involves identifying ridesharing groups, i.e., groups of customers that can be efficiently served by a single vehicle. Once the ridesharing system identifies a set of ridesharing groups it wants to implement, a separate downstream algorithm can then assign drivers to serve the ridesharing groups. In this chapter we will study algorithm design from a worst-case perspective, which is appropriate when the demand is non-stationary and difficult to predict.

4.1 Introduction

We study an online hypergraph matching problem inspired by ridesharing and delivery applications. The vertices of a hypergraph are revealed sequentially and must be matched within \( d \) timesteps of their reveal, otherwise they will leave the system in favor of an outside option. Hyperedges can contain at most \( k \) vertices and are revealed to the algorithm once all of its vertices have arrived, and can only be included into the matching before any of its vertices leave the system. We study utility maximization and cost minimization objectives in this model.

Algorithms are evaluated by their competitive ratio on the weight of the matching they produce. In the utility maximization setting, hyperedge weights represent utility, and the goal is to find a matching with large weight. In cost minimization settings, hyperedge weights represent costs and the goal is to find a matching with low weight.

This setup is inspired by the prevalence of online matching techniques for modern on-demand services like ridesharing and same day delivery services for meals, groceries, and parcels. In these services, couriers are matched to delivery jobs in real-time, and online graph matching techniques are often used to identify pairs of delivery jobs which can be served efficiently by a single courier. Vehicles used by couriers often have the capacity to carry many customers or packages in the contexts.
of ridesharing and delivery, respectively. This observation motivates an online hypergraph matching problem where hyperedges represent sets of jobs that can be efficiently served by a single courier. The cardinality bound $k$ on hyperedges represents the capacity of service vehicles and departure of vertices after $d$ timesteps represents a disutility for waiting.

4.1.1 Overview of Results

We study Online-$(k, d)$-Max-Matching and Online-$(k, d)$-Min-Matching, which are the utility maximization and cost minimization variants of online hypergraph matching with delays respectively.

For Online-$(k, d)$-Max-Matching we show that whenever $k \geq 3$, the optimal competitive ratio is $\frac{1}{d}$, and we present a polynomial-time randomized batching algorithm which is $\frac{1}{d}$-competitive. To prove that no algorithm can have a competitive ratio better than $\frac{1}{d}$, we first define an adversarial variant of the secretary problem and prove that its optimal competitive ratio is $\frac{1}{d}$. We then prove that this adversarial secretary problem is a special case of Online-$(k, d)$-Max-Matching.

We study Online-$(k, d)$-Min-Matching with monotone costs, which are natural cost structures for ridesharing and delivery problems. We present results for $k = 2$ and $k > 2$ separately, since the techniques used in the two cases are quite different.

For $k = 2$, the optimal competitive ratio for deterministic algorithms is $\frac{3}{2}$. We present a polynomial-time deterministic thresholding algorithm which is $\frac{3}{2}$-competitive. Furthermore, we show that there exists a deterministic $\frac{1+\sqrt{5}}{2}$-competitive algorithm in the more general case where the maximum delay $d$ for each vertex is different and unobserved. This result shows that the knowledge of $d$ does not improve the optimal competitive ratio for deterministic algorithms by more than $\frac{1+\sqrt{5}}{2} - \frac{3}{2} \approx 0.118$.

For $k > 2$, there exists a randomized batching algorithm which is $2 - \frac{1}{d}$ competitive. This algorithm, however, is not polynomial-time. Using a reduction from set cover, we show that it is NP-hard to achieve a competitive ratio better than $\log k - O(\log \log k)$. Leveraging the reduction, we construct a randomized greedy batching procedure and show that it is $(2 - \frac{1}{d}) \log k$-competitive, establishing the optimal polynomial-time competitive ratio up to a factor of $2 - \frac{1}{d}$.

4.1.2 Organization

The remainder of this chapter is organized as follows. We review literature related to these problems in Section 4.2. In Section 4.3 we present both the utility maximization and cost minimization versions of the online hypergraph matching problem with delays. In Section 4.4, we present our results for the utility maximization setting wherein we present a polynomial-time algorithm which achieves the optimal competitive ratio. In Section 4.5, we present our results for the cost minimization setting with monotone cost structures. We conclude and discuss directions for future work in Section 4.6.
4.2 Related Work

This chapter is related to online resource allocation and online matching problems with delays. In the following, we survey some recent results in these fields and discuss how they relate to this work.

Online resource allocation has applications in advertising [53], network routing problems [61], and ridesharing [9]. In general, obtaining polynomial-time constant-factor approximations for these problems is NP-hard, so [53, 61] study special cases where users can request for at most $k$ different resource types. This assumption is realistic in most settings as a single user will not need more than a constant number of resources. [53] present a sample-and-price algorithm and show that it is $\frac{1}{k^{2}}$-competitive if the arrival order of requests is uniformly random. If the full distribution over user requests is known to the network operator, [61] show via approximate dynamic programming that a competitive ratio of $\frac{1}{k+1}$ is possible. These works do not consider the trade-off between waiting costs and market thickness, i.e. deferring actions to gain more information about future requests.

The trade-off between waiting costs and market thickness has been studied extensively in the matching theory literature. The option of waiting for a thicker market is studied in [8, 27, 3, 7], and [9] in different contexts. In these works, requests arrive sequentially in time, and the platform can match pairs of requests together.

In [8, 27], the platform can defer matching decisions but must pay a cost proportional to the total time requests wait before being matched. In [3, 7] requests arrive to the system according to some known distribution. Requests will stay in the system for a random and unknown amount of time, so the platform operator risks users leaving the system if it chooses to wait for a thicker market. Online matching with deadlines was studied in [9]. Requests enter the system sequentially and are willing to wait up to $d$ timesteps to be matched, after which they will leave the system. Here, $d$ is known to the platform. The authors present a $\frac{1}{4}$-competitive algorithm and also show that no algorithm can have a competitive ratio better than $\frac{1}{2}$.

Batching is a natural approach for online matching whereby a maximum weight matching is computed periodically to match vertices that have arrived after the previous matching was computed. [9] prove that a batching algorithm is 0.279-competitive in an online matching problem with delays when the arrival order of vertices is uniformly random. This improves upon the $\frac{1}{4}$-competitive algorithm they propose for adversarial arrival order. [5] use a batching algorithm for on-demand high capacity ridesharing whereby an assignment of drivers to riders is computed every 30 seconds. They show via numerical experiments that ridesharing can enable ridehailing services to reduce fleet size while maintaining service quality to riders.

This section contributes generalizations to these aforementioned works in two ways. With respect to the online matching literature, we study a hypergraph generalization where groups of more than 2 vertices can be matched together. This is motivated by ridesharing applications where service vehicles can hold more than 2 customers at any given time. With respect to the online resource allocation literature, we present results that do not assume any distributional information on the
input instances. Our model also allows for waiting so that the platform has \( d \) units of time after a request appears to determine whether to accept or to reject it. This is in contrast to the works of [53, 61] where the platform must immediately decide between accepting or rejecting upon a request’s arrival.

### 4.3 Model

The vertices of a hypergraph \( H = (V, E, w) \) are revealed sequentially, one per timestep, and each vertex must be matched within \( d \) timesteps of its arrival. An edge (For the sake of brevity, we use edge as an abbreviation of hyperedge when it is clear that we are discussing hypergraphs.) along with its weight is revealed once all of its vertices have been revealed. Furthermore, we say a vertex is critical during the last timestep in which it can be matched, i.e. a vertex revealed at timestep \( t \) becomes critical at timestep \( t + d - 1 \). The cardinality of every edge is at most a small constant \( k \).

The set of such problem instances is a family of shareability hypergraphs, defined as follows.

**Definition 2 (Shareability Hypergraph).** A hypergraph \( H = (V, E, w) \) is a Shareability Hypergraph with parameters \((n, d, k)\) if and only if:

1. The set of vertices is ordered \( V = [n] \),
2. \( \text{diam}(e) < d \) for all \( e \in E \), where \( \text{diam}(e) := \max_{i,j \in e} |i - j| \),
3. The rank of \( H \) is at most \( k \), i.e., \( |e| \leq k \) for all \( e \in E \),
4. \( w : E \to \mathbb{R}_+ \), i.e., all hyperedge weights \( w(e) \) are non-negative.

We study both utility maximization and cost minimization objectives in this model. In the utility maximization setting, the weight \( w(e) \) of an edge \( e \) denotes the utility of including \( e \) in the matching.

In the cost minimization setting, all vertices must be matched, and the weight \( w(e) \) of an edge \( e \) represents the cost incurred when including \( e \) in the matching.

We now formally define the utility maximization version of online hypergraph matching with deadlines, which we call the **Online-\((k,d)\)-Max-Matching** problem.

**Definition 3 (Online-\((k,d)\)-Max-Matching).** The vertices of a shareability hypergraph \( H = (V, E, w) \) with parameters \((n, d, k)\) are revealed sequentially, one per timestep. The vertex that is revealed at timestep \( t \) will depart at timestep \( t + d \). A hyperedge (along with its weight) \( e \in E \) is revealed once all of its vertices have been revealed. The **Online-\((k,d)\)-Max-Matching** problem is to construct a matching of \( H \) with large weight, subject to the constraint that a hyperedge can only be included after it is revealed but before any of its vertices depart, and hyperedge inclusion is irrevocable.

The cost minimization version, which we call the **Online-\((k,d)\)-Min-Matching** problem can be defined similarly. To avoid the degenerate outcome where the empty matching is optimal, a cost is
incurred for each vertex that is unmatched. Equivalently, the hyperedge set contains all singleton
sets, i.e. \( \{ i \} \in E \) for every \( i \in V \) where \( w(\{ i \}) \) is equal to the cost incurred for leaving \( i \) unmatched.
The optimal solution corresponds to a minimum weight cover of \( V \) by sets in \( E \).

**Definition 4** (Online-\((k, d)\)-Min-Matching). The vertices of a shareability hypergraph \( H = (V, E, w) \) with parameters \((n, d, k)\) are revealed sequentially, one per timestep. Each vertex \( i \) has a weight \( w(\{ i \}) \), which represents the cost of leaving this vertex unmatched. The vertex that is revealed at timestep \( t \) will depart at timestep \( t + d \). A hyperedge (along with its weight) \( e \in E \) is revealed once all of its vertices have been revealed. The Online-\((k, d)\)-Min-Matching problem is to find a matching of \( H \) with low cost subject to the constraint that a hyperedge can only be included after it is revealed but before any of its vertices disappear, and hyperedge inclusion is irrevocable. Here, a matching’s cost is the sum of its weight and the weight of all unmatched vertices.

**Remark 1** (Parameter Knowledge). Throughout this chapter, unless stated otherwise, the algorithm knows the value of \( d \), but does not know (and does not need to know) the values of \( k \) or \( n \).

### 4.3.1 Offline Variants

Our analysis will involve the offline versions of Online-\((k, d)\)-Max-Matching and Online-\((k, d)\)-Min-Matching where the entire hypergraph can be observed before the matching is chosen. We now define the offline analogs \( k\)-Max-Matching and \( k\)-Min-Matching respectively.

**Definition 5** (\( k\)-Max-Matching). We use \( k\)-Max-Matching to refer to the offline version of Online-\((k, d)\)-Max-Matching where the whole hypergraph can be observed before computing a matching.

**Definition 6** (\( k\)-Min-Matching). We use \( k\)-Min-Matching to refer to the offline version of Online-\((k, d)\)-Min-Matching where the whole hypergraph can be observed before computing a matching.

**Remark 2** (Hardness of Offline Variants). Computing optimal solutions to \( k\)-Max-Matching and \( k\)-Min-Matching is NP-hard whenever \( k \geq 3 \). Therefore we can expect computational challenges to arise in the online variants as well.

### 4.3.2 Approximation Ratios and Competitive Ratios

In online optimization problems, the problem instance is revealed over time, and irrevocable decisions must be made without full knowledge of the problem instance. In offline problems, all decisions are made with full knowledge of the problem instance.

For a family of maximization problem instances \( \{ \max_{x \in X_p} f_p(x) : p \in P \} \), the performance of an offline algorithm \( A \) is determined by its approximation ratio \( \rho(A) \). The performance of an online algorithm \( A' \) is determined by its competitive ratio \( \bar{\rho}(A') \). Formally,

\[
\rho(A) := \inf_{p \in P} \frac{\mathbb{E}[f_p(A(p))]}{f_p(\text{OPT}_p)} \quad \text{and} \quad \bar{\rho}(A') := \inf_{p \in B} \frac{\mathbb{E}[f_p(A'(p))]}{f_p(\text{OPT}_p)},
\]
where $\text{OPT}_p \in \arg \max_{x \in X_p} f_p(x)$ is an optimal offline solution. The approximation and competitive ratios are thus the worst case ratio between the expected objective value produced by the algorithm and the objective value of an optimal offline solution. The approximation and competitive ratios will be in $[0, 1]$ with larger values being preferred.

Similarly, for a family of minimization problem instances $\{\min_{x \in X_p} f_p(x) : p \in \mathcal{P}\}$, the approximation ratio of an offline algorithm $A$ and the competitive ratio of an online algorithm $A'$ are defined as

$$
\rho(A) := \sup_{p \in \mathcal{P}} \frac{\mathbb{E}[f_p(A(p))]}{f_p(\text{OPT}_p)} \quad \text{and} \quad \bar{\rho}(A') := \sup_{p \in \mathcal{P}} \frac{\mathbb{E}[f_p(A'(p))]}{f_p(\text{OPT}_p)}
$$

where $\text{OPT}_p \in \arg \min_{x \in X_p} f_p(x)$ is an optimal offline solution. The approximation ratio and competitive ratio will be in $[1, \infty]$ with smaller values being preferred.

### 4.4 Algorithms and Hardness for Utility Maximization

In this section, we study the Online-$(k, d)$-Max-Matching problem where $k \geq 3$ (see [9] for treatment of the $k = 2$ setting). Our first result characterizes the optimal competitive ratio for Online-$(k, d)$-Max-Matching.

**Theorem 1.** The optimal competitive ratio for Online-$(k, d)$-Max-Matching is $\frac{1}{d}$ whenever $k \geq 3$.

We establish Theorem 1 in two steps. First, we show in Section 4.4.1 that no algorithm can have a competitive ratio larger than $\frac{1}{d}$. Then, we present a randomized batching algorithm and prove that it is $\frac{1}{d}$-competitive in Section 4.4.2. This algorithm, however, is not polynomial-time since $k$-Max-Matching is NP-hard when $k \geq 3$.

Our next result shows that the optimal competitive ratio of $\frac{1}{d}$ can be achieved in polynomial time. To this end, we present two algorithms: Randomized-Batching defined in Algorithm 3 and Depth-$k$-Greedy defined in Algorithm 4. Depth-$k$-Greedy is a $k$-Max-Matching algorithm which requires knowledge of $k$. Randomized-Batching is an Online-$(k, d)$-Max-Matching algorithm that uses a $k$-Max-Matching algorithm as a subroutine.

**Theorem 2.** Randomized-Batching using Depth-$k$-Greedy as a subroutine is $\frac{1}{d}$-competitive for Online-$(k, d)$-Max-Matching and runs in $O(|E|^{k+1})$, which is polynomial in the size of $H$ for any fixed $k$.

We present the proof of Theorem 2 in Section B.3.

#### 4.4.1 Upper Bounds

We now show that if the matching capacity $k$ is at least 3, no online algorithm can have a better competitive ratio than $\frac{1}{d}$, with $d$ being the number of timesteps a vertex stays in the system.
Lemma 5 (Upper bound on the competitive ratio). If \( k \geq 3 \), then no online algorithm can have a competitive ratio better than \( \frac{1}{2} \) for Online-(\( k, d \))-Max-Matching.

Proof of Lemma 5. Consider a family \( G \) of shareability hypergraphs where \( G \in G \) is of the form \( G = (V, E, w) \) with \( V = \{0, 1, ..., 2d - 2\} \), \( w \in \mathbb{R}_{+}^{E} \), and \( E = \{e_t\}_{t=0}^{d-1} \) with \( e_0 := \{0, d - 2, d - 1\} \), \( e_{d-1} := \{d - 1, d, 2d - 2\} \) and

\[
e_t := \{t, d - 1, t + d - 1\} \text{ for all } 0 < t < d - 1.
\]

First, note that any matching can have at most one hyperedge since \( e_t \cap e_{t'} \neq \emptyset \) for any \( t, t' \) since all edges contain the vertex \( d - 1 \). Second, every edge \( e_t \) can only be chosen in the timestep in which its weight is revealed. This is due to the following reasons. First, the earliest vertex in \( e_t \) is \( t \), meaning \( e_t \) can only be added to the matching at or before time \( t + d - 1 \). Second, the latest vertex in \( e_t \) is \( t + d - 1 \), meaning the earliest time \( e_t \) can be chosen is \( t + d - 1 \). For these reasons Online-(\( k, d \))-Max-Matching restricted to instances in \( G \) is equivalent to the Adversarial Secretary Problem with \( d \) applicants (\( d \)-ASP), which is defined below. See Appendix B.2 for a visualization of graphs in \( G \) and the relationship to \( d \)-ASP.

Definition 7 (Adversarial Secretary Problem with \( d \) applicants (\( d \)-ASP)). The Adversarial Secretary Problem with \( d \) applicants, which we denote by \( d \)-ASP, has the following setting: There are \( d \) applicants for a secretary position, with corresponding non-negative aptitude scores \( \{w_i\}_{i=1}^{d} \). At most one applicant can be hired and decisions are irrevocable. Time is discretized into \( d \) timesteps. The aptitude \( w_i \) of the \( i \)th applicant is revealed at time \( i \), and the \( i \)th applicant can only be hired at time \( i \). The goal is to maximize the aptitude of the hired secretary.

Contrary to the canonical secretary problem where the arrival order of candidates is uniformly random, \( d \)-ASP makes no distributional assumption on the ordering of the candidates. Next we establish a hardness result for \( d \)-ASP.

Lemma 6 (Adversarial Secretary Problem). The best possible competitive ratio for the adversarial secretary problem with \( d \) applicants is \( \frac{1}{2} \).

See Appendix B.1 for a proof of Lemma 6. Since \( d \)-ASP is a special case of Online-\((k, d)\)-Max-Matching, this proves that the best possible competitive ratio of the latter is at most \( \frac{1}{2} \). \( \square \)

4.4.2 Algorithms

Section Outline: In this section we present a polynomial-time \( \frac{1}{2} \)-competitive algorithm for Online-\((k, d)\)-Max-Matching. First, we show that any offline algorithm \( A \) for \( k \)-Max-Matching can be converted into an online algorithm \( A' \) for Online-\((k, d)\)-Max-Matching which inherits the approximation ratio of \( A \) up to a factor \( \frac{1}{2} \) loss, i.e., \( \bar{\rho}(A') = \frac{1}{2} \rho(A) \). However, this is not sufficient to
prove that $\frac{1}{d}$-competitiveness is achievable in polynomial time because $k$-Max-Matching is NP-hard, and using a $O\left(\frac{1}{k^2}\right)$-approximation algorithm for $k$-Max-Matching leads to a suboptimal $O\left(\frac{1}{kd}\right)$ competitive ratio. To overcome this, we show that for this $O\left(\frac{1}{kd}\right)$-competitive procedure, there cannot be too many edges in the optimal (i.e., maximum weight) matching that contribute less than a $\frac{1}{d}$ fraction of their weight to the procedure’s matching in expectation. Using this observation, the final ingredient of our algorithm is a search over matchings to improve the contribution of these edges from $\frac{1}{kd}$ to $\frac{1}{d}$.

We present Randomized-Batching described in Algorithm 3 and study its performance. To discuss this algorithm, we first define the notion of batches, visible edges, and cut edges.

**Definition 8 (Batches).** Given a shareability hypergraph $H = (V, E, w)$, and integers $z \in \{0, 1, ..., d - 1\}$, $i \leq \frac{n}{d}$, we define the batch $W_{z,i} := (V_{z,i}, E_{z,i}, w)$ to be a subgraph of $H$ containing $d$ contiguous vertices and all induced edges. Concretely,

$$V_{z,i} := \{z + id + t\}_{t=0}^{d-1},$$

$$E_{z,i} := \{e \in E : e \subset V_{z,i}\}.$$

**Definition 9 (Visible and Cut edges).** For a shareability hypergraph $H = (V, E, w)$ and a given $z \in \{0, 1, ..., d - 1\}$, we say that an edge $e \in E$ is visible with respect to $z$ if $e \in E_{z,i}$ for some $i \leq \frac{n}{d}$. If an edge is not visible with respect to $z$, we say it is cut with respect to $z$.

**Algorithm 3: Randomized-Batching($H; A$)**

1. **Input:** Shareability Graph $H = (V, E, w)$, Offline matching algorithm $A$;
2. **Output:** Matching $M$;
3. $M \leftarrow \emptyset$;
4. Draw $Z$ uniformly at random from $\{0, 1, 2, ..., d - 1\}$;
5. for $1 \leq t \leq n$ do
6.   Vertex $t$ is revealed;
7.   if $(t + 1 \equiv Z \mod d)$ or $(i = n)$ then
8.     Define the batch number $i$ such that $id + Z = t + 1$;
9.     Use $A$ to compute a matching $M_{Z,i}$ for the batch $W_{Z,i}$;
10.    $M \leftarrow M \cup M_{Z,i}$;
11. Return $M$

Randomized-Batching runs a $k$-Max-Matching algorithm $A$ every $d$ timesteps, effectively treating the online problem as $n/d$ separate offline problems. The start and end of the batches is determined by $Z$ which is chosen uniformly at random over $\{0, 1, ..., d - 1\}$. Figure 4.1 illustrates a batching procedure.
CHAPTER 4. ONLINE MATCHING FOR RIDESHARING

Figure 4.1: An example of batching for the online matching problem. The time horizon is partitioned into batches of length $d$, and an offline algorithm is used to compute a matching (blue hyperedges) for each batch.

**Remark 3.** A batching algorithm without randomization will have a competitive ratio of zero. In particular, its performance is poor if there are edges with large weight that are cut by the batching. Figure 4.2 illustrates such a setting.

Figure 4.2: Consider a deterministic batching algorithm with $z = 0$. The batching algorithm is unable to select the red edges, since they are not visible with respect to $z = 0$. If the red edges each have weight 1 and all other edges have weight $\epsilon$, then the weight of the matching obtained by batching is at most $\epsilon |E|$, whereas the optimal solution has weight at least 3. Hence the competitive ratio of batching is at most $\frac{\epsilon |E|}{3}$, which converges to zero as $\epsilon \to 0$.

Fortunately, we can remedy the concerns of Remark 3 via randomization. The randomness of $Z$ ensures that every edge has a positive chance of being visible.

The following theorem shows how Randomized-Batching can be used to convert a $k$-Max-Matching algorithm into an algorithm for Online-$(k, d)$-Max-Matching while inheriting its approximation ratio up to a factor of $\frac{1}{d}$.

**Theorem 3** (Competitive Ratio of Randomized-Batching). Let $A$ be an offline matching algorithm for $k$-Max-Matching with approximation ratio $\rho(A)$. Algorithm 3 using $A$ is $\frac{\rho(A)}{d}$-competitive for Online-$(k, d)$-Max-Matching.

To establish Theorem 3 we will prove a stronger result described in Lemma 7.

**Lemma 7** (Performance of Randomized-Batching). Let $A$ be an algorithm for $k$-Max-Matching with approximation ratio $\rho(A)$. For any maximum weight matching $M^*$ in $H$, the matching $M$ returned by Algorithm 3 using $A$ satisfies

$$\mathbb{E}[w(M)] \geq \frac{\rho(A)}{d} \sum_{e \in M^*} w(e) (d - \text{diam}(e)).$$

Once Lemma 7 is established, proving Theorem 3 is straightforward.

**Proof of Theorem 3.** Apply Lemma 7 and note that $d - \text{diam}(e) \geq 1$ for all $e \in E$ since $H$ is a shareability hypergraph with parameters $(n, d, k)$. 

\qed
Proof of Lemma 4.3 Let \( A \) be an offline matching algorithm for \( k\)-Max-Matching with approximation ratio \( \rho(A) \) and consider an instance \( H = (V,E,w) \) of Online-\((k,d)\)-Max-Matching. For each \( z \in \{0,1,\ldots,d-1\} \) define \( E_z := \bigcup_{i=\lceil n/d \rceil}^{\lfloor n/d \rfloor} E_{z,i} \) to be all edges visible with respect to \( z \) (See Definitions 8 and 9). With this notation, given \( Z = z \), Randomized-Batching returns a matching \( M := \bigcup_i M_{z,i} \), where \( M_{z,i} \) is obtained by applying \( A \) to \( W_{z,i} \). We will use \( M_{z,i}^* \) to denote a maximum weight matching in \( W_{z,i} \). We will make use of the following observation.

Observation 1. For any \( e \in E \) we have \( \sum_{z=0}^{d-1} \mathbb{I}_{[e \in E_z]} = d - \text{diam}(e) \).

Proof of Observation 4.4 Fix any \( e \in E \) and let \( t \) be the smallest arrival time of any vertex in \( e \). Without loss of generality (by a linear change of coordinates) we assume that \( t = 0 \). By definition of \( \text{diam}(e) \), all vertices of \( e \) appear between time 0 and \( \text{diam}(e) \), i.e., \( e \subset \{0,1,\ldots,\text{diam}(e)\} \).

For any \( z \) satisfying \( \text{diam}(e) - d + 1 \leq z \leq 0 \), we have \( z \leq 0 \) and \( z + d - 1 \geq \text{diam}(e) \), and thus \( \{0,1,\ldots,\text{diam}(e)\} \subset \{z,z+1,\ldots,z+d-1\} \). Recalling Definition 8 any edge that is a subset of \( \{z,z+1,\ldots,z+d-1\} \) will belong to \( E_z \). This implies that \( e \in E_z \) for every \( z \in \{\text{diam}(e) - d + 1,\ldots,-1\} \mod d \). In particular, \( e \in E_z \) for \( d - \text{diam}(e) \) values of \( z \).

Let \( M^* \) be a maximum weight matching for \( H \), and let \( M_{z,i}^* \) be a maximum weight matching for \( H_z := (V,E_z,w) \). Given \( Z = z \), we have

\[
\begin{align*}
w(M) &= \sum_i w(M_{z,i}) \\
&\geq \sum_i \rho(A) w(M_{z,i}^*) = \rho(A) w(M_z^*),
\end{align*}
\]

where (a) is because \( A \) a \( \rho(A) \)-approximation algorithm, and \( M_{z,i}^* \) is obtained by applying \( A \) to \( W_{z,i} \). Because \( Z \) is uniformly distributed over \( \{0,1,\ldots,d-1\} \), we have

\[
\mathbb{E}_Z [w(M)] = \sum_{z=0}^{d-1} \mathbb{P}[Z = z] \mathbb{E}[w(M)|Z = z]
\]

\[
\begin{align*}
&\geq \frac{1}{d} \sum_{z=0}^{d-1} \rho(A) \sum_{e^* \in M_z^*} w(e^*) \\
&\geq \frac{\rho(A)}{d} \sum_{z=0}^{d-1} \sum_{e^* \in M_z^*} w(e^*) \\
&= \frac{\rho(A)}{d} \sum_{z=0}^{d-1} \sum_{e^* \in M_z^*} w(e^*) \mathbb{I}_{[e^* \in E_z]} \\
&= \frac{\rho(A)}{d} \sum_{e^* \in M^*} w(e^*) \sum_{z=0}^{d-1} \mathbb{I}_{[e^* \in E_z]} \\
&\geq \frac{\rho(A)}{d} \sum_{e^* \in M^*} w(e^*) (d - \text{diam}(e)).
\end{align*}
\]
Where (a) is due to (4.1), (b) is because $M^* \cap E_z$ is a matching in $H_z$ and $M^*_z$ is a maximum weight matching in $H_z$. Finally, (c) is due to Observation 1.

The next lemma shows that if there exists an optimal solution $M^*$ with additional structure, namely all its edges have small diameter, Randomized-Batching achieves a competitive ratio that is independent of $d$.

Lemma 8 ($M^*$ with small diameter edges). Suppose all edges in $M^*$ have small diameter, i.e., there exists $c \in (0,1)$ such that $\text{diam}(e) \leq cd$ for all $e \in M^*$. Then by Lemma 7 Randomized-Batching($\cdot$, $A$) will produce a matching with weight at least $(1-c)\rho(A)w(M^*)$. In other words, Randomized-Batching($\cdot$, $A$) is $(1-c)\rho(A)$-competitive against an offline algorithm which knows the hypergraph ahead of time, but can only include edges with diameter less than $cd$ in its matching.

We now briefly explore the implications of Theorem 3 for different choices of $A$. The best known polynomial-time algorithms for $k$-Max-Matching are $O(\frac{1}{k})$-approximations [11, 15], and in particular the Greedy algorithm is a $\frac{1}{k}$-approximation. See Appendix A.1 for a description of Greedy and further literature on $k$-Max-Matching.

Corollary 1. Choosing $A$ to be Greedy when running Algorithm 3 gives a polynomial-time algorithm with competitive ratio of $\frac{1}{kd}$ for Online-($k, d$)-Max-Matching.

Corollary 2. If $A^*$ is an exact procedure for $k$-Max-Matching, i.e., $\rho(A^*) = 1$, then Algorithm 3 using $A^*$ as a subroutine achieves a competitive ratio of $\frac{1}{d}$ for Online-($k, d$)-Max-Matching.

Corollary 2 in conjunction with Lemma 5 proves Theorem 1, establishing $\frac{1}{d}$ as the optimal competitive ratio for Online-($k, d$)-Max-Matching. However, it is known from [39] that achieving $\rho(A) = \Omega\left(\frac{\log k}{k}\right)$ for $k$-Max-Matching is NP-hard, and thus no exact procedure $A^*$ can run in polynomial-time unless $P = NP$. For this reason, Corollary 2 does not show that a competitive ratio of $\frac{1}{d}$ is achievable in polynomial-time.

Polynomial-time algorithms

To get a $\frac{1}{d}$-competitive algorithm in polynomial time, we need to improve the contribution from edges with large diameter. Note that in Lemma 7 each edge $e$ in a maximum weight matching $M^*$ contributes a $\frac{\rho(A)}{d}(d - \text{diam}(e))$ fraction of its weight to the matching produced by Randomized-Batching. In other words, edges with larger diameter contribute less to our bound. To address this, we present Depth-$k$-Greedy described in Algorithm 4. Depth-$k$-Greedy improves the contribution of large diameter edges by first creating candidate matchings, each of which contains only large diameter edges. It then adds edges to each candidate matching via Greedy until it becomes a maximal matching, and returns the maximal matching with the largest weight. The following result shows that Randomized-Batching with Depth-$k$-Greedy as the choice for $A$ yields a polynomial-time $\frac{1}{d}$-competitive algorithm.
Theorem 2 (Restated). Randomized-Batching(, A) using \( A = \text{Depth-k-Greedy} \) is \( \frac{1}{d} \)-competitive for Online-(k, d)-Max-Matching. Furthermore, its running time is \( O\left(|E|^{k+1}\right) \) which is polynomial in the size of the shareability hypergraph \( H \) for any fixed \( k \).

Algorithm 4: Depth-k-Greedy(\( H \))

1. Input: Hypergraph \( H = (V, E, w) \) where the vertices are ordered;
2. Output: A matching \( M \) for \( H \);
3. Define \( L \) to be the first \( k \) vertices;
4. Define \( R \) to be the last \( k \) vertices;
5. Define \( M_{L,R}(H) := \{ M \text{ is a matching} : e \cap L \neq \emptyset, e \cap R \neq \emptyset \forall e \in M \} \);
6. Enumerate all matchings in \( M_{L,R}(H) \) as \( \{M_i\}_{i = 1}^{\left|M_{L,R}(H)\right|} \);
7. for \( 1 \leq i \leq |M_{L,R}(H)| \) do
   8. \( V_i \leftarrow \{i \in V : i \text{ is matched in } M_i\} \);
   9. \( \tilde{M}_i \leftarrow \text{{Greedy}}(V \setminus V_i, E) \);
10. \( i^* \leftarrow \arg \max_i w(M_i \cup \tilde{M}_i) \);
11. \( M \leftarrow M_{i^*} \cup \tilde{M}_{i^*} \);
12. Return \( M \)

We present the proof of Theorem 2 in Appendix B.3. We have several remarks regarding \( \text{Depth-k-Greedy} \). \( \text{Depth-k-Greedy} \) needs to know the value of \( k \). While replacing \( \text{Greedy} \) with \( \text{Depth-k-Greedy} \) improves the competitive ratio of Randomized-Batching from \( \frac{1}{kd} \) to \( \frac{1}{d} \), the same is not true for \( k \)-Max-Matching. Specifically, \( \text{Greedy} \) and \( \text{Depth-k-Greedy} \) have the same approximation ratio of \( \frac{1}{k} \) for \( k \)-Max-Matching. \( \text{Depth-k-Greedy} \) uses the arrival order of the vertices within each batch, information which is not used by generic \( k \)-Max-Matching algorithms as there is no meaningful temporal ordering of vertices in offline optimization problems. The utilization of the vertex arrival order within \( \text{Depth-k-Greedy} \) is what enables the performance improvement over Corollary 1.

4.5 Algorithms and Hardness for Cost Minimization

In this section we study Online-(k, d)-Min-Matching with monotone costs. As a reminder, in this setting, hyperedge weights \( w(e) \) represent the cost of including \( e \) in the matching. We first present a monotone cost assumption in Section 4.5.1. In Section 4.5.2 we prove that for \( k = 2 \), the optimal competitive ratio for deterministic algorithms is \( \frac{3}{2} \) and is achieved by a thresholding algorithm. We study the \( k > 2 \) setting in Section 4.5.3 wherein we characterize both the optimal competitive ratio and the optimal competitive ratio attainable with polynomial-time algorithms up to a factor of \( 2 - \frac{1}{d} \).

4.5.1 Cost Assumptions on Edge Weights

Let \( H = (V, E, w) \) be the shareability hypergraph for an instance of Online-(k, d)-Min-Matching. Throughout Section 4.5 we make the following assumption on the edge weights in the shareability
CHAPTER 4. ONLINE MATCHING FOR RIDESHARING

Figure 4.3: The minimum cost matching is highlighted in green and blue respectively for instances $H_1$ and $H_2$.

A hypergraph.

**Assumption 1 (Monotonicity).** For any $e \in E$, and any $e' \subseteq e$, we have $e' \in E$ and $w(e') \leq w(e)$.

Assumption 1 is realistic in ridesharing and delivery problems for the following reason. The cost $w(e)$ is the minimum cost required to service the jobs in $e$, i.e., visiting the corresponding pickup and dropoff points. Let $p$ be such a minimum cost path. Note that $p$ also serves all requests in $e'$, and therefore, the minimum cost required to serve $e'$ can only be smaller than $w(e)$. Thus we see that $w(e') \leq w(e)$.

4.5.2 The $k = 2$ Case

We first study the $k = 2$ case where the shareability hypergraph is in fact a shareability graph. After introducing some shorthand notation, we prove a lower bound of $\frac{3}{2}$ on the optimal deterministic competitive ratio. We then complement this result by presenting a $\frac{3}{2}$-competitive deterministic algorithm.

**Notation:** For $i \in V$, we use $w_i \in \mathbb{R}_+$ to denote its weight, i.e., the cost of leaving $i$ unmatched. For $(i, j) \in E$, we use $w(i, j) \in \mathbb{R}_+$ to denote its cost. Given a matching $M$, vertex $i$ is unmatched under $M$ if it is not an endpoint of any edge in $M$. We use $w(M)$ to represent the cost of a matching $M$, defined via $w(M) := \sum_{(i, j) \in M} w(i, j) + \sum_{i \in U} w_i$ where $U$ is the set of unmatched vertices in $M$.

**Lower Bound:** We present a simple lower bound on the competitiveness of deterministic algorithms for Online-$(k, d)$-Min-Matching with $k = 2$.

**Lemma 9 (Lower Bound on the Competitive Ratio).** When $k = 2$, no deterministic algorithm can have a competitive ratio smaller than $\frac{3}{2}$ for Online-$(k, d)$-Min-Matching.

**Proof of Lemma 9.** For the classical online bipartite matching problem, it is known from the seminal paper of [51] that no deterministic algorithm can match more than half of the vertices. We use a similar idea here. Consider two instances $H_1, H_2$ of Online-$(k, d)$-Min-Matching shown in Figure 4.3 where every vertex has weight 1 and all edges also have weight 1.
CHAPTER 4. ONLINE MATCHING FOR RIDE-ShARING

Figure 4.4: The observed graph when $A$ becomes critical under both $H_1$ and $H_2$.

$H_1$ and $H_2$ can be constructed so that $A$ becomes critical before $D$ arrives (see Figure 4.4). As a consequence, when $A$ is critical, it is impossible to determine whether the instance is $H_1$ or $H_2$.

If at this point a deterministic algorithm chooses to match $A$ to $B$, then the resulting matching will be suboptimal if the instance turns out to be $H_1$. Similarly, if the algorithm chooses to match $A$ to $C$, the resulting matching will be suboptimal if the instance turns out to be $H_2$. We conclude by observing that for both $H_1$ and $H_2$, any non-maximum matching will have cost at least $\frac{3}{2}$ of the cost of the maximum matching.

**Algorithms:** We now discuss our algorithmic results for this setting. We present Risk-Threshold and Risk-Threshold-ag as described in Algorithm 6 and Algorithm 5 respectively. These algorithms operate based on a specified threshold parameter $\theta$. Formally, they define a risk score $\theta_{ij} := \frac{w(i,j)}{w_i + w_j}$ for each observed edge $(i, j) \in E$. A high risk score means the edge cost is not much smaller than the cost of the individual vertices. Committing to such an edge gives little reward, but prevents the vertices from matching with more compatible partners that may appear later. For this reason, the algorithms refuse to include edges with risk scores that exceed the specified threshold $\theta$. Risk-Threshold-ag commits to edges with sufficiently low risk scores as soon as they appear, whereas Risk-Threshold waits as long as possible (i.e., until a vertex becomes critical) to decide how to match the vertices. Risk-Threshold has a better competitive ratio, but Risk-Threshold-ag does not require knowledge of $d$, and can be used in the more general case where vertices stay in the system for different and unknown amounts of time.

The motivation for using a thresholding rule is illustrated in the following example.

**Example 1.** Consider the shareability graph depicted in Figure 4.5 where $w_A = 1$, $w_B = x > 1$, and $w(A, B) = x$. There is also a third vertex $C$ with $w_C \in \{0, x\}$ and $w(B, C) = x$.

Suppose $A$ becomes critical before $C$ arrives. In this case, our algorithm needs to decide whether to match $(A, B)$, or wait and match $(B, C)$ instead. The optimal matching depends on $w_C$. In particular, if $w_C = 0$, then the minimum cost is $w(A, B) + w_C = x$ and is achieved by matching $(A, B)$. However, if $w_C = x$, then the minimum cost is $w_A + w(B, C) = 1 + x$. Since we are
interested in the worst-case error, we make the following two observations.

1. If \( w_C = 0 \), and we choose not to match \((A, B)\), then our resulting cost will be \( w_A + w(B, C) = 1 + x \). Recall that the minimum cost is \( x \), so our matching has a cost that is \( \frac{1+x}{1} \) times as large as the minimum cost.

2. If \( w_C = x \) and we choose to match \((A, B)\), then our resulting cost will be \( w(A, B) + w_C = 2x \). Recall that the minimum cost in this case is \( 1 + x \), so our matching has a cost that is \( \frac{2x}{1+x} \) times as large as the minimum cost.

Since our goal is to minimize the worst case performance, this shows that we should match \((A, B)\)

![Figure 4.5: A visualization of the graph in Example 1. A expires before C arrives, so an algorithm needs to decide between matching A to B to collect an immediate reward, or keep B unmatched in case (B, C) turns out to be a more favorable match.](image-url)
if \( \frac{2x}{1+x} \leq \frac{1+x}{x} \), and not match \( A \) otherwise. Thus the optimal deterministic algorithm for this family of instances is a thresholding rule which matches \((A, B)\) if and only if \( x \leq \frac{1}{\sqrt{2}-1} \).

The performance of Algorithms 5 and 6 are described in the following results.

**Theorem 4.** Using \( \theta = \frac{\sqrt{5}-1}{2} \), the competitive ratio of Risk-Threshold-ag(\( \cdot, \frac{\sqrt{5}-1}{2} \)) is \( \frac{\sqrt{5}+1}{2} \approx 1.61 \).

**Theorem 5.** Using \( \theta = \frac{3}{4} \), Risk-Threshold(\( \cdot, \theta \)) is \( \frac{3}{4} \)-competitive.

Theorem 5 in conjunction with Lemma 9 shows that Risk-Threshold has the optimal competitive ratio among deterministic algorithms. We now prove Theorems 4 and 5.

**Proof of Theorem 4.** For a given instance \( H = (V, E, w) \), let \( M^* \) denote a minimum cost matching, and let \( U^* \) be the set of vertices unmatched under \( M^* \). For each \((i, j) \in E\), we denote \( \theta_{ij} := \frac{w(i, j)}{w_i + w_j} \).

For \( \theta \in [\frac{1}{2}, 1] \), define the set \( E_\theta := \{(i, j) \in E : \theta_{ij} \leq \theta\} \). Next, let \( M \) be a matching produced by Risk-Threshold-ag(\( H, \theta \)), and let \( U \) be the set of unmatched vertices under \( M \). Note that \( M \) is a maximal matching in \( (V, E_\theta, w) \). To compare \( w(M) \) to \( w(M^*) \), we assign to each vertex a score \( v \) according to \( M \) by the following rule:

\[
v_i := \begin{cases} 
  w_i & \text{if } i \in U \\
  \theta_{ij} w_i & \text{if } (i, j) \in M \text{ for some } j \in V 
\end{cases}
\]

Note that by construction, \( \sum_{i \in V} v_i = w(M) \). One way to interpret \( v \) is that \( v_i \) is the contribution of vertex \( i \) to the total cost of the matching. With this notation, we can now write

\[
w(M) = \sum_{i \in V} v_i = \sum_{(i, j) \in M^*} (v_i + v_j) + \sum_{i \in U^*} v_i = \sum_{(i, j) \in M^* \cap E_\theta} (v_i + v_j) + \sum_{(i, j) \in M^* \setminus E_\theta} (v_i + v_j) + \sum_{i \in U^*} v_i,
\]

where \((a)\) is due to the fact that the set of matched nodes under \( M^* \) and the set of unmatched nodes under \( M^* \) form a partition of \( V \). For term 1, note that \( M^* \cap E_\theta \) is a matching in \((V, E_\theta, w)\). Since \( M \) is a maximal matching in \((V, E_\theta, w)\), this means that for every \((i, j) \in M^* \cap E_\theta\), at least one of \( i \) or \( j \) is also matched in \( M \). Without loss of generality, suppose \( i \) is matched. Then we have

\[
v_i + v_j \leq \theta_{ij} w_i + w_j \leq \theta_{ij} \max(w_i, w_j) + \max(w_i, w_j) \leq (1 + \theta) \max(w_i, w_j) \leq (1 + \theta) w(i, j).
\]

Where \((a)\) is due to Assumption 1. For term 2, note that \((i, j) \notin E_\theta \) means that \( \theta_{ij} > \theta \). Thus for any \((i, j) \in M^* \setminus E_\theta \), we have:

\[
v_i + v_j \leq \frac{1}{\theta_{ij}} w(i, j) = \frac{1}{\theta} w(i, j).
\]
Finally, for term 3, note that $v_i \leq w_i$ is always true. Putting these observations together, we see that:

$$w(M) = \sum_{(i,j) \in M \cap E_a} (v_i + v_j) + \sum_{(i,j) \in M^* \setminus E_a} (v_i + v_j) + \sum_{i \in U^*} v_i$$

$$\leq \sum_{(i,j) \in M \cap E_a} (1 + \theta)w(i,j) + \sum_{(i,j) \in M^* \setminus E_a} \frac{1}{\theta}w(i,j) + \sum_{i \in U^*} w_i$$

$$\leq \max \left(1 + \theta, \frac{1}{\theta}\right) \sum_{(i,j) \in M \cap E_a} w(i,j) + \max \left(1 + \theta, \frac{1}{\theta}\right) \sum_{(i,j) \in M^* \setminus E_a} w(i,j) + \max \left(1 + \theta, \frac{1}{\theta}\right) \sum_{i \in U^*} w_i$$

$$= \max \left(1 + \theta, \frac{1}{\theta}\right) \left( \sum_{(i,j) \in M \cap E_a} w(i,j) + \sum_{(i,j) \in M^* \setminus E_a} w(i,j) + \sum_{i \in U^*} w_i \right) = \max \left(1 + \theta, \frac{1}{\theta}\right) w(M^*).$$

Thus Risk-Threshold($\cdot, \theta$) is a max$(1 + \theta, \frac{1}{\theta})$-competitive algorithm. To finish, we now observe that this value is minimized when $\theta = \frac{\sqrt{5} - 1}{2}$, which implies that Risk-Threshold($\cdot, \frac{\sqrt{5} - 1}{2}$) is a $\frac{\sqrt{5} + 1}{2}$-competitive algorithm for Online-$(k,d)$-Min-Matching when $k = 2$. 

**Proof of Theorem 3** For a given instance $H = (V, E, w)$, let $M^*$ be a minimum cost matching, and let $M$ be a matching produced by Risk-Threshold. We use $U^*$ and $U$ to denote the vertices unmatched under $M^*$ and $M$ respectively. For each $(i, j) \in E$, we denote $\theta_{ij} := \frac{w(i,j)}{w_i + w_j}$. Similar to the proof of Theorem 4, we define costs $v^*, v$ to vertices for $M^*$ and $M$ respectively

$$v_i^* := \begin{cases} w_i & \text{if } i \in U^* \smallskip \
\theta_{ij} w_i & \text{if } (i, j) \in M^* \text{ for some } j \in V \end{cases} \quad \text{and} \quad v_i := \begin{cases} w_i & \text{if } i \in U \smallskip \
\theta_{ij} w_i & \text{if } (i, j) \in M \text{ for some } j \in V \end{cases}$$

so that $w(M^*) = \sum_{i \in V} v_i^*$ and $w(M) = \sum_{i \in V} v_i$. The edges in $M^* \cup M$ form a disjoint collection of cycles and maximal paths $\{C_k\}_{k=1}^{m_c}$, $\{P_j\}_{j=1}^{m_p}$. If we can prove

$$\sum_{i \in E'} v_i \leq \frac{3}{2} \sum_{i \in E'} v_i^* \quad \text{for every component } E' \in \{C_k\}_{k=1}^{m_c} \cup \{P_j\}_{j=1}^{m_p} \quad (4.2)$$

then the main result follows:

$$w(M) = \sum_{i \in V} v_i = \sum_{i \in U^* \cup U} v_i + \sum_{k=1}^{m_c} \sum_{i \in C_k} v_i + \sum_{j=1}^{m_p} \sum_{i \in P_j} v_i$$

$$\leq \sum_{i \in U^* \cup U} v_i + \sum_{k=1}^{m_c} \frac{3}{2} \left( \sum_{i \in C_k} v_i^* \right) + \sum_{j=1}^{m_p} \frac{3}{2} \left( \sum_{i \in P_j} v_i^* \right)$$

$$\leq \frac{3}{2} \left( \sum_{i \in U^* \cup U} v_i^* + \sum_{k=1}^{m_c} \sum_{i \in C_k} v_i^* + \sum_{j=1}^{m_p} \sum_{i \in P_j} v_i^* \right) = \frac{3}{2} w(M^*).$$
To this end, we first show that (4.2) holds for cycles. Note that a cycle in \( M^* \cup M \) must have even length (otherwise there would be a node incident to two edges from the same matching). In particular, every node in the cycle is matched under \( M \), so we conclude that

\[
\sum_{i \in C} v_i^{(a)} \leq \sum_{i \in C} \frac{2}{3} w_i = \frac{4}{3} \sum_{i \in C} \frac{1}{2} w_i \leq \frac{4}{3} \sum_{i \in C} v_i^* \leq \frac{3}{2} \sum_{i \in C} v_i^*.
\]

We concluded (a) from the fact that Risk-Threshold adds an edge \((i, j)\) to \( M \) only if \( \theta_{ij} \leq \frac{2}{3} \), and (b) from the fact that \( \theta_{ij} \geq \frac{1}{2} \) for every edge, due to Assumption [1].

We establish (4.2) for maximal paths by induction where we make use of an intermediate notion of cost:

**Definition 10 (Restricted Cost).** Given a graph \( G = (V, E, w) \), an edge set \( E' \subset E \) and a matching \( M \), we define \( w(M; E') \) to be the cost of \( M \cap E' \) in the subgraph \((V(E'), E', w)\).

We prove the following relationship between restricted costs of \( M \) and \( M^* \) on paths in \((V, M^* \cup M, w)\).

**Lemma 10 (Path cost bound).** For any (possibly nonmaximal) path \( P \subset (V, M \cup M^*, w) \), we have:

\[
w(M; P) \leq \frac{3}{2} w(M^*; P).
\]

See Appendix C.1 for a proof of Lemma 10. The result (4.2) for maximal paths follows from the observation that \( w(M; P) = w(M \cap P) = \sum_{i \in P} v_i \) and \( w(M^*; P) = w(M^* \cap P) = \sum_{i \in P} v_i^* \) if \( P \) is a maximal path.

### 4.5.3 The \( k > 2 \) Case

**Section Outline:** In this section, we present a \((2 - \frac{1}{k})\)-competitive algorithm for \( \text{Online-}(k, d)\)-\text{Min-Matching} \) which runs in exponential time, and a \((2 - \frac{1}{k})\log k\)-competitive algorithm which runs in polynomial time. We further prove that no polynomial-time algorithm can have a competitive ratio better than \( \log k - O(\log \log k) \) whenever \( d = n^{\Omega(1)} \). To these ends, we first describe \( \text{Online-}(k, d)\)-\text{Min-Matching} \) as an online weighted set cover problem. We then show that any offline set cover algorithm can be converted into an algorithm for \( \text{Online-}(k, d)\)-\text{Min-Matching} \) using Randomized-Batching by inflating its approximation ratio by \( 2 - \frac{1}{k} \). We inherit the \( \log k - O(\log \log k) \) hardness result from the unweighted set cover by proving that it is a special case of \( \text{Online-}(k, d)\)-\text{Min-Matching}.

First we show that \( \text{Online-}(k, d)\)-\text{Min-Matching} \) is an online weighted set cover problem with universe \( V \), subsets \( E \subset 2^V \), and weight function \( w \). To do this, we establish a bijection between the feasible sets of these problems which preserves the objective value. For any matching \( M \), let \( U \)
be the vertices unmatched by $M$. The cost of $M$ for \texttt{Online-}$(k, d)$-\texttt{Min-Matching} is $\sum_{e \in M} w(e) + \sum_{i \in U} w(\{i\})$. Since $\{i\} \in E$ for all $i \in V$, $S := M \cup \{\{i\}\}_{i \in U}$ is a cover of $V$ using sets in $E$. Furthermore, the costs of $M$ and $S$ for their respective problems is the same. Conversely, for any set cover $S$, if the sets in $S$ are not mutually disjoint, then by Assumption 1, there exists a cover with smaller weight. Therefore assume without loss of generality that all sets in $S$ are disjoint. Then $S$ is a matching for $H$, and its cost for \texttt{Online-}$(k, d)$-\texttt{Min-Matching} is the same as its cost for weighted set cover.

Throughout this section, we use $k$-WSC to denote the weighted set cover problem where each set in the system has cardinality at most $k$, and $k$-SC to be the special case where the weight of all sets is 1. \texttt{Online-}$(k, d)$-\texttt{Min-Matching} is thus an online version of $k$-WSC. See Appendix A.2 for further information on the set cover problem.

Given this connection to set cover, a natural approach to \texttt{Online-}$(k, d)$-\texttt{Min-Matching} is \texttt{Randomized-Batching}$(\cdot, A)$ described in Algorithm 3 where now $A$ is an algorithm for $k$-WSC. The following theorem shows that any $k$-WSC algorithm can be converted into an \texttt{Online-}$(k, d)$-\texttt{Min-Matching} algorithm via \texttt{Randomized-Batching} by inflating its approximation ratio by $2 - \frac{1}{d}$.

**Theorem 6.** Let $A$ be an algorithm for $k$-WSC with approximation ratio $\rho(A)$. Then \texttt{Randomized-Batching}$(\cdot, A)$ is $(2 - \frac{1}{d}) \rho(A)$-competitive for \texttt{Online-}$(k, d)$-\texttt{Min-Matching}.

**Proof of Theorem 6.** Let $H = (V, E, w)$ be a shareability hypergraph and $M^*$ be a minimum weight set cover for $H$. Our proof has three key steps. First, we show that any $e \in E$ can intersect at most two batches. Second, we use this observation to construct a random set cover $\tilde{M}$ such that all of its sets are visible with respect to the batches, and $\mathbb{E}[w(\tilde{M})] \leq (2 - \frac{1}{d})w(M^*)$. Third, we prove that if such a $\tilde{M}$ exists, then the algorithm will output a matching $M$ with $w(M) \leq \rho(A)w(\tilde{M})$.

Define $\tilde{M}_x = \bigcup_{i=1}^{n/d} \tilde{M}_{x,i}$ where

$$\tilde{M}_{x,i} := \{e \cap V_{x,i} : e \in M^*\}.$$  

**Observation 2.** Note that since $H$ is a shareability hypergraph, for any $e \in E$, $\sum_{i=0}^{n/d} 1[e \cap V_{x,i} \neq \emptyset] \leq 2$, i.e., every edge $e \in E$ intersects at most two batches.

**Proof of Observation 2.** We can show this easily by contradiction. Suppose there exists $e \in E$ that intersects more than two batches. Then define

$$j = \max \{k : e \cap V_{x,k} \neq \emptyset\}$$

$$i = \min \{k : e \cap V_{x,k} \neq \emptyset\}.$$  

It must be the case that $i + 1 < j$. If this is not the case, i.e., if $j = i + 1$, then $e$ intersects $V_{x,k}$ only if $k \in \{i, i + 1\}$, and thus it does not intersect more than two batches. Therefore, we know
that $i + 1 < j$. Finally, note that for any $a \in V_{z,i}, b \in V_{z,j}$, we have $b > a + d$. This implies that $\text{diam}(e) > d$, which is a contradiction to the fact that $\text{diam}(e) < d$ promised by $H$ being a shareability hypergraph.

Now consider the set cover produced by $\text{Randomized-Batching}(\cdot, A)$ given $Z = z$. This procedure computes a set cover $M$ as $\bigcup_{i=1}^{n/d} M_{z,i}$, where $M_{z,i}$ is a set cover of $W_{z,i}$ chosen by $A$. We thus see that

$$
\mathbb{E}[w(M)|Z = z] = \sum_{i=0}^{n/d} \mathbb{E}[w(M_{z,i})|Z = z]
$$

$$
\leq \sum_{i=0}^{n/d} \rho(A) w(M_{z,i})
$$

$$
= \rho(A) \sum_{i=0}^{n/d} \sum_{e \in M_{z,i}} w(e \cap V_{z,i})
$$

$$
\leq \rho(A) \sum_{e \in M^*} w(e) \sum_{i=0}^{n/d} \mathbb{1}[e \cap V_{z,i} \neq \emptyset],
$$

where $(a)$ is because $A$ is a $\rho(A)$-approximation $k$-WSC algorithm and $M_{z,i}$ is a set cover for $V_{z,i}$. We now take expectation over $Z$ to bound $\mathbb{E}[w(M)]$:

$$
\mathbb{E}[w(M)] = \mathbb{E}_Z[\mathbb{E}[w(M)|Z = z]]
$$

$$
\leq \mathbb{E}_Z \left[ \rho(A) \sum_{e \in M^*} w(e) \sum_{i=0}^{n/d} \mathbb{1}[e \cap V_{z,i} \neq \emptyset] \right]
$$

$$
= \rho(A) \sum_{e \in M^*} w(e) \mathbb{E}_Z \left[ \sum_{i=0}^{n/d} \mathbb{1}[e \cap V_{z,i} \neq \emptyset] \right]
$$

For any $e \in E$, let $i$ denote its earliest vertex. Note that when $Z = i \mod d$, $\sum_{i=0}^{n/d} \mathbb{1}[e \cap V_{z,i} \neq \emptyset] = 1$. Therefore, $P \left( \sum_{i=0}^{n/d} \mathbb{1}[e \cap V_{z,i} \neq \emptyset] = 1 \right) \geq \frac{1}{2}$. Since we already showed that $\sum_{i=0}^{n/d} \mathbb{1}[e \cap V_{z,i} \neq \emptyset]$ is always either 1 or 2, this means that the probability it is 2 is at most $1 - \frac{1}{2}$. Applying this gives:

$$
\mathbb{E}[w(M)] \leq \rho(A) \sum_{e \in M^*} w(e) \mathbb{E}_Z \left[ \sum_{i=0}^{n/d} \mathbb{1}[e \cap V_{z,i} \neq \emptyset] \right]
$$

$$
\leq \rho(A) \sum_{e \in M^*} w(e) \left[ 2 \left( 1 - \frac{1}{d} \right) + \frac{1}{d} \right]
$$

$$
= \left( 2 - \frac{1}{d} \right) \sum_{e \in M^*} w(e) = \left( 2 - \frac{1}{d} \right) \rho(A) w(M^*).
The $k$-WSC problem has been studied extensively in the computer science literature. [17] proposed a polynomial-time greedy algorithm and proved that it is a $(1 + \log k)$-approximation. It was shown in Section 3 of [82] that this performance is nearly optimal in polynomial-time by proving that it is NP-hard to have an approximation ratio smaller than $\log k - O(\log \log k)$. The following corollaries are straightforward implications of Theorem 6.

**Corollary 3.** Randomized-Batching $(\cdot, \mathcal{A})$ is $(2 - \frac{1}{d})(1 + \log k)$-competitive when $\mathcal{A}$ is the greedy algorithm from [17].

**Corollary 4.** Randomized-Batching $(\cdot, \mathcal{A})$ is $(2 - \frac{1}{d})$-competitive if $\mathcal{A}$ computes a minimum weight set cover, i.e., if $\rho(\mathcal{A}) = 1$.

The following lemma shows that the result of Corollary 4 is optimal within a factor of $2 - \frac{1}{d}$.

**Lemma 11.** It is NP-hard to have a competitive ratio better than $\log k - O(\log \log k)$ for Online-$(k, d)$-Min-Matching if $d = n^\Omega(1)$.

**Proof of Lemma 11** We prove the desired result by a reduction to $k$-SC. Concretely, we show that a polynomial-time $\alpha$-competitive algorithm for Online-$(k, d)$-Min-Matching can be used to construct a polynomial-time $\alpha$-approximation for $k$-SC instances where the universe has $d$ elements. With this reduction at hand, we appeal to the result of [82] which states that it is NP-hard to approximate $k$-SC better than $\log k - O(\log \log k)$. Since $d = n^\Omega(1)$, it is NP-hard to achieve a competitive ratio smaller than $\log k - O(\log \log k)$ for Online-$(k, d)$-Min-Matching. Thus all that remains is to establish a reduction from Online-$(k, d)$-Min-Matching to $k$-SC.

To prove the reduction from Online-$(k, d)$-Min-Matching to $k$-SC, consider any instance $(X, S)$ of $k$-SC where the goal is to find a minimum cardinality cover of the set $X$ by sets in $S$. Construct a hypergraph $H := (V, E, w)$ where $V := X$ and $E := \overline{S}$ is the downward closure of $S$, which means $E$ contains all subsets of all sets in $S$. Finally, $w(e) := 1$ for every $e \in E$. Note that by setting $E = \overline{S}$, $H$ satisfies Assumption 1.

First note that the cost of a minimum cost matching of $H$ is a lower bound on the minimum cardinality set cover of $X$ by $S$. To see this, for any cover $s_1, s_2, \ldots, s_m$ of $X$ by sets in $S$, define $e_i := s_i \setminus \bigcup_{j < i} s_j$. By construction, $e_1, \ldots, e_m$ form a partition of $X$. Note that $e_i$ is a subset of $s_i$, so by downward closure, $e_1, \ldots, e_m$ are all in $E$, meaning $H$ has a matching of cost $m$.

Next we show that any matching $M := \{e_1, \ldots, e_m\}$ in $H$ can be converted into a cover of $X$ by sets in $S$ so that the cover’s cardinality is at most the cost of $M$. Since $E$ is the downward closure of $S$, $e_i \in E$ implies that there exists some $s \in S$ for which $e_i \subset s$. Let $s_i$ be any set in $S$ so that $e_i \subset s_i$. Finding $s_i$ takes $O(|S|)$ time, since we can simply check each member of $S$ to see if it contains $e_i$. Since the $e_i$’s are disjoint and non-empty, there can be at most $|X|$ of them, meaning
that the construction of $\{s_1, \ldots, s_m\}$ will take at most $O(|X| \cdot |S|)$ time, which is polynomial in the size of the input. Then $\{s_1, \ldots, s_m\}$ is a cover of $X$ using sets in $S$ of cardinality $m = \sum_{i=1}^{m} w(e_i)$.

Putting this all together, suppose there is a polynomial-time $\alpha$-competitive \textsc{Online-}$(k, d)$-\textsc{Min-Matching} algorithm, which we denote $A_\alpha$. For any instance $(X, S)$ of $k$-$SC$, construct $H$ as described above, and apply $A_\alpha$ to $H$ to obtain a matching $M := \{e_1, \ldots, e_m\}$. The cost of $M$ is at most $\alpha$ times the cost of the minimum cardinality set cover of $X$ by $S$, since the cost of a minimum cost matching is a lower bound on the minimum cardinality set cover. Finally, construct from $M$ a set cover whose cardinality is the same as the cost of $M$, which gives a set cover whose cardinality is at most $\alpha$ times the minimum cardinality. This establishes the desired reduction and thus completes the proof.

\begin{flushright}
$\square$
\end{flushright}

### 4.6 Discussion

In this chapter we studied online hypergraph matching with delays inspired by ridesharing and delivery problems. We studied both the utility maximization and cost minimization variants of the problem. For utility maximization, we presented a polynomial-time randomized batching algorithm and proved that it attains the optimal competitive ratio. We studied cost minimization under a monotone cost assumption, which is realistic for ridesharing and delivery problems. For $k = 2$ we introduced a thresholding algorithm and proved that it achieves the optimal competitive ratio for deterministic algorithms. For $k > 2$, we characterize both the optimal competitive ratio and the optimal polynomial-time competitive ratio up to a factor of $2 - \frac{1}{d}$.

There are several interesting directions for future work. With regards to \textsc{Online-}$(k, d)$-\textsc{Max-Matching}, adding additional application-specific assumptions could enable better approximation guarantees. For \textsc{Online-}$(k, d)$-\textsc{Min-Matching}, closing the $(2 - \frac{1}{d})$ gap between the achievability and impossibility results for $k > 2$ and finding a randomized algorithm that beats the $\frac{3}{2}$ threshold for $k = 2$ would both be interesting. For both models, extensions to heterogeneous and unobserved willingness to wait for $k > 2$ would provide compelling insight for practitioners and real-world deployment.

The next chapter will discuss how to achieve better competitive ratios in rider-driver matching through side information. In particular, we will study a ridehailing system with rider reservations where riders can submit their transportation needs to the system ahead of time, thereby giving the system partial information about the future demand. We will show that this information can be leveraged to attain better performance guarantees than what is achievable in classical online models in which side information is unavailable.
Chapter 5

Online Matching with Reservations

The previous chapter studied ridesharing systems from a worst-case perspective motivated by situations where demand prediction is difficult, e.g., settings with non-stationary demand, or systems with limited training data. This chapter explores techniques to improve performance guarantees in settings where demand prediction is difficult. In particular, we study ridehailing systems with a rider reservation feature. Rider reservations allow riders to inform the system of their travel needs ahead of time, thereby giving the system some information about future demand. The rider reservation model is a generalization of the classical worst-case online problem (i.e., no riders reserve) and the offline problem where all demand is known in advance (i.e., all riders reserve) and attempts to interpolate between them to achieve better performance guarantees than the online case, while being having a more realistic information model than the offline case.

5.1 Introduction

A need for decision making under uncertainty in algorithm design for mobility systems stems from the on-demand nature of mobility services. While performance guarantees can be established using worst-case analysis, such results do not adequately incorporate domain knowledge. A popular approach for incorporating domain knowledge into online decision making problems is through demand forecasts whereby future demand is estimated based on historical data. This approach has seen success in various applications but it is not well suited to rare events or other situations where data is scarce. For example, leading up to the Tokyo 2020 Olympics, the city had only ever hosted the Olympics one time in 1964. Furthermore, the city’s economy and infrastructure have changed significantly in those 56 years, making the 1964 data a poor estimation of the 2020 activity.

Another approach to handle unpredictable demand is to negotiate with the riders, i.e., ask them to report their travel needs ahead of time in exchange for better service. Indeed, in the leadup to the Tokyo Olympics, the Japanese authorities coordinated and incentivized citizens to cooperate and
help minimize congestion during the games. Lyft’s shared saver plan also aimed to offer even lower prices to customers, provided they were willing to book in advance and adjust their pickup location and pickup time \[48\]. In this chapter, we will study the potential benefits of such cooperation between a ridehailing system and its riders.

5.1.1 Statement of Contributions

In this work, we study a setting whereby riders can help a ridehailing system by sharing their transportation needs ahead of time and receive improved service in return. Specifically, we study driver-rider assignment in ridehailing systems through the lens of an Online Bipartite Matching problem with random reservations where a random set of riders will reserve their rides ahead of time. We present a Reservation Priority matching algorithm and prove that its performance (competitive ratio) monotonically interpolates between that of the optimum online and optimum offline algorithms as the fraction of reservations increases from 0 to 1.

5.1.2 Organization

This chapter is organized as follows. We present a model for a ridehailing system with a reservation feature in Section 5.2. Section 5.3 describes literature related to our model. We present the Reservation Priority algorithm and discuss its performance guarantees in 5.4. We conclude this chapter and discuss directions for future work related to this problem in 5.5.

5.2 Model

The Online Bipartite Matching with Random Reservations (OBMRR) has one parameter \(p \in [0, 1]\), so we use \(\text{OBMRR}(p)\) to specify the problem for a specific value of \(p\). A problem instance is a bipartite graph \(G = (A, B, E)\) which contains a perfect matching\[1\] We denote \(n := |A| = |B|\). The vertices in \(A := \{a_1, ..., a_n\}\) are the offline vertices, and are known to the algorithm. The vertices in \(B := \{b_1, ..., b_n\}\) are the online vertices which are not initially known to the algorithm. The operation period consists of \(n\) timesteps. Before the first timestep, since the requests in \(R\) were reserved in advance, all edges between \(A\) and \(R\) are revealed to the algorithm. In the \(t\)th timestep, the neighbors of \(b_t\) are revealed (if they are not already known, i.e. if \(b_t\) did not reserve), and the algorithm must irrevocably decide to either leave \(b_t\) unmatched, or match it to one of its

\[1\] This is the worst case scenario in the following sense: If \(G\) does not have a perfect matching, then there is either excess supply or excess demand that can help the algorithm but does not help the offline optimum.
neighbors in $A$ that has not yet been matched. The objective is to maximize the cardinality of the obtained matching.

For a given OBMRR($p$) problem instance $G = (A, B, E)$, we use $(G, R)$ to denote the realization where the reserve vertices and non-reserve vertices are $R$ and $B \setminus R$ respectively. We will evaluate algorithms based on their competitive ratio, defined similarly to Section 4.3.2.

**Definition 11 (Competitive Ratio for OBMRR($p$)).** The competitive ratio of a OBMRR($p$) algorithm $A$ is denoted $\rho(A)$ and is given by

$$\rho(A) := \sup_{G=(A,B,E)} \frac{\mathbb{E}[A(G, R)]}{\text{OPT}_G}$$

where the expectation is over both the randomness of $R$ and the randomness of the algorithm $A$. $\text{OPT}_G$ is the size of the maximum matching in $G$.

**Remark 4 (Application to Driver-Rider Matching in Ridehailing Systems).** One motivation for studying OBMRR comes from driver-rider matching in ridehailing systems. In such systems, the offline vertices $A$ represent the set of available drivers on a ridehailing service. The online vertices $B$ represent customers who request transit in real-time. Since the ridehailing system only learns a rider’s travel needs when they hail a ride, a rider’s compatibility with the drivers is only observed at this time. This is modeled by the fact that the neighborhoods of online vertices are revealed sequentially in the OBMRR model.

**Remark 5 (An economic interpretation of independent reservations).** In the OBMRR($p$) problem, every online vertex $b \in B$ will reserve its request with probability $p$. Such a property is a consequence of i.i.d. private valuations in economic literature [30]. Specifically, we can make a standard assumption that the online vertices $b_1, ..., b_n$ have private valuations $v_1, ..., v_n$ for the ridehailing service which are independently and identically distributed according to a distribution $\mathcal{P}$. Further suppose $c \geq 0$ is the inconvenience incurred by making a reservation (e.g., loss in flexibility). If $q$ is the prior probability that an online vertex will be matched, i.e., what the vertex believes its chances of being matched is, $b_i$ will reserve if and only if $v_i - c \geq v_i q$. Hence

$$p := \mathbb{P}_{v_i \sim \mathcal{P}} \left[ v_i \geq \frac{c}{1 - q} \right].$$

**Remark 6 (Classical Problems as special cases of OBMRR).** The classical online bipartite matching problem is a special case of OBMRR with $p = 0$. The offline bipartite matching problem is also a special case of OBMRR where $p = 1$. This model thus aims to study a middle ground between these classical settings where some incomplete information about the graph $G$ is known in advance.

2When using a greedy matching algorithm, it is known that at least $\frac{1}{2}$ of the online vertices will be matched. When using the RANKING algorithm from [51], it is known that on average at least $1 - \frac{1}{e}$ online vertices will be matched. Hence $\frac{1}{2}$ or $1 - \frac{1}{e}$ would be reasonable values for $q$. Alternatively, $q$ can be estimated from historical data.
5.2.1 Notation

Throughout this chapter, we will be using the following notation regarding graphs and sets.

- \( N(v) \) is the set of neighbors of \( v \) in a graph.
- Given a matching \( M \) and a vertex \( v \) matched by \( M \), we use \( M(v) \) to denote the partner of \( v \) under \( M \) (if \( v \) is unmatched, then \( M(v) = \emptyset \)). For a set of vertices \( V \), \( M(V) \) is the set of all partners of vertices in \( V \).
- Given two subsets \( A, B \) of a ground set \( X \), we use \( A \triangle B := (A \setminus B) \cup (B \setminus A) \) to denote their symmetric difference.

5.3 Related Work

This chapter is related to the classical problem of online bipartite matching \([51]\). In this model, the problem instance is a bipartite graph, where the vertices are partitioned into offline and online vertices. The offline vertices represent resources and the online vertices represent requests. The online vertices are revealed sequentially, along with their incident edges, and the algorithm must decide to either irrevocably match the online vertex with an offline neighbor, or to leave it unmatched. Only after a decision is made will the next online vertex appear, and the goal is to obtain a matching of large cardinality. This problem is thus a sequential optimization with imperfect information, since the neighborhoods of later online vertices are not observed when making irrevocable decisions for earlier online vertices. In terms of competitive analysis, it is known that the best possible competitive ratio that a deterministic algorithm can have is \( \frac{1}{2} \), and the best possible competitive ratio that a randomized algorithm can have is \( 1 - \frac{1}{e} \), which is achieved by the RANKING algorithm from \([51]\). The OBMRR model generalizes the online bipartite matching model by incorporating information on future vertices through reservations.

The reservation aspect of OBMRR is related to multi-staged matching problems, since reserved requests are observed before non-reserved requests. Specifically, in OBMRR, \( R \) can be viewed as the first stage, and \( B \setminus R \) can be viewed as a second stage. In multi-staged matching problems, a problem instance is a graph \( G = (V, E) \) whose edges are partitioned into several subsets \( E_i = \bigcup_{i=1}^{n} E_i \). The goal is to compute a large cardinality matching, or a matching with large weight in the case of weighted graph instances. The problem has \( n \) stages where \( E_i \) is revealed in stage \( i \), and the algorithm can irrevocably choose some edges in \( E_i \) to add to its matching. However, edges in \( E_i \) cannot be added to the matching after stage \( i \).

Existing work on multi-staged matching choose edges guided by hypermatchability to hedge against uncertainty of future stages \([55, 56, 30]\). At a high level, in each stage of the problem, these works use a convex program based on the available edges \( E_i \) to design an edge sampling scheme, known as a matching skeleton. The edges sampled by this procedure will then be added...
to the matching. The matching skeleton is designed to be “fair” in the following sense: any two nodes should have a similar probability of being matched. Since every vertex has a sufficiently large probability of being matched, the authors can establish lower bounds on the probability of favorable events, i.e., the probability that an edge in the optimal matching is chosen by the algorithm.

Matching skeletons were first studied in streaming online bipartite matching problems [34]. Interestingly, the matching skeletons in these works are related to the Edmonds-Gallai Decomposition [26] found in the Blossom graph matching algorithm. Streaming problems study settings where the problem instance is too large to fit in memory, so algorithms can scan the data, but due to memory constraints, cannot store all of the data and must instead compress or summarize the observations in a representation that is helpful for the task, e.g., computing a matching in a very large graph. Streaming problems are similar to online decision making problems, but there are several important differences. Streaming algorithms often have memory constraints, and sometimes allow multiple passes over the data, whereas online algorithms see the data at most one time (i.e., multiple passes are not allowed), and consider memory constraints less often.

Our work is perhaps most related to [54], who study a semi-online bipartite matching problem. In their model, the bipartite graph is partitioned into an offline component which is initially known, and an online component which is revealed over time. The goal is to compute a large cardinality matching, where edge selection is irrevocable, as standard in online algorithms literature. They study this semi-online bipartite matching problem from a worst case perspective: both the graph and the partition into online and offline components is arbitrary, and the authors measure algorithm performance using competitive ratio. The primary tool in this work is also a matching skeleton inspired by [34].

Semi-online bipartite matching bears some similarities to OBMRR. The offline component of semi-online bipartite matching represents initial knowledge of the system, which can model the information on reserved requests for a ridehailing system, i.e. the vertices $R$ and their neighborhoods. The online component can then model the arrival of non-reserved requests, i.e. vertices in $B \setminus R$. However, there are two key differences between the model studied in [54] and ridehailing with reservations. First, the algorithm from [54] commits to a matching for the offline component before any of the online vertices arrive. However in ridehailing systems, the reserved requests do not need to be assigned immediately; they only need to be assigned by the specified pickup time. Therefore there is no need to match reserved requests immediately. Instead, the preferences of reserved requests should be used to more effectively decide how to match non-reserved requests. Second, [54] studies an adversarial partition of the graph into offline and online components. We are instead exploring a random partition of the graph, which may be more realistic than adversarial partitions.

In light of these observations, we will compare the performance guarantees we obtain for OBMRR to those of [54] to see if non-immediate assignment of reserved vertices and random partitions can

---

3The sampling is done so that sampled edges do not share any endpoints, and hence the set of sampled edges is a matching.
lead to better competitive ratios.

5.4 The Reservation Priority Algorithm

We present the Reservation-Priority algorithm described in Algorithm 7. As the name suggests, Reservation-Priority will always make sure that there are enough offline vertices so that each reserve vertex can be matched. To this end, the following definition will be important to the algorithm.

Definition 12 (Conflicts). Given a matching $M$ for $G$ and a set $R \subset B$, we say an edge $(a, b)$ is in conflict with $(M, R)$ if there does not exist a matching $M' \supset (M \cup \{(a, b)\})$ for $G$ which matches every vertex in $R$.

Reservation-Priority begins by choosing an arbitrary ranking $\pi : A \rightarrow [n]$ of offline vertices so that $\pi(a)$ is the ranking of vertex $a$. As online vertices arrive, the algorithm maintains a tentative matching for all vertices in $R$. When $b \in R$ arrives, it is matched according to the tentative matching. When $b \notin R$ arrives, the algorithm will only consider matching $b$ to vertices $a \in N(b)$ if there is still a way to match all vertices in $R$ even after adding $(a, b)$ to the matching, i.e., if $(a, b)$ is not in conflict with the edges already included into the matching. If there are multiple acceptable candidates, i.e. $\{a : (a, b) \text{ is not in conflict with the current matching}\}$ has more than one element, then $b$ is matched to the candidate with the highest rank according to $\pi$.

Remark 7. Since conflict edges are never added to the matching and $G$ has a perfect matching, we can conclude that Reservation-Priority matches every vertex in $R$.

Remark 8. Reservation-Priority is a deterministic algorithm in the sense that its output is completely determined by the instance $G$ and the reservation set $R$.

Algorithm 7: Reservation-Priority

1. **Input:** Bipartite Graph $G = (A, B, E)$, reserve vertices $R \subset B$;
2. **Output:** Matching $M$;
3. $M_0 \leftarrow \emptyset$;
4. Chose an arbitrary ranking $\pi$ for the vertices in $A$;
5. for $1 \leq t \leq n$ do
6.   \[ N_{R,t}(b_t) := \{a \in N(b_t) : a \text{ is unmatched in } M_{t-1}, (a, b_t) \text{ does not conflict with } (M_{t-1}, R)\} ;\]
7.   if $N_i(b_t) \neq \emptyset$ then
8.     $a \leftarrow \max_{u \in N_i(b_t)} \pi(u)$;
9.     $M_t \leftarrow M_{t-1} \cup \{(a, b_t)\}$;
10. Return $M_n$
5.4.1 Performance Guarantees for Reservation Priority

In this section we show that the competitive ratio of Reservation-Priority for $\text{OBMRR}(p)$ is exactly $\frac{1}{2} - \frac{1}{p}$. To this end, we first show in Theorem 7 that Reservation-Priority is $\frac{1}{2} - \frac{1}{p}$-competitive, i.e., the expected size of the matching it produces is always at least $\frac{1}{2} - \frac{1}{p}$ times the size of the maximum matching. In Theorem 8 we show that the analysis of Theorem 7 is tight by constructing a worst case graph on which Reservation-Priority produces a matching with expected size no greater than $\frac{1}{2} - \frac{1}{p}$ times the size of the graph’s maximum matching.

To reason about the size of the matchings produced by Reservation-Priority, we introduce the following notation to count the number of offline vertices that are matched by the algorithm.

**Definition 13 (Matched Offline Vertices).** For a $\text{OBMRR}(p)$ instance $G = (A, B, E)$, let $A_R$ denote the subset of $A$ that is matched when Reservation-Priority is applied to $(G, R)$.

A key property in establishing performance guarantees for Reservation-Priority is a notion of monotonicity. In Lemma 12 we show that Reservation-Priority has a monotone property: If an offline vertex $a$ is matched under the realization $(G, R)$, then by adding more reserved vertices will not cause $a$ to become unmatched, i.e., $a$ will also be matched in the realization $(G, R')$ for any $R'$ that contains $R$.

**Lemma 12 (Monotonicity of Reservations).** If $R \subset R'$, then $A_R \subset A_{R'}$.

See Appendix D.2 for a proof of Lemma 12. The permutation $\pi$ used by the algorithm correlates the preferences of the online vertices, since online vertices will try to match with the highest ranking acceptable neighbor. This correlation is needed to establish monotonicity, and as we will see in the next section, enable improved performance guarantees by randomizing $\pi$. This monotone property of Reservation-Priority is crucial to the following performance guarantee.

**Theorem 7.** Reservation-Priority is $\frac{1}{2} - \frac{1}{p}$-competitive for $\text{OBMRR}(p)$.

See Appendix D.1 for a proof of Theorem 7. Note that $\frac{1}{2} - \frac{1}{p}$ is an increasing function of $p$, meaning that the performance guarantee improves as the fraction of requests that reserve increases. The main idea behind this result is to construct a sequence of increasing sets $R_0 \subset R_1 \subset R_2 \subset \ldots \subset R$, where $R_0$ is empty. We can then use tools from matching theory to show that the sequence $\{A_{R_i}\}_{i=0}^{\infty}$ is also increasing, and since $R_i \subset R$ for every $i$, $\lim_{i \to \infty} |A_{R_i}|$ is a lower bound on $A_R$, and hence a lower bound on the size of the matching produced by the algorithm. The final step in the proof is to take expectation over the randomness of $R$.

**Theorem 8.** The competitive ratio of Reservation-Priority is at most $\frac{1}{2} - \frac{1}{p}$ for $\text{OBMRR}(p)$.

See Appendix D.3 for a proof of Theorem 8. Reservation-Priority uses conflicts to avoid making bad matches. Therefore the main idea behind this result is to construct graphs with enough
edges so that finding matches for $R$ is easy, i.e., conflicts will not appear very often, which will make it difficult for the algorithm to know how to avoid making bad matches. Of course, the constructed graph cannot have too many edges, otherwise finding large cardinality matches will be easy even in the absence of reservations [18]. Balancing these two objectives leads to graphs on which Reservation-Priority returns a matching whose size is no more than $\frac{1}{2p}$ times as large as the optimal matching in expectation.


**Corollary 5.** The competitive ratio of Reservation-Priority is exactly $\frac{1}{2p}$ for OBMRR($p$).

**Observation 3.** Reservation-Priority is $p$-agnostic; it does not need to know the value of $p$.

**Remark 9 (Optimal End Point Behavior).** As we discussed in the introduction, OBMRR($p$) generalizes the classical online bipartite matching problem [21] and the offline bipartite matching problem, which are represented by $p = 0$ and $p = 1$ respectively. Note that when $p = 0$, $\frac{1}{2p} = \frac{1}{2}$ is the optimal performance achievable by deterministic algorithms in the online setting, and when $p = 1$, $\frac{1}{2p} = 1$ which is the optimal performance for the offline bipartite matching problem. Hence the competitive ratio of Reservation-Priority monotonically interpolates between optimum online and optimum offline performance among deterministic algorithms as $p$ increases from 0 to 1.

### 5.4.2 Reservation-Priority with Randomization

As we mentioned in the introduction, randomized algorithms can perform better than deterministic algorithms for online bipartite matching. So far we’ve presented a deterministic algorithm called Reservation-Priority which is able to improve its performance based on the fraction of online vertices that choose to reserve. In particular, the competitive ratio of Reservation-Priority monotonically increases from the optimum online (for deterministic algorithms) to the optimum offline as $p$ increases from 0 to 1. A natural question is whether randomized algorithms can also benefit from reservation information. Regarding this question we have the following observation:

**Corollary 6.** Let $\pi$ be a uniformly random bijection between $V$ and $[n]$ where $n = |V|$ that is chosen independently from $R$. Reservation-Priority with a uniformly random $\pi$ is $\max\left(\frac{2}{1-p}, 1 - \frac{1}{e}\right)$-competitive.

**Observation 4.** The RANKING algorithm is optimal among reservation-agnostic algorithms, and has a competitive ratio of $1 - \frac{1}{e}$. From Corollary [6], Reservation-Priority has a better performance guarantee whenever $\frac{2}{1-p} > 1 - \frac{1}{e}$, or equivalently, when $p > \frac{e-2}{e-1} \approx 0.418$.

Thus the performance of Reservation-Priority using random $\pi$ is not strictly increasing. For $0 \leq p \leq \frac{e-2}{e-1}$ its performance guarantee is the same as that of RANKING. A strict improvement is only guaranteed when a sufficiently large fraction of online vertices reserve, i.e., $p > \frac{e-2}{e-1}$. 

**Proof of Corollary 5.** First note that Reservation-Priority is the same as RANKING when \( \pi \) is chosen uniformly at random and \( R = \emptyset \). Therefore by the result of [51], for any instance \( G \) we have \( \mathbb{E}_\pi[|A_\emptyset|] \geq (1 - \frac{1}{e}) \text{OPT}_G \).

Next, recall from Theorem 7 that \( \mathbb{E}_R[|A_R|] \geq \frac{2}{1-p} \text{OPT}_G \) for any \( \pi \).

\[
\mathbb{E}_{\pi,R}[|A_R|] \overset{(a)}{\geq} \mathbb{E}_{\pi,R}[\max(|A_R|,|A_\emptyset|)] \\
\overset{(b)}{=} \mathbb{E}_\pi[\max(\mathbb{E}_R[|A_R|],|A_\emptyset|)] \\
\overset{(c)}{=} \mathbb{E}_\pi \left[ \max \left( \frac{2}{1-p} \text{OPT}_G, |A_\emptyset| \right) \right] \\
\overset{(d)}{=} \max \left( \frac{2}{1-p} \text{OPT}_G, \mathbb{E}_\pi[|A_\emptyset|] \right) \\
\overset{(e)}{=} \max \left( \frac{2}{1-p} \text{OPT}_G, \left( 1 - \frac{1}{e} \right) \text{OPT}_G \right) \\
= \max \left( \frac{2}{1-p}, 1 - \frac{1}{e} \right) \text{OPT}_G
\]

where (a) is due to Lemma 12, while (b) and (c) are due to Jensen’s inequality.

5.5 Discussion

Recall from the introduction that the main goals of this section are to determine whether the different modeling choices made in our work lead to better performance guarantees when compared to other work [51, 54]. We briefly review the modeling differences here:

- Karp Vazirani and Vazirani [51] study an online bipartite matching problem without any reservations. They developed the RANKING algorithm for this problem which has the optimal competitive ratio for the no-reservation setting.

- Kumar et al [54] study a semi-random online bipartite matching problem. In this model, an adversarial \( p \) fraction of online vertices reserve and thus reveal their neighbors to the algorithm before any online vertices arrive. They present an algorithm which has a \( p + (1 - p)^2 \left( 1 - \frac{1}{e} \right) \) competitive ratio in their model.

We are particularly interested in whether the random reservation model studied in our work can lead to improved performance guarantees, since the random model is a relatively reasonable and common assumption to make [3, 7]. Regarding this question, Figure 5.1 plots the competitive ratios achieved by [51, 54] and our work in their respective models. A few remarks are in order.
1. Neither the algorithm from [54] nor Reservation-Priority can improve upon the performance guarantee of RANKING when \( p \leq \frac{e^2 - e - 2}{e} \). Interestingly, our work and [54] have the same threshold on \( p \). Once \( p > \frac{e^2 - e - 2}{e} \), both our work and [54] can improve the performance guarantee of RANKING. This begs the question on whether improvements for the regime \( p \leq \frac{e^2 - e - 2}{e} \) are possible, or why this barrier exists.

2. When \( p > \frac{e^2 - e - 2}{e} \), the competitive ratio of Reservation-Priority in our model is greater than that of [54]'s algorithm in their model. Concretely, in this regime \( \frac{1}{2p} > p + (1 - p)^2 (1 - \frac{1}{e}) \). This suggests that it may be easier to design algorithms with provable performance guarantees in random reservation models.

**Remark 10.** Despite Figure 5.1 showing that Reservation-Priority and the algorithm from [54] perform worse than RANKING in the regime \( 0 \leq p \leq \frac{e^2 - e - 2}{e} \), it is relatively simple to augment them to match the performance of RANKING in this regime. In the case of Reservation-Priority, choosing \( \pi \) randomly is sufficient, as described in Corollary 6. For the algorithm from [54], they can simply run RANKING if \( p \leq \frac{e^2 - e - 2}{e} \) and run their algorithm otherwise. What is important is that they cannot offer strict improvement for small \( p \).

There are several interesting directions for future work. First and foremost is to determine whether it is possible to achieve a competitive ratio larger than \( 1 - \frac{1}{e} \) in the OBMRR\((p)\) model when \( p \leq \frac{e^2 - e - 2}{e} \). A similar question can be asked for the adversarial model studied in [54]. Second is related to hardness results, i.e., what are the upper bounds on possible performance in the OBMRR\((p)\) model as a function of \( p \)? Some results can be obtained by studying the worst case instance of [54] which happens to be a bipartite graph with a complete upper triangular adjacency matrix. Initial analysis seems to suggest that an upper bound increasing monotonically from \( 1 - \frac{1}{e} \) to 1 can be obtained, although the precise details have not yet been carefully and thoroughly constructed.

This concludes the discussion on matching techniques for ridesharing and ridehailing systems. In the next part, we will discuss the potential privacy risks involved with sharing mobility data, and how techniques from cryptography and differential privacy can mitigate those risks while still managing to extracting meaningful insights from the data.
Figure 5.1: A plot of achievable competitive ratios in various different reservation models as a function of the fraction of reservations $p$. The black curve shows the best possible reservation-agnostic algorithm, RANKING [51]. Red shows the performance of [54] under an reservation model where a $p$ fraction of online vertices are adversarially chosen to be reserve vertices. Blue shows the performance of Reservation-Priority in the OBMRR model.
Part IV

Privacy-Preserving Analysis and Computation on Mobility Data
Chapter 6

Private Location Sharing for Routing services

In this chapter we describe how existing forms of location sharing can lead to privacy risks for users of routing services. The privacy concerns stem from the centralized nature of current location sharing schemes wherein users give their individual data to a data custodian, who then makes decisions based on the location data of many people. With regards to privacy, such schemes have a single point of failure in the sense that users must trust that the data custodian will not share or distribute their data for others to see. In practice, however, there are many incentives for the data custodians to share and distribute user data \[52\]. In this chapter, we propose a decentralized protocol for users to estimate network congestion in a privacy-preserving way. The protocol enables estimation of network traffic that is needed to effectively operate a routing service while not exposing the individual level location data of any single user. Furthermore, the decentralized nature of the protocol means there is no single point of failure.

6.1 Introduction

Big Data and data-driven methodologies have shown promise in improving the efficiency, safety and adaptability of mobility services. However, certain types of data sharing can also lead to privacy risks for users. In this chapter we focus on merits and risks of sharing location data. We discuss how location data is useful for determining congestion levels in routing services (e.g., Google Maps, Apple Maps, Waze), and we discuss user privacy risks involved with location sharing. With this as motivation we show how a protocol for decentralized location sharing can mitigate privacy risks while retaining some of the merits of location information for routing services.

Repeated exposure to conventional location sharing can lead to privacy risks for users. In many
CHAPTER 6. PRIVATE LOCATION SHARING FOR ROUTING SERVICES

Figure 6.1: In most routing services, users give their location data in exchange for route recommendations. Routing services often sell this data in data marketplaces. Third parties who buy location data from these routing services will be able to infer preferences, habits, and schedules of users who frequently interact with routing services.

current routing services, users provide their location data in exchange for routing recommendations. While users often only provide a small amount of their location data each time that they use a routing service, if a user regularly uses routing services, the data they share over many interactions can be stitched together to form a more complete picture of the user’s routines, behaviors, preferences, etc. User privacy in such settings thus requires trust that the routing services will not share user data with other entities. However in practice, advertising companies offer to buy this user data to build user profiles for the sake of targeted advertising. As a result, even though users only share small amounts of their location data in each interaction with a routing service, a single entity may end up with a large amount (likely more than the user is comfortable with) of their location data (see Fig. 6.1).

While location sharing presents privacy challenges, it also provides utility for routing services. Location information is helpful because congestion levels of a road can be estimated from the number of vehicles on the road. A key insight toward addressing privacy challenges is that the congestion level only depends on aggregate location information; what matters is the number of vehicles on a road, not which particular users are on the road. This suggests that aggregation procedures can be used to protect individual user location while still providing the location information needed for routing services.

6.1.1 Statement of Contributions

Motivated by this observation, in this chapter we propose a decentralized location sharing protocol where users on the road will periodically compute and announce the traffic counts (e.g., approximate number of vehicles traveling on each road) of the transportation network in a decentralized and privacy-preserving manner. Since only the total number of vehicles on each road is announced, the
Figure 6.2: A visualization of the routing service protocol described in Algorithm 8. Users traveling in the transportation network share their location data in a privacy-preserving way to estimate the traffic counts in a decentralized manner (upper left). These counts are then used to estimate travel times (right). When a user requests a route from the routing service, a shortest path is computed using the estimated travel times (lower left).

location of individual users is not discernible by observers, which is contrary to many current location sharing setups where users give their individual location data directly to routing services. With this protocol, user privacy does not rely on a trusted data custodian, and there are no single points of failure. The protocol is computationally efficient and does not require specialized hardware; all it needs is GPS, which is included in most mobile devices.

Furthermore, assuming the roads in the network are sufficiently large, we can prove that the travel time estimates produced by the protocol will be close to the estimates produced by the ground truth with high probability. This result showcases an interesting complementarity between differential privacy and delay functions used in travel time estimation. In low traffic situations, differential privacy constraints lead to poor accuracy for traffic count estimation. However, delay functions are insensitive for small inputs and can thus tolerate the poor accuracy. On the other hand, delay functions are very sensitive in high traffic situations, and differential privacy can provide high accuracy in these settings. Thus when a delay function is composed with a differentially private mechanism, the two compensate for the others’ weaknesses to yield accurate and private travel time estimates. We corroborate this insight using numerical experiments which show that the protocol provides a privacy-preserving routing service with minimal overhead to the travel time of users.

6.1.2 Organization

This chapter is organized as follows. In Section 6.3 we present a model for the transportation system and specify the system objective. In Section 6.4 we specify and motivate privacy requirements desired
for our application. We review the necessary statistical and cryptographic techniques relevant to our approach in 6.5. We present a decentralized and privacy-preserving routing service protocol in Section 6.6. In Section 6.7, we prove that if the roads in the transportation network are sufficiently large, then the protocol provides a privacy-preserving routing service whose travel time estimates are provably close to the ground truth. We evaluate our protocol in numerical experiments and present the results in Section 6.8. We summarize our work and identify important areas for future work in Section 6.9.

6.2 Related Work

Privacy research in transportation mainly focuses on location privacy, whereby the aim is to prevent untrusted entities from learning geographic locations or location sequences of an individual [10]. A number of privacy-preserving approaches have been proposed for various location-based applications, e.g., trajectory publishing, mobile crowdsensing, traffic control, etc. From a methodological perspective, these approaches are often implemented through spatial cloaking [16], differential privacy [24], and Secure Multi-Party Computation (SMPC) [35].

Spatial cloaking-based approaches rely on aggregation to convert users’ exact locations to coarse information. These approaches are often based on k-anonymity [31], where a mobility dataset is divided into equivalence classes based on data attributes (e.g., geological regions, time, etc.) so that each class contains at least k records [33, 40]. These k-anonymity-based approaches can guarantee that every record in the dataset is indistinguishable from at least k-1 other records. However, k-anonymity is generally considered to be a weak privacy guarantee. Furthermore, due to coarse data aggregation, spatial cloaking-based approaches can lead to low data accuracy.

Differential privacy-based approaches provide a sound privacy guarantee by producing randomized responses to queries, whereby two datasets that differ in only one entry produce statistically indistinguishable responses [25]. In other words, differential privacy ensures that an adversary with arbitrary background information (e.g., query responses, other entries) cannot infer individual entries with high confidence. Existing research for location data either probabilistically generates obfuscated locations from a user’s true location [88, 91] or adds noises to the number of users within each equivalent class [36, 37, 41, 23, 59]. However, differential privacy-based approaches can suffer from two drawbacks. First, due to randomization, there is a trade-off between the accuracy of the response and the level of privacy. Second, most existing research requires a trusted data collector to generate random responses, which does not fit our decentralized setting in this work.

SMPC serves as an excellent technique for decentralized settings, whereby several players jointly compute a function over their data while keeping these data private. Existing SMPC-based research proposes traffic monitoring and control approaches that keep users’ location data confidential, based on secret sharing [90, 72], homomorphic encryption [58, 96], and blockchain [101]. SMPC can ensure
accuracy since no noises are added to protect location privacy. However, SMPC can suffer from high computational overhead due to encryption, and the computation results might leak private information (See Remark 14 for more details).

6.3 Model

In this section we describe the transportation network model, the objective for the users’ distributed algorithm to estimate traffic counts, and the privacy requirements for the algorithm.

6.3.1 Transportation Network

The transportation network is represented as a directed graph $G := (V, E)$ where edges $E$ represent roads and vertices $V$ represent road intersections. We use $n := |V|$ and $m := |E|$ to denote the number of vertices and edges in the graph respectively. The concepts of traffic flow, traffic counts, and travel times are essential to this work, so we will describe them here.

**Definition 14** (Traffic Flow). For a given road $e \in E$, its traffic flow $x_e$ measures the number of vehicles that enter the road during a fixed time interval (e.g., every second).

**Definition 15** (Travel Time). Each edge $e \in E$ has an associated delay function $f_e : \mathbb{R} \rightarrow \mathbb{R}$ where $f_e(x_e)$ is the estimated travel time on the road $e$ if the traffic flow on the edge is $x_e$.

**Definition 16** (Traffic Counts). For a given road $e \in E$, its traffic count $s_e$ is the number of vehicles currently on the road. At steady state the traffic count is equal to the traffic flow multiplied by the travel time. Specifically, $s_e = x_e f_e(x_e)$. For convenience, we define the flow-counts function $F_e$ as $F_e(x_e) := x_e f_e(x_e)$ so that $s_e = F_e(x_e)$.

Throughout this chapter we make the following natural assumption on delay functions.

**Assumption 2** (Properties of Delay Functions). We assume that for each road $e \in E$, $f_e$ is a positive, non-decreasing and differentiable function on $\mathbb{R}_+$.

**Remark 11.** The Bureau of Public Roads (BPR) function $f_{BPR,e}(x_e) := 1 + 0.15 \left( \frac{x_e}{c_e} \right)^4$ is a commonly used volume delay function which satisfies Assumption 2. Namely, it is a degree 4 polynomial with positive coefficients (i.e., $c_e > 0$).

**Definition 17** (Travel Time as a Function of Traffic Counts). For each road $e \in E$ we define $\tau_e$ as the function that estimates travel time based on traffic counts. In other words, for an edge $e$ with volume $x_e$ and counts $s_e$, we have $\tau_e(s_e) = f_e(x_e)$. Since we know from Definition 16 that $x_e = F_e^{-1}(s_e)$, we have $\tau_e(s_e) := f_e(F_e^{-1}(s_e))$.

Road capacities are a concept that will be important to our methodology and results, which we define as follows:
Definition 18 ($\delta$-capacity). For $\delta > 0$, the $\delta$-capacity of a road $e$, denoted $c_{e,\delta}$, is the largest value so that for all $x_e \leq c_{e,\delta}$ we have $f_e(x_e) \leq (1 + \delta)f_e(0)$.

6.3.2 Users and Traffic Counts

At any given time $t$, let $N(t)$ denote the number of users currently traveling in the transportation network. For $1 \leq i \leq N(t)$, the state of user $i$ at time $t$, given by $s(t, i) \in \{0, 1\}^m$, specifies which road the user is on. The $e$th entry of the vector $s(t, i)$ is given by:

$$s_e(t, i) = 1 \left[ \text{user } i \text{ is on road } e \right].$$

Note that exactly one entry of $s(t, i)$ is 1 and all others are 0. The traffic counts at time $t$, denoted $s(t) \in \mathbb{N}^m$, represents the total number of vehicles on each road and is defined as

$$s(t) := \sum_{i=1}^{N(t)} s(t, i).$$

The number of users traveling on road $e$ at time $t$, denoted by $s_e(t)$, is defined as $s_e(t) = \sum_{i=1}^{N(t)} s_e(t, i)$.

6.3.3 Communication Model

In this work we assume that users can communicate with one another through private channels. Concretely, this means that for any pair of users $i$ and $j$, user $i$ can send a message that can only be deciphered by user $j$. Such a communication channel can be easily established using standard public key cryptography systems.

6.3.4 System Objective

The goal of the system is to periodically broadcast estimated travel times for all roads in the network for the sake of route recommendation. The accuracy of travel time estimates will be measured by mean absolute percentage error (MAPE) as defined in Definition 19.

Definition 19 (Mean Absolute Percentage Error (MAPE)). Suppose $T$ is a (possibly randomized) estimator for a positive target value $t^*$. Then the mean absolute percentage error (MAPE) of $T$ is given by

$$\mathbb{E}_T \left[ \left| \frac{T - t^*}{t^*} \right| \right]$$

where the expectation is taken over the randomness in $T$.

The following remark explains how the traffic counts are valuable to this effort.
Remark 12 (routing service from traffic counts). The functions \( \{\tau_e\}_{e \in E} \) from Definition 17 can be used to compute estimated travel times \( \{\tau_e(s_e(t))\}_{e \in E} \) for all roads at time \( t \) from the traffic counts. A routing service can then recommend routes to users based on shortest paths computed from the estimated travel times.

With Remark 12 in mind, the system’s goal is to compute and announce the traffic counts of the system every \( \Delta t \) minutes. Concretely, for each \( k \in \mathbb{N} \), at time \( k\Delta t \) the \( N(k\Delta t) \) traveling users must compute an approximation to \( s(k\Delta t) \) in a distributed and privacy-preserving way where privacy is defined according to Definition 20.

Definition 20 (Privacy-Preserving Mechanism). A mechanism is \( \epsilon \)-privacy preserving if it is \( \epsilon \)-differentially private and can be computed in a distributed setting in a way that is cryptographically secure against semi-honest adversaries.

The precise definitions for cryptographic security, semi-honest adversaries and differential privacy are presented and motivated in Section 6.4.

Remark 13 (On the choice of location data for travel time estimation). In this work we use location data to estimate travel times in a transportation network. This is done by first estimating the traffic flow from traffic counts, and then estimating travel time from traffic flow. One natural alternative is to have users share both their location and speed. In this alternative approach, the location would specify which road the user is on and the average speed reported on a road could be used to estimate its travel time. We opted not to use speed information for two main reasons. The first is due to privacy requirements. As we will discuss and motivate in Section 6.4.2, differential privacy is an important property that we want our method to have. Due to the properties of differential privacy, there are effective ways to compute counts (such as the number of users on a given road) but no clear way to compute an average of user data (such as average speed) in a differentially private way. See Remark 15 for more details. The second is for ease of deployment. Requiring only location data means that our protocol only needs sparse GPS measurements, whereas speed estimation needs continuous GPS measurements, which essentially means that users are being tracked.

6.4 Privacy Requirements

To ensure user privacy, there are two requirements we impose on a desired protocol for the computation of traffic counts: cryptographic security and differential privacy.

6.4.1 Cryptographic Security

Cryptographic security pertains to settings where a group of agents, each with private data, would like to compute a joint function of everyone’s data without any agent needing to reveal its private
data to other agents. In our setting, at time \(k\Delta t\) the \(N(k\Delta t)\) users traveling in the network are agents, where \(s(t, i)\) is the private data of the \(i\)th user, and the desired function is the sum of everyone’s private data. We make the following standard assumption on user behavior:

**Assumption 3** (Semi-honest users). *We assume that all users are semi-honest\(^1\) which means they will follow the protocol but may try to do additional computation to learn the secret data of other users.*

The definition of cryptographic privacy measures privacy by comparing protocols to an ideal computation model which is defined below.

**Definition 21** (Ideal Computation Model). *In the Ideal Computation Model, there are \(n\) agents \(a_1, \ldots, a_n\) with private data \(x_1, \ldots, x_n\) wanting to compute \(f(x_1, \ldots, x_n)\). Each agent sends its private data to a trusted third party which uses the private data to compute \(f(x_1, \ldots, x_n)\) and sends this value back to all of the agents.*

However, since trusted third parties cannot be assumed to exist, the ideal computation model cannot be implemented in a trustless and decentralized setting. Still, this model serves as a gold standard, and cryptographically secure protocols are required to provide the same level of security as this ideal model.

**Definition 22** (Cryptographic Security). *A protocol between \(n\) agents \(a_1, \ldots, a_n\) with private data \(x_1, \ldots, x_n\) wanting to compute \(f(x_1, \ldots, x_n)\) is cryptographically secure if no probabilistic polynomial time agent learns anything more about other agents’ data than they would have learned in the Ideal Computation Model.*

In other words, a protocol is cryptographically secure if no computationally efficient agent learns more from interacting with the protocol than they would from interacting with the Ideal Computation Model.

**Remark 14.** *We emphasize that this does not mean that agents learn nothing about other agents’ data. This is illustrated by a simple three agent example \(a_1, a_2, a_3\) with private data \(x_1, x_2, x_3\) and query function \(f(x_1, x_2, x_3) = x_1 + x_2 + x_3\). In this example, by learning \(f(x_1, x_2, x_3)\), \(a_1\) learns the sum of the other agents’ data: \(x_2 + x_3 = f(x_1, x_2, x_3) - x_1\).*

In light of Remark\(^1\) it is more accurate to say cryptographically secure protocols reveal nothing about other agents’ data beyond the value of the output.

Cryptographic security is necessary for user privacy, since we certainly do not want users to be able to determine the location of certain individuals through our protocol.

Unfortunately, for the application of user location data, cryptographic security alone is not enough to ensure user privacy, which we illustrate in the following example.

---

\(^1\)Semi-honest adversaries, honest-but-curious adversaries, and passive adversaries are equivalent and used interchangeably in the cryptography literature.
Example 2 (Insufficiency of Cryptographic Privacy for Sparse Data). If Alice is an early bird and wakes up to run errands in the city before anyone else wakes up, then in the morning $N(t) = 1$ since Alice is the only person in the network, and therefore we have $s(t) = s(t, Alice)$, hence the traffic counts reveal Alice’s location information. While $s(t)$ does not explicitly label the single traveler in the system as Alice, this information can be inferred if the traveler begins and ends its route at Alice’s house. More generally, cryptographic security does not provide user privacy in sparse data settings. Even when there are multiple users active in the network, side information attacks can be used to associate trajectories in sparse datasets to certain individuals [65].

This motivates the second privacy requirement we enforce in this work, which is differential privacy.

6.4.2 Differential Privacy

With Example 2 in mind, to protect the privacy of users like Alice, the output of a privacy-preserving protocol should not depend too much on the data of any single user. One way to ensure this is through differential privacy. To quantify the influence of a single user, we first introduce the concept of adjacent datasets.

**Definition 23 (Adjacent Datasets).** Two datasets $D_1, D_2$ are adjacent if $D_1$ contains at most one datapoint that is not in $D_2$ and $D_2$ contains at most one datapoint that is not in $D_1$. Concretely, $D_1, D_2$ are adjacent if $|D_1 \setminus D_2| \leq 1$ and $|D_2 \setminus D_1| \leq 1$.

In our setting, a dataset $D_1 = \{s(t, i)\}_{i=1}^{N(t)}$ would be the locations of the $N(t)$ users who are traveling within the transit network at time $t$. The dataset $D_2$ obtained from $D_1$ by modifying the location of one user, who we will call Alice, would be adjacent to $D_1$ since $D_1 \setminus D_2$ contains only the datapoint corresponding to Alice’s original location, and $D_2 \setminus D_1$ contains only Alice’s newly modified location.

One sufficient way to ensure that a mechanism does not depend too much on any single users’ data is to demand that the mechanism behaves similarly on adjacent datasets. This is the approach taken by differential privacy which is defined below.

**Definition 24 (Differentially Private Mechanism).** For $\epsilon > 0$, a $\epsilon$-differentially private mechanism $M : D \rightarrow \mathcal{X}$ is a randomized function mapping datasets into an output space $\mathcal{X}$ so that for any event $E \subset \mathcal{X}$ and any adjacent datasets $D_1, D_2$, we have

$$P[M(D_1) \in E] \leq e^\epsilon P[M(D_2) \in E]$$

To understand why differential privacy gives us the desired privacy we seek, first note that for any two adjacent datasets $D_1, D_2$, the distributions of $M(D_1), M(D_2)$ are very similar. More specifically, the total variation distance between the distributions of $M(D_1), M(D_2)$ is at most $\epsilon$. Because of this, no hypothesis test can determine from the output of the mechanism whether its input was $D_1$
or \( D_2 \) with success probability better than \( \frac{1 + \epsilon}{2} \), which is barely better than random guessing for small \( \epsilon \). This result holds even if the hypothesis test is given knowledge of all datapoints in \( D_1 \cap D_2 \).

Now suppose \( D_1 \) is a dataset that contains Alice’s location, and \( D_2 \) is obtained from \( D_1 \) by modifying Alice’s location arbitrarily. If an observer were able to accurately infer Alice’s location based on the output of a \( \epsilon \)-differentially private mechanism, then it would be able to reliably distinguish between the inputs \( D_1 \) and \( D_2 \). However, since differential privacy makes such a task statistically impossible, by contraposition it is statistically impossible for an observer to accurately infer Alice’s location based on the mechanism’s output. Hence differential privacy ensures privacy of Alice’s data.

The following remark describes a general methodology for achieving differential privacy.

**Remark 15** (Query sensitivity and the required noise level). Dwork’s pioneering work [25] proposes adding noise to queries in order to achieve differential privacy. Given a data set \( D \) and a query \( f \), the mechanism \( D \rightarrow f(D) + Z \) is differentially private so long as \( Z \) is a random variable with sufficiently large variance. Specifically, to achieve \( \epsilon \)-differential privacy, the variance of \( Z \) should be at least \( \frac{L_f^2}{2\epsilon^2} \) where \( L_f \) is the sensitivity of the function \( f \), which is defined as

\[
L_f := \sup_{\text{Adjacent datasets } D_1, D_2} f(D_1) - f(D_2).
\]

Revisiting Remark 13, the role of sensitivity in differential privacy is a main reason why we chose to estimate travel time using counts rather than with average speed. The sensitivity of counting functions is 1 since the modification of a single data point can change the count by at most 1, however the sensitivity of an average is unbounded since a large change to a single data point in a data set can lead to a large change in the average. As such, counting functions are much more compatible with the concept of differential privacy than averages are.

### 6.5 Statistical and Cryptographic Tools

In this section we will describe statistical and cryptographic tools that will be used in our protocol. We will be using a Laplace Mechanism, which is described in Section 6.5.1 to ensure that the protocol is differentially private and will use secure multi-party computation, which is described in Sections 6.5.2, 6.5.3 and 6.5.4 to achieve cryptographic security. Using these tools, we present our privacy-preserving travel time estimation protocol in Section 6.6.

#### 6.5.1 Differential Privacy via the Laplace Mechanism

As previously mentioned, our goal is to compute a differentially private approximation to \( s(k \Delta t) \) for every \( k \in \mathbb{N} \). For this we will use the Laplace Mechanism [25], which produces a differentially
private estimate $S(k\Delta t)$ to $s(k\Delta t)$ based on the following rule

$$S(k\Delta t) := s(k\Delta t) + Z$$

where $Z \in \mathbb{R}^m$ has independent and identically distributed entries according to the Laplace distribution with mean 0 and scale parameter $\frac{1}{\epsilon}$ defined below.

**Definition 25 (Laplace Distribution).** The Laplace Distribution with mean 0 and scale parameter $\frac{1}{\epsilon}$ is a probability distribution over $\mathbb{R}$ denoted as $L_\epsilon$ with probability density function given by

$$L_\epsilon(z) := \frac{\epsilon}{2} e^{-\epsilon |z|} \text{ for } z \in \mathbb{R}. \quad (6.1)$$

We use the Laplace mechanism because it provides a differentially private approximation with the minimum possible mean absolute error [31].

**Fact 1.** The mechanism $s(t) \mapsto S(t)$ is $2\epsilon$-differentially private.

Since we are interested in a decentralized and trustless computation model, the following remark shows that care must be taken in the computation of $S(k\Delta t)$, particularly pertaining to the computation of $Z$, in order for differential privacy to be achieved.

**Remark 16 (Z must remain hidden).** It is essential to the differential privacy of $S(k\Delta t)$ that no observer learns the value of $Z$. Since $S(k\Delta t)$ is announced as the output of the protocol, if in addition $Z$ is known by an observer then that observer can reconstruct $s(k\Delta t)$ by computing $S(k\Delta t) - Z$. In this case, the computation of $S(k\Delta t)$ is not cryptographically secure because the observer learns more about the user data than $S(k\Delta t)$ since in particular it learns the value of $s(k\Delta t)$.

In light of Remark 16 in the next subsection we discuss a cryptographic technique called secret sharing which we will use for a cryptographically secure computation of $S(k\Delta t)$.

### 6.5.2 Secure Multiparty Computation via Secret Sharing

In this section we review a cryptographic tool known as secret sharing and discuss how different variants of it can be used to enable cryptographically secure arithmetic operations on private data. We describe how cryptographically secure addition can be performed on private data in Section 6.5.3 using Additive Secret Sharing, and how cryptographically secure multiplication can be performed on private data in Section 6.5.4 using Shamir Secret Sharing.

### 6.5.3 Secure Multi-Party Addition via Additive Secret Sharing

Suppose there are $N$ agents $a_1, ..., a_N$ and someone wants to share a secret value $x \in \mathbb{N}$ with the agents so that the $N$ agents can reconstruct $x$ if they work together, but no group of fewer than $N$ agents can reconstruct the secret. This can be done using Additive Secret Sharing.
In Additive Secret Sharing, a large prime integer \( p \) is first chosen. The shares \( s_1, s_2, \ldots, s_{N-1} \) are all chosen independently and uniformly at random from the set \( \{0, 1, 2, \ldots, p - 1\} \) and the final share is determined by \( s_N := x - \sum_{i=1}^{N-1} s_i \mod p \). Finally, \( s_i \) is given to \( a_i \) for each \( 1 \leq i \leq N \).

First, note that the \( N \) agents can reconstruct \( x \) by simply adding all of their shares together since by construction we have \( \sum_{i=1}^{N} s_i = x \).

Next note that any group of strictly fewer than \( N \) agents cannot reconstruct the secret. A straightforward calculation shows that for any strict subset \( S \subset [N] \), the distribution of \( \{s_i\}_{i \in S} \) does not depend on \( x \), and therefore \( \{s_i\}_{i \in S} \) provides no information on the value of \( x \).

**Example 3 (Secure Multi-Party Addition).** One valuable application of Additive Secret Sharing is cryptographically secure computation of the sum of agents’ private data. Given \( N \) agents \( a_1, \ldots, a_N \) with private data \( x_1, \ldots, x_N \), their objective is to compute \( x := \sum_{i=1}^{N} x_i \). For each \( 1 \leq i \leq N \), \( a_i \) shares \( x_i \) via Additive Secret Sharing by producing shares \( s_{1,i}, s_{2,i}, \ldots, s_{N,i} \) where \( s_{j,i} \) is given to \( a_j \). At the end of this process, \( a_i \) has received \( \{s_{i,j}\}_{j=1}^{N} \) and can compute \( s_i := \sum_{j=1}^{N} s_{i,j} \). The important observation here is that \( \{s_i\}_{i=1}^{N} \) are additive secret shares for \( x \). Hence the agents can share the values \( \{s_i\}_{i=1}^{N} \) with one another and compute \( x \) via

\[
\sum_{i=1}^{N} s_i = \sum_{i=1}^{N} \sum_{j=1}^{N} s_{i,j} = \sum_{j=1}^{N} \sum_{i=1}^{N} s_{i,j} = \sum_{j=1}^{N} x_j = x.
\]

### 6.5.4 Secure Multi-Party Multiplication via Shamir Secret Sharing

Shamir Secret Sharing [79] offers a more general \( k \)-of-\( N \) method for secret sharing. In a setting with \( N \) agents \( a_1, \ldots, a_N \) and a secret \( x \) to be shared among them, a \( k \)-of-\( N \) secret sharing scheme assigns shares \( s_1, \ldots, s_N \) to the agents so that any subset of \( k \) agents can recover \( x \), but no subset of fewer than \( k \) agents can recover \( x \). Note that Additive Secret Sharing is a \( N \)-of-\( N \) scheme.

Shamir’s Secret Sharing is based on the fact that a \( k-1 \) degree polynomial is uniquely determined from \( k \) evaluations. As in the Additive Secret Sharing setting, a large prime \( p \) is chosen. To share a secret value \( x \), the sharer generates a random \( k-1 \) degree polynomial

\[
X(z) = x + \sum_{\ell=1}^{k-1} C_{\ell} z^{\ell}
\]

where \( C_1, C_2, \ldots, C_{k-1} \) are independent and uniformly distributed over \( \{0, 1, \ldots, p - 1\} \). The share given to \( a_i \) is \( s_i := X(i) \mod p \). By construction, the shares and coefficients satisfy the following
the values of \( H \) Thus if we define the polynomial
\[
\begin{bmatrix}
s_1 \\
s_2 \\
\vdots \\
s_N
\end{bmatrix} = \begin{bmatrix}
X(1) \\
X(2) \\
\vdots \\
X(N)
\end{bmatrix} = \begin{bmatrix}
1^0 & 1^2 & \cdots & 1^{k-1} \\
2^0 & 2^2 & \cdots & 2^{k-1} \\
\vdots & \vdots & \ddots & \vdots \\
N^0 & N^2 & \cdots & N^{k-1}
\end{bmatrix} \begin{bmatrix}
x \\
C_1 \\
\vdots \\
C_{k-1}
\end{bmatrix} \mod p
\]

where \( V_{N,k} \) is the \( N \times k \) Vandermonde matrix.

Since \( X \) is a degree \( k-1 \) polynomial, any group of \( k \) agents can solve a linear system to obtain the values of \( x, C_1, \ldots, C_{k-1} \) and thus recover the secret.

However, for any subset \( S \subset [N] \) with \( |S| < k \), a careful calculation shows that the distribution of \( \{X(i)\}_{i \in S} \) does not depend on \( x \) and hence \( \{X(i)\}_{i \in S} \) provides no information on the value of \( x \).

**Example 4 (Secure Multi-Party Multiplication).** Shamir Secret Sharing and Additive Secret Sharing can be used to perform cryptographically secure multiplication. Given \( N \) agents \( a_1, \ldots, a_N \) with additive secret shares \( \{s_i\}_{i=1}^N, \{s'_i\}_{i=1}^N \) for the values \( x, y \) respectively so that \( x = \sum_{i=1}^n s_i \) and \( y = \sum_{i=1}^n s'_i \), the goal is for the agents to compute the product \( xy \) in a cryptographically secure way.

The computation of \( xy \) will require one round of communication. In this communication round, for each \( 1 \leq i \leq N \), agent \( a_i \) performs Shamir secret sharing for its values \( s_i \) and \( s'_i \). Specifically, it generates two random polynomials \( X_i, Y_i \) of degree \( \frac{N-1}{2} \) so that \( X_i(0) = s_i \) and \( Y_i(0) = s'_i \) and then sends \( X_i(j), Y_i(j) \) to agent \( j \) for each \( 1 \leq j \leq N \).

After the communication, agent \( a_i \) obtains \( \{X_j(i), Y_j(i)\}_{j=1}^N \). From these it computes \( X(i) := \sum_{j=1}^N X_j(i) \) and \( Y(i) := \sum_{j=1}^N Y_j(i) \). Note that \( \{X(i)\}_{i=1}^N \) are Shamir shares for the polynomial \( X := \sum_{j=1}^N X_j \) and similarly \( \{Y(i)\}_{i=1}^N \) are Shamir shares for the polynomial \( Y := \sum_{j=1}^N Y_j \). Since the polynomials \( \{X_j\}_{j=1}^N, \{Y_j\}_{j=1}^N \) all have degree at most \( \frac{N-1}{2} \), the polynomials \( X, Y \) also have degree at most \( \frac{N-1}{2} \). Thus if we define the polynomial \( H(z) := X(z)Y(z) \), then \( H \) has degree at most \( N-1 \).

Now agent \( a_i \) computes \( X(i)Y(i) \). By definition, \( \{X(i)Y(i)\}_{i=1}^N \) are Shamir shares for the polynomial \( H \). Noting that
\[
H(0) = X(0)Y(0) = \left( \sum_{j=1}^N X_j(0) \right) \left( \sum_{j=1}^N Y_j(0) \right) = \left( \sum_{j=1}^n s_j \right) \left( \sum_{j=1}^n s'_j \right) = xy,
\]
we have a degree \( N-1 \) polynomial \( H \) whose constant term is the desired value \( xy \). Furthermore, the \( N \) agents know the value of \( H \) at \( 1, 2, \ldots, N \) and hence can solve a linear system to obtain the coefficients of \( H \) and thus obtain \( xy \).

The Shamir shares \( \{X(i)Y(i)\}_{i=1}^N \) can be converted into Additive shares \( \{\theta_i\}_{i=1}^N \) by setting \( \theta_i := \lambda_i X(i)Y(i) \) where \( \lambda_i \) is the \((1, i)\) entry of \( V_{N,k} \).
Remark 17 (Honest Majority Regime). We would like to mention that the Secure Multi-Party Multiplication scheme we describe in this section requires an honest majority assumption on user behavior. This means that the scheme requires at least \( N + 1 \) users to be fully honest, meaning that they will follow the protocol and will not collude in any way with any other users. The remaining \( N - 1 \) users are assumed to be semi-honest. This is a stronger condition than Assumption 3 which only requires that all users are semi-honest. There exist Secure Multi-Party Multiplication schemes for the semi-honest setting based on Beaver Triples \([21]\), but the scheme we presented in this section is more computationally efficient, and in practice honest majority may not be an unreasonable assumption.

6.6 Cryptographically Secure and Differentially Private Estimation of Travel Times

In this section we show how the differential privacy and secret sharing tools we have discussed can be used to construct a privacy-preserving and decentralized protocol for travel time estimation, which can then be used by a routing service to recommend routes as per Remark 12. The protocol is described in Algorithm 8.

In order to satisfy the privacy requirements as stated in Definition 20, our protocol must be both differentially private and cryptographically secure. Recall from Section 6.5.1 that we will use the Laplace mechanism to obtain a differentially private estimate \( S(k\Delta t) \) to the traffic counts, which is defined below

\[
S(k\Delta t) = s(k\Delta t) + Z
\]

where \( Z \) is an i.i.d. vector of \( L_e \) distributed random variables. We thus need to compute \( S(k\Delta t) \) in a decentralized and cryptographically secure way.

In this section we demonstrate how to compute one entry of the vector \( S(k\Delta t) \), i.e., \( s_e(k\Delta t) + Z_e \) for an edge \( e \in E \). The computation of the entire vector \( S(k\Delta t) \) is obtained by parallelizing the computation of all entries.

We begin by choosing a large prime integer \( p \). Inverse transform sampling is a method which can transform a uniform random variable into a random variable with a desired distribution. Using this method we can compute \( Z_e = F^{-1}(U_e) \) where \( U_e \) is uniformly distributed over \( \{0, 1, ..., p - 1\} \) and \( F^{-1} \) is a scaled version of the cumulative distribution function of \( L_e \). Concretely, \( F^{-1} \) is given by

\[
F^{-1}(u) = \begin{cases} 
\frac{1}{e} \ln \left( \frac{2u}{p} \right) & \text{for } u \leq \frac{p}{2}, \\
-\frac{1}{e} \ln \left( 2 \left( 1 - \frac{u}{p} \right) \right) & \text{if } u > \frac{p}{2}.
\end{cases}
\]

To make the sampling of \( Z_e \) more computationally efficient, we will approximate \( F^{-1} \) with a degree \( d \)
CHAPTER 6. PRIVATE LOCATION SHARING FOR ROUTING SERVICES

are additive shares for polynomial $P_{e,d}$. A larger degree leads to a more accurate approximation of the Laplace distribution but comes at a computational cost.

Thus approximately computing $S_e(k\Delta t)$ amounts to computing $s_e(k\Delta t) + P_{e,d}(U_e)$ where $U_e$ is required to be uniformly distributed over $\{0, 1, \ldots, p-1\}$. First, additive shares $\{\alpha_1, \ldots, \alpha_{N(k\Delta t)}\}$ for $s_e(k\Delta t)$ can be computed using Secure Multi-Party Addition as is done in Example 3. Next, note that if $Y_1, Y_2, \ldots, Y_{N(k\Delta t)}$ are independent and uniformly random on $\{0, 1, \ldots, p-1\}$, then $\sum_{i=1}^{N(k\Delta t)} Y_i \mod p$ is also uniformly distributed. Therefore user $i$ will draw a random value $U_e,i$ so that the value $U_e := \sum_{i=1}^{N(k\Delta t)} U_e,i$ will be uniformly distributed. Using Secure Multi-Party Addition, the users can obtain additive shares $\{\beta_{1,1}, \ldots, \beta_{1,N(k\Delta t)}\}$ for $U_e$. The users will need to compute $P_{e,d}(U_e)$. Since $P_{e,d}$ is a polynomial of degree $d$, the users will need additive shares for $U_e^2, U_e^3, \ldots, U_e^d$ for the computation of $P_{e,d}(U_e)$. The users can obtain such shares through Secure Multi-Party Multiplication by multiplying $U_e$ with itself using Shamir and Additive Secret Sharing as described in Example 4. Using this method the users obtain additive shares $\{\beta_{2,1}, \ldots, \beta_{2,N(k\Delta t)}\}_{z=0}^{d}$ for $\{U_e^2\}_{z=0}^{d}$ respectively. Letting $c_0, \ldots, c_d$ be the coefficients of $P_{e,d}$ so that $P_{e,d}(u) = \sum_{z=0}^{d} c_z u^z$, the users can now construct additive shares for $s_e(k\Delta t) + P_{e,d}(U_e)$ by taking linear combinations of previously computed shares. Explicitly, the shares

$$\left\{\alpha_i + \sum_{z=0}^{d} c_z \beta_{z,i}\right\}_{i=1}^{N(k\Delta t)}$$

are additive shares for $s_e(k\Delta t) + P_{e,d}(U_e)$ since by construction we have

$$\sum_{i=1}^{N(k\Delta t)} \left(\alpha_i + \sum_{z=0}^{d} c_z \beta_{z,i}\right) = \left(\sum_{i=1}^{N(k\Delta t)} \alpha_i\right) + \sum_{z=0}^{d} c_z \left(\sum_{i=1}^{N(k\Delta t)} \beta_{z,i}\right)$$

$$= s_e(k\Delta t) + \sum_{z=0}^{d} c_z U_e^z$$

$$= s_e(k\Delta t) + P_{e,d}(U_e).$$

Algorithm 8 describes the procedure that each user performs to enable the decentralized and privacy-preserving computation of $S(k\Delta t)$.

### 6.7 Analysis: Accuracy of Travel Times based on $S(k\Delta t)$

In the previous section we presented a decentralized and privacy-preserving protocol for computing $S(k\Delta t)$, which is a differentially private estimate of the traffic counts $s(k\Delta t)$ at time $k\Delta t$ (i.e., the $k$th timestep). Since the traffic counts are useful for travel time estimation through volume delay functions, one natural question is how the travel time estimates obtained from $S(k\Delta t)$ will differ...
CHAPTER 6. PRIVATE LOCATION SHARING FOR ROUTING SERVICES

Algorithm 8: Private and Distributed Traffic Count Estimation

Parameters: Large prime number $p$, approximate inverse CDF $P_{e,d}(u) := \sum_{z=0}^{d} c_z u^z$ for the Laplace distribution;

Inputs: Location information $s(k\Delta t, i)$ for user $i$;

Output: Estimated traffic counts $S(k\Delta t)$;

for $e \in E$ do

Using Additive Secret Sharing, obtain a share $\alpha_i$ of $s(k\Delta t) = \sum_{j=1}^{N(k\Delta t)} s(k\Delta t, j)$ through Secure Multi-Party Addition;

Draw $U_{e,i}$ uniformly at random over $\{0, 1, ..., p - 1\}$;

Using Additive Secret Sharing, obtain a share $\beta_1,i$ of $U_e := \sum_{j=1}^{N(k\Delta t)} U_{e,j}$ through Secure Multi-Party Addition;

for $1 \leq z \leq d$ do

Using Shamir and Additive Secret Sharing, obtain a share $\beta z,i$ of $U_z e$ through Secure Multi-Party multiplication;

Compute $\theta_i := \alpha_i + \sum_{z=0}^{d} c_z \beta z,i$ and send $\theta_i$ to all other users;

$S_e(k\Delta t) \leftarrow \sum_{j=1}^{N(k\Delta t)} \theta_j$;

Return $S(k\Delta t)$;

from those obtained from the ground truth traffic counts $s(k\Delta t)$. In this section we show that if the roads in the transportation network $G$ are sufficiently large, then the travel time estimates obtained from $S(k\Delta t)$ will be close to those obtained had we used the non-privacy-preserving ground truth value $s(k\Delta t)$.

We first discuss the errors in estimating the traffic counts due to the Laplace mechanism in Section 6.7.1. Next, in Section 6.7.2 we show how the properties of volume delay functions mitigate the errors induced by the Laplace mechanism, and how composing these two together can help achieve accurate and private travel time estimates.

6.7.1 Accuracy of the Laplace Mechanism

Recall that $S_e(k\Delta t) = s_e(k\Delta t) + Z_e$ is a differentially private estimate of the traffic count on road $e$ at time $t$. The mean absolute percentage error (MAPE) of $S_e(k\Delta t)$ as an estimate for $s_e(k\Delta t)$ is given by

$$\frac{\mathbb{E} \left[ \| S_e(k\Delta t) - s_e(k\Delta t) \| \right]}{s_e(k\Delta t)} = \frac{\mathbb{E} \left[ \| Z_e \| \right]}{s_e(k\Delta t)} = \frac{1}{\epsilon s_e(k\Delta t)}.$$

From this we can make two conclusions. When there is a lot of traffic on $e$, meaning that $s_e(k\Delta t)$ is much larger than $\frac{1}{\epsilon}$, then $\epsilon s_e(k\Delta t)$ will be large and hence $S_e(k\Delta t)$ will have a small MAPE. However, if $s_e(k\Delta t)$ is small, then $S_e(k\Delta t)$ will have a large MAPE.
This shows us that the Laplace Mechanism has poor accuracy when reporting small values. In fact, this is true for all differentially private mechanisms since the Laplace Mechanism has the minimum mean absolute error among all differentially private mechanisms \[31\]. This observation is consistent with Example 2 in that sparse data and small values pose the most difficulty in privacy-preserving efforts.

Fortunately, this bad news does not end our hopes for achieving both accuracy and privacy in travel time estimation. Even if \(S_e(k\Delta t)\) may not always be a good estimate for \(s_e(k\Delta t)\), recall that our ultimate objective is travel time estimation, so we are interested in how well \(\tau_e(S_e(k\Delta t))\) approximates \(\tau_e(s_e(k\Delta t))\). Recall Definition 17 for a description of \(\tau_e\). Next we will show how properties of delay functions can enable accurate travel time estimates even if traffic count estimation is poor.

### 6.7.2 Protocol Accuracy for Travel Time Estimation

In this section we show that if a road \(e \in E\) is sufficiently large, then the travel time estimates computed from \(S_e(k\Delta t)\) are close to the travel time estimates computed from the ground truth \(s_e(k\Delta t)\). Mathematically, this means that \(\tau_e(S_e(k\Delta t))\) is a good estimate for \(\tau_e(s_e(k\Delta t))\).

The key insight behind our result lies in the complementary qualities of differential privacy and volume delay functions. As we saw in Section 6.7.1, the Laplace mechanism has good accuracy when reporting large values, but poor accuracy for reporting small values. Volume delay functions on the other hand, are very sensitive when the input is large, but very insensitive when the input is small. When composing a volume delay function with a Laplace mechanism, the complementary qualities manifest in two ways. When the traffic \(s_e(k\Delta t)\) is larger than a road’s capacity, the volume delay function is very sensitive. Fortunately, in this case the high accuracy of the Laplace mechanism ensures that the traffic count is estimated accurately, leading to accurate traffic flow estimation, which leads to accurate travel time estimation. On the other hand, when the traffic is below the road’s capacity, the Laplace mechanism has poor accuracy, however the delay function is very insensitive in this regime and is able to tolerate large estimation error, leading to accurate travel time estimation. Thus the Laplace mechanism and volume delay function cover each others’ weaknesses to enable accurate travel time estimation for any level of traffic.

We formalize this insight through Theorem 9, which, given desired privacy and accuracy levels \(\epsilon\) and \(\delta\) respectively, provides conditions under which \(\tau_e(S_e(k\Delta t))\) will be close to \(\tau_e(s_e(k\Delta t))\) with high probability. The condition is determined by the road’s \(\delta\)-critical traffic count, which is defined below.

**Definition 26 (\(\delta\)-critical traffic count).** The \(\delta\)-critical traffic count of a road \(e \in E\) is the number of vehicles on the road in steady state so that the travel time is exactly \(1 + \delta\) times as large as its
free-flow travel time. Mathematically, the \( \delta \)-critical capacity of \( e \) is

\[
F_e(c_{e,\delta}) = c_{e,\delta}f_e(c_{e,\delta}) = (1 + \delta)c_{e,\delta}f_e(0)
\]

where \( c_{e,\delta} \) is the \( \delta \)-capacity of the road \( e \) as defined in Definition 18. With this setup in place, we now present Theorem 9.

**Theorem 9 (Accuracy of Travel Time Estimates).** Let \( \epsilon, \delta \geq 0 \) specify the desired privacy and accuracy levels respectively and let \( p \in [0, 1] \) represent a failure probability. For a road \( e \in E \), if \( f_e \) satisfies Assumption 2 and

\[
(1 + \delta)c_{e,\delta}f_e(0) \geq \frac{1}{\epsilon} \left( \frac{1}{\delta} + 1 \right) \log \frac{1}{p}
\]

where \( c_{e,\delta} \) is the \( \delta \)-capacity of \( e \), then for any value of \( s_e(k\Delta t) \in \mathbb{N} \), the following condition is satisfied with probability at least \( 1 - p \):

\[
\frac{|\tau_e(S_e(k\Delta t)) - \tau_e(s_e(k\Delta t))|}{\tau_e(s_e(k\Delta t))} \leq \delta.
\]

See Appendix E.1 for a proof of Theorem 9. Next, we will discuss whether the condition required by Theorem 9 is satisfied in practice by roads in real transit networks.

### 6.7.3 Discussion

One natural and immediate question is whether the requirement on the \( \delta \)-critical traffic count of roads in Theorems 9 are satisfied by real road networks. To answer this question we first discuss parameter choices. To ensure a meaningful privacy guarantee for each timestep, \( \epsilon \) should be significantly smaller than 1. For this reason we focus on applications where \( \epsilon = 0.2 \).

With \( \epsilon = 0.2, \delta = 0.1 \) and \( p = 0.1 \), the condition in Theorem 9 requires that a road’s \( \delta \)-critical count be at least 127 cars. For such roads, the Theorem states that the estimated travel time will be within 10 percent of the ground truth with probability at least 90 percent. To see whether such a requirement is reasonable, we studied a real world transportation network. The \( \delta \)-critical capacities of all roads in the Sioux Falls transportation network are plotted in Figure 6.3. The figure shows that more than 80 percent of the roads in the Sioux Falls network have \( \delta \)-critical counts above 127. Thus the conditions required by Theorem 9 are realistic for most roads.

Care must also be taken in choosing the value of \( \Delta t \), i.e. the amount of time between updates to the estimated traffic counts. Specifically, \( \Delta t \) should be chosen so that it is similar to the typical travel time of a road in the network. If \( \Delta t \) is much smaller than the typical travel time on a road, then sample average approximation can be used to denoise \( S(k\Delta t) \) to get a better estimate of \( s(k\Delta t) \). While better estimation is usually good, it is also synonymous with less privacy. To illustrate this
CHAPTER 6. PRIVATE LOCATION SHARING FOR ROUTING SERVICES

97

count, suppose that for timesteps $1, 2, 3, ..., N$ the ground truth traffic counts for a given road $e$ is constant, meaning that there is a value $s_e$ so that $s_e(k\Delta t) = s_e$ for all $1 \leq k \leq N$. Now for each $k$, $S_e(k\Delta t)$ is an unbiased estimator for $s_e$ with variance $\frac{2}{\Delta t}$. This variance is necessary to ensure differential privacy. However, note that $\frac{1}{N} \sum_{k'=1}^{N} S_e(k'\Delta t)$ is also an unbiased estimator for $s_e$ but now has variance $\frac{2}{N\Delta t}$. This decrease in variance leads to worse privacy guarantees. For this reason, $\Delta t$ should be chosen similar to the travel time of a road so that $N$ can never get too large.

6.8 Numerical Experiments

We evaluate the performance of our protocol in the Sioux Falls road network setting. The purpose of these numerical experiments is to compare the travel time experienced by vehicles when they are routed using (a) travel time estimates derived from our protocol and (b) travel time estimates derived from the ground truth traffic counts. In particular we want to quantify the extent of travel time degradation incurred by the use of our protocol’s privacy-preserving mechanisms. Thus, these experiments help us test the practical implications of our road-level theoretical results when viewed in the context of the entire network. In Section 6.8.1, we present details about the data sources, road network, traffic demand, and the experimental setup. In Section 6.8.2 we study the impact of our protocol on route choices and travel time. These simulations suggest that the price for privacy may be negligible or even zero, thereby strengthening the case for conducting real-world field studies.

6.8.1 Setup

The properties of the Sioux Falls road network as well as the typical user demands is obtained from the Transportation Network Test Problems (TNTP) dataset. The Sioux Falls road network consists of 24 nodes and 76 edges. Each edge is characterized by a maximum speed, free flow throughput (i.e., vehicles per hour), and length of the segment. Note that we use the terms edge and road interchangeability. Travel times are computed using BPR functions whose parameters are obtained from the aforementioned edge characteristics. Additionally, the dataset also reports the steady state traffic demand between 528 origin-destination (OD) pairs.

We conduct two types of experiments: private routing and non-private routing. In both types of experiments we simulate traffic flow on the network at a time resolution of 10 seconds. At every time step, we draw new demand from a Poisson distribution, with the mean demand proportional to the steady state demand reported in the dataset. Thus, at each time step, we draw a random number of vehicles with a corresponding origin and destination. For each vehicle, we identify a shortest travel time route between its origin and destination nodes. In the private routing experiment, the shortest path is computed using the most recent travel time estimates produced by our protocol. In the non-private routing experiment, the shortest path is computed using the ground truth travel time which depends on the ground truth traffic counts. In both types of experiments, the simulated
CHAPTER 6. PRIVATE LOCATION SHARING FOR ROUTING SERVICES

Table 6.1: Demand scenarios used in our simulation

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Rate (vehicles/hr)</th>
<th>Min road utilization</th>
<th>Max road utilization</th>
<th>Average road utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>60,100</td>
<td>0.05</td>
<td>1.15</td>
<td>0.52</td>
</tr>
<tr>
<td>Low</td>
<td>30,050</td>
<td>0.00</td>
<td>0.90</td>
<td>0.28</td>
</tr>
<tr>
<td>High</td>
<td>90,150</td>
<td>0.07</td>
<td>1.47</td>
<td>0.74</td>
</tr>
</tbody>
</table>

movement of the vehicles on the road network is determined by the ground truth traffic counts on that road.

The duration of the simulation is 2 hours, with additional buffer time in the end for vehicles already in transit to complete their trips. For the private routing experiments, the vehicles are assumed to use our protocol to update travel time estimates every $\Delta t = 2$ minutes to minimize the number of reports that a vehicle makes from the same road (See Section 6.7.3). In our experiments, we consider $\epsilon$ values of 0.01 and 0.1. We will discuss how our protocol would perform for other values of $\epsilon$ in Section 6.8.2. We consider three vehicle demand profiles. In the baseline scenario, about 60,120 vehicles are expected to join the road network every hour. We also consider a low demand scenario, where all the OD Poisson parameters are decreased by 50% and a high demand scenario, where all the OD Poisson parameters are increased by 50%. To gain some more insight into the degree of strain this demand induces on the road network, we refer the reader to Table 6.1. Here, we report the rate of vehicles being added to the network for each scenario as well as the minimum, maximum, and average road utilization during the simulation period. The road utilization is defined as the term $x/c$ from Remark 11. Utilization greater than 1 indicates congestion on a road. The baseline demand profile results in several congested roads – representing a realistic scenario for testing our protocol. Additionally, as expected, increasing the demand rate results in a higher road utilization and more congested roads.

6.8.2 Results

We evaluate the impact of privacy noise on the routes and travel times of vehicles in the network. Table 6.2 presents several performance measures for $\epsilon = 0.01$ under the three demand profiles. Note that the choice of $\epsilon = 0.01$ is overly conservative since smaller $\epsilon$ represents a stricter privacy requirement and thus more noise injected into the system. Typical values of $\epsilon$ that are chosen in practice are larger than 0.1 depending on the application. Nevertheless, we consider this relatively stringent privacy requirement to understand the performance limits our protocol.

From Table 6.2, we first observe that the increase in average travel time for a vehicle is only 8 seconds in the baseline case. This corresponds to a 1.3% increase in travel time. For the low and high demand scenarios, the increase in average travel times are 3.1 sec (0.6%) and 13.2 sec (1.9%) respectively. Vehicles will experience a low additional travel time if the routing decisions for vehicles
CHAPTER 6. PRIVATE LOCATION SHARING FOR ROUTING SERVICES

Figure 6.3: Distribution of $\delta$-critical counts for all roads in the network for $\delta = 0.1$. The dashed red line denotes the threshold above which the road travel time estimate will have an error less than 10%, 90% of the time to give a privacy level of $\epsilon = 0.2$.

Performance measure | Low demand | Baseline demand | High demand
--- | --- | --- | ---
Non-private Travel time (sec) | 546.5 | 591.9 | 678.2
Travel time with our protocol (sec) | 549.6 | 599.9 | 691.4
Increase in travel time (sec) | 3.1 | 8.0 | 13.2
Increase in travel time (%) | 0.6 | 1.3 | 1.9
Cars with no change in route (%) | 90.9 | 88.3 | 87.1
Cars with no increase in travel time (%) | 65.9 | 41.3 | 20.6

Table 6.2: Performance measures for $\epsilon = 0.01$

from our protocol closely resembles what they would have done without any privacy noise. In other words, if the shortest paths on the graph with noisy travel time estimates is the same (or very similar) to the shortest path on a graph with accurate travel time estimates, the vehicles will experience very little additional travel time. Our results confirm that this is indeed the case – the routes chosen by almost 90% of the vehicles are unchanged due to our protocol. Finally, as we compare the three demand scenarios, we observe that higher demand results in greater congestion and subsequently higher travel times. Furthermore, when demand is higher, the choice of an appropriate route for each vehicle is even more critical to minimize. Working with noisy estimates of travel time, during periods of high congestion can thus lead to incorrect routing choices. This is consistent with our observation that the route of 90.9% of the cars are unchanged by our protocol when the demand is low, but only 87% of the routes remain unchanged when the demand is high.

Next, we study the performance of our protocol with a lower privacy setting of $\epsilon = 0.1$. Note that we expect that the performance of our protocol converges to the non-private setting as $\epsilon \to \infty$. Interestingly, our results show that even with $\epsilon = 0.1$, the performance of our protocol becomes indistinguishable to the non-private setting. The performance of our protocol for $\epsilon = 0.1$ is shown in Table 6.3. We observe that the increase in travel time is nearly 0 for all the three demand scenarios.
In fact, the randomness in the travel time estimates can also result in marginal improvements in travel time in some settings (reflected as a negative increase in travel time). Not surprisingly, the routes chosen by nearly all the cars also is unaffected by our privacy preserving protocol. The results from varying $\epsilon$ are very encouraging – for a reasonable privacy requirement, we are able to get privacy for ‘free’ with no loss in system performance. Although not shown for brevity, our experiments indicated that cars observe no increase in average travel time for all values of $\epsilon > 0.1$.

The results from these experiments are significantly better than what is guaranteed by Theorem 9. As we discussed in Section 6.7.3, Theorem 9 promises that the estimated travel time on each road will be within 10 percent of the ground truth at least 90 percent of the time when $\epsilon = 0.01$. Our numerical results in Table 6.2 show that even when $\epsilon = 0.01$ (i.e., 20 times as noisy as $\epsilon = 0.2$), our protocol only introduces an overhead of 8 percent in the baseline case and 13 percent in a high demand case. We believe this is because real world road networks have redundancy. By this we mean that there are many near-optimal paths from an origin to a destination, so it is very likely that at least one of these paths has an accurately estimated travel time. Theorem 9 looks only at the edge level and is thus does not exploit favorable network topologies. On a positive note, Theorem 9 is general in the sense that it can be applied to a network with any topology and any demand structure.

6.9 Discussion

In this chapter we propose a protocol for a decentralized routing service where travel times are computed from user location data in a privacy-preserving way. In most current routing services, users give their individual location data directly to routing services in exchange for route recommendations. Since this data is associated with the users’ identity, users’ schedules, habits, preferences and other private information can be inferred through repeated interactions with routing services. Contrary to this, the protocol proposed in this paper is both differentially private and cryptographically secure, meaning that only the aggregate effect of traffic on travel time is obtainable from the protocol, and users’ individual location data cannot be inferred by other parties. We also show that for large roads, it is possible to estimate travel time both accurately and privately. This is due to complementary
qualities of differential privacy and delay functions. We evaluated the performance of the protocol through simulation in the Sioux Fall transportation network and showed that the protocol incurs minimal performance overhead in practice while providing a principled privacy guarantee.

There are many interesting and important directions for future work. The first direction is related to finding a more refined definition for privacy. Travel time estimation in the literature is often based on flow or average speed of vehicles on the road. However, we chose to estimate travel times based on traffic counts due to compatibility with differential privacy. More specifically, without additional domain-specific assumptions, it is impossible to compute flow or average speed in both an accurate and differentially private way (see Remark 15). Thus while differential privacy is a general and powerful concept, it is perhaps too restrictive for some common mobility applications such as flow or speed estimation. Developing a more specialized notion of privacy for mobility applications could enable more algorithmic possibilities while retaining meaningful privacy guarantees. The second direction is related to adoption rate. In this paper we implicitly assume that all vehicles in the network are willing to participate in the protocol, though technically a uniform and known adoption rate would be sufficient. While we believe this is a reasonable assumption in an era of connected vehicles, developing a protocol that is agnostic to participation rate would provide robustness. A third direction is related to other applications. Developing decentralized and privacy-preserving pricing for roads and for mobility services would be an interesting direction.
Chapter 7

Conclusions

The modern transportation industry is one with bountiful opportunities and potential for growth and improvement. Advancements in information technology enable customers and service providers to conveniently communicate and exchange information in real-time. This ease of real-time communication has led to many on-demand mobility services including ridehailing, ridesharing, food delivery and bikesharing. While these services bring new features to the transportation space, they also bring new opportunities. The ubiquity of transportation and delivery to citizens of all backgrounds means that optimizing the efficiency of these on-demand mobility systems can bring quality of life improvements to many people throughout the world.

In this thesis, we explored several challenges associated with on-demand mobility services. Part II covered rebalancing techniques in ridehailing and ridesharing systems to reposition vacant service vehicles so that they are better align with future customer pickup locations, thereby reducing waiting times. Part III covered matching techniques to maximize the efficiency of ridesharing systems by identifying in real-time groups of customers that can be efficiently served by a single vehicle. Part III also briefly explores cooperation between a ridehailing system and its customers, by sharing information through ride reservations, that can further improve matching efficiency between drivers and riders. Part IV covers potential privacy concerns that may arise in the location sharing that is often used by data-driven mobility services. Part IV then proposes techniques to mitigate such privacy concerns while still obtaining valuable learnings from user data.

The wide variety of techniques described in this thesis is a testimony to the multi-faceted nature of modern mobility systems. We conclude this thesis by summarizing its contributions and highlighting important directions for future work.
CHAPTER 7. CONCLUSIONS

7.1 Summary of Contributions

7.1.1 Contributions from Part II

Part II presented model predictive control algorithms for ridehailing and ridesharing systems to simultaneously serve requests and reposition vacant vehicles in anticipation of future demand.

In Chapter 2, we developed a Stochastic Model Predictive Control algorithm for Autonomous Mobility-on-Demand systems that leverages uncertain travel demand forecasts. This is done by predicting a distribution of future demand rather than predicting point estimates. We discussed two variants of the proposed algorithm, one using integer programming and a relaxed, totally unimodular linear programming approach that trades optimality for scalability. Through experiments, we show that the latter algorithm provides a 62.3 percent reduction in customer waiting time compared to existing non-stochastic approaches.

In Chapter 3 we presented a model predictive control algorithm to coordinate a fleet of self-driving vehicles for servicing travel requests in a ride-sharing setting. To this end, we first derive an integer network flow model to represent the transportation network. We then designed a Ride-sharing Autonomous Mobility on Demand (RAMoD) algorithm based on receding horizon network flow optimization. We presented a case study for San Francisco, CA, and compared our algorithm to the state-of-the-art, using high fidelity simulations in MATSim. Our experiments showed the proposed RAMoD-MPC algorithm outperforms the state-of-the-art unit-capacity mobility algorithms in terms of total driving distance and reactive ride-sharing algorithms in terms of mean wait time. In particular, by slightly increasing the total trip length for customers, the RAMoD-MPC algorithm is able to significantly reduce the distance traveled by mobility providers.

Key take-aways:

- In ridehailing and ridesharing systems, repositioning vacant vehicles to areas where future demand is likely to appear is important to sustaining good service quality over a long period of time.

- Future demand is valuable for identifying where vacant vehicles should be allocated to.

- Algorithms incorporating a distribution over future demand exhibit robust behavior compared to algorithms that incorporate only point estimates of future demand.

- Ridesharing systems can provide service quality similar to that of ridehailing systems while having lower operation costs. This is because a single service vehicle can carry multiple customers, thereby reducing the total distance driven by the ridesharing fleet.
7.1.2 Contributions from Part III

Part III presented techniques from matching theory to design rider-driver and rider-rider assignment algorithms for ridehailing and ridesharing systems respectively.

In Chapter 4 we presented a real-time algorithm for ridesharing systems that selects groups of riders to share a ride. By representing riders as vertices and potential ridesharing groups as hyper-edges, we formulated the ridesharing problem as an online hypergraph matching problem. When the objective is utility maximization, we presented a polynomial-time randomized batching algorithm that has the best possible worst-case performance. When the objective is cost minimization, we present a deterministic thresholding algorithm for the 2-capacity setting, and prove that it has the optimal worst-case performance among deterministic algorithms. We also discussed another variant of the thresholding algorithm that has slightly worse performance guarantees but is a decentralized algorithm that is more computationally efficient. For larger vehicle capacities, we presented a polynomial-time algorithm based on the set cover problem, and show that its competitive ratio is within a factor of 2 of the best possible competitive ratio achievable in polynomial time.

In Chapter 5 we presented a real-time algorithm for ridehailing systems that can leverage rider reservations to improve the efficiency of rider-driver matching. To this end, we formulated the ridehailing system with rider reservations as an online bipartite matching problem with random reservations which interpolates between the classical online bipartite matching problem and the classical offline bipartite matching problem as the fraction of reserved trips increases. We present a reservation priority algorithm whose performance is an increasing function of the number of ride reservations. In particular, reservation priority coincides with the optimum online when there are no reservations and coincides with the optimum offline when all riders reserve their requests. In summary, the information provided from rider reservations enables better performance guarantees than what is possible in classical worst-case analysis.

Key take-aways:

- In addition to being a useful heuristic in ridehailing and ridesharing systems, batching algorithms also have nice theoretical properties, and are even optimal algorithms in certain settings.

- Thresholding algorithms for rider-rider matching in ridehailing systems can be implemented in a decentralized manner, making them a nice candidate for scaling to even larger systems while maintaining computational tractability. They also have nice theoretical properties.

- Cooperation between riders and ridehailing systems can lead to win-win outcomes. If riders inform the ridehailing system of their travel preferences ahead of time, the system can leverage this improve its rider-driver assignment, leading to better overall service for customers.
• The common approach that ridehailing systems take to learn about rider demand is through demand forecasting. The cooperation approach can be effective even in settings where demand forecasting techniques struggle, for example, in settings with highly non-stationary demand or in rare events where training data either does not exist or is very scarce.

7.1.3 Contributions from Part IV

Part IV studies potential privacy concerns that arise when sharing the location data of users, and how to design systems to mitigate such concerns while still extracting information from the data that is needed to effectively operate mobility systems.

Chapter 6 begins by discussing potential privacy concerns that arise from centralized data sharing schemes used by current routing services. In a centralized data sharing scheme, users report their individual-level location data to a data custodian, who then uses the data within some optimization pipeline to make decisions. In the context of routing services, user location data can be used to infer congestion levels within the network and can thus be used to steer new users away from congested areas. Unfortunately, the habits, preferences, and schedules of a user can be inferred from many repeated measurements of their individual-level location data, thereby introducing potential privacy risks to the user. As such, centralized data sharing schemes are privacy-preserving only if the data custodian protects the data, and does not re-distribute it for others to see. Unfortunately, the multi-billion dollar market for location data makes such a condition tricky to satisfy.

We proceeded by observing that, at least for the application of routing services, what matters is not which people are on a given road, but how many people are on a given road. This led us to use cryptographic techniques, particularly secure multi-party computation, to compute the aggregate traffic counts needed by routing services (i.e., the number of cars on every road) without needing users to expose their individual level location information. Even when using secure multi-party computation techniques, a satisfying level of privacy is still out of reach, particularly when usage of the transportation network is sparse (e.g., very early morning). Data aggregation provides privacy only when there are many data points to aggregate, and thus aggregation will not provide privacy when there are very few cars on the road. An extreme example is when there is only 1 car in the network, in which case the traffic count information is equivalent to the individual level location of that car’s owner. To combat this sparsity issue, we use differential privacy to add noise to the traffic counts to protect user location at the individual level. The merits of differential privacy, however, do not come for free. Due to noise injection, there is an unavoidable trade-off between accuracy and privacy. This trade-off does not end up being an issue as our ultimate goal is to estimate travel times on roads, not traffic counts. The final piece to the puzzle is a complementarity between differential privacy and volume delay functions that are often used in traffic models. When traffic on a road is low, differential privacy provides poor accuracy for traffic counts, but traffic models are insensitive to traffic counts in this regime. On the other hand, when traffic is high, the traffic models are very
sensitive to traffic counts, but differential privacy can provide high accuracy for this regime.

Putting everything together, we developed a distributed protocol whereby users of the road network can compute noisy estimates of traffic counts on all roads in the network in a decentralized and privacy-preserving way. The noisy traffic counts can then be used to compute accurate travel time estimates for all roads in the network, and can thus be used for route recommendation. The decentralized nature of the protocol means it does not have a single point of failure, contrary to existing centralized data sharing schemes.

Key take-aways:

- Aggregate location information can help routing services estimate network congestion.

- Sharing individual level user location can cause privacy risks, and is not necessary for routing services since aggregate location is already enough.

- Users in a transportation network can compute a noisy estimate of the aggregate location information in a decentralized and privacy-preserving way, based on cryptographic techniques and differential privacy. The noisy estimates of aggregate location data can be converted into accurate travel time estimates on all roads in the network through the use of volume delay functions.

- We presented a decentralized and privacy-preserving routing service where no participant’s individual level data is ever observed by another party.

7.2 Future Directions

In addition to the next steps described at the end of each chapter, the following future directions are important to the development of modern mobility systems.

1. **Systems with multiple mobility operators**: This thesis primarily studied systems with a single ridehailing or ridesharing service. In such cases, the system’s overall efficiency is synonymous with the service’s efficiency. However, in practice there may be multiple mobility companies offering similar or competing services. As an example, in the United States Uber and Lyft offer ridehailing and ridesharing services, and are competitors vying for market share. Competition between service providers wherein each service provider tries to optimize its own utility rather than the overall system utility can lead inefficient outcomes such as “races to the bottom” or “tragedies of the commons”. In systems with competing mobility providers, there are two questions that arise.

   1.a) **Incentive Alignment**: Taking an economic perspective, one method to avoid inefficient outcomes is to design incentives or policies that will cause self-interested service providers
to act in a cooperative or coordinated manner that is conducive to high overall system efficiency. For example, congestion pricing can lead to efficient outcomes in traffic assignment problems. In traffic assignment problems where the goal is to optimally route a set of users from their specified start to end locations, selfish users choosing their routes in the presence of carefully chosen road tolls leads to optimal system efficiency \[70, 89\]. Existing works in traffic assignment and congestion games often use steady state traffic models and assume full knowledge of transportation demand, both of which are not realistic for modern ridehailing systems \[19\]. Thus this future direction will involve generalizing existing works in traffic assignment and congestion games to real-time settings with unknown or partially known demand.

1.b) \textit{Cooperation and Privacy:} Cooperation between multiple mobility service providers will require them to share information with each other on which resources they need. On the other hand, the service providers are competitors and may not want to reveal trade secrets to other service providers, e.g., number of customers, routing logic, rider-driver assignment logic. With these trade-offs in mind, what is the minimum amount of information that needs to be shared among service providers for a socially optimal outcome to be achievable? Zero knowledge proof techniques from cryptography could be used to verify coordination between multiple service providers without requiring direct inspection of private data \[13\].

2. \textbf{Improved modeling choices in matching systems:} Part \textbf{[III]} presents matching problems that are used to model rider-rider and driver-rider assignment in ridesharing and ridehailing systems. There are several aspects of real-world systems that are not captured by the matching problems presented in this thesis that have great practical importance and thus deserve further investigation.

2.a) \textit{Ridesharing robustness to cancellations:} One common phenomenon in ridesharing systems is rider cancellations. A efficient ridesharing group can become inefficient if one or several of the riders in the group choose to cancel their participation. There are thus many important questions involving how to model rider cancellations, as a rider’s decision to cancel may depend on the group it is assigned to (e.g., the amount of detour when compared to a non-ridesharing trip). When accounting for cancellations, ridesharing groups should be evaluated on their \textit{expected} utility, requiring the algorithm to have contingency routes in the event that a subset of a ridesharing group decides to cancel their participation. Thus the trade-off between a thorough set of contingency plans versus computational speed would need to be explored.

2.b) \textit{Dynamic Pickups in Ridesharing} In Chapter \textbf{[4]} we studied an online matching model where matching decisions were irrevocable. This modeling decision was motivated by
the fact that too many unexpected or long detours will negatively affect user experience. By making matching decisions irrevocable, the platform must honor whatever service it initially promises the customer, and thus eliminating unforeseen customer inconveniences. However, this modeling decision may be overly restrictive on the ridesharing service. To illustrate this, consider the following example. Alice is currently the only passenger in a ridesharing service vehicle. Bob hails a ride, and Bob’s trip is on the way to Alice’s location, i.e., the nominal path from Bob’s pickup to Bob’s dropoff is a subset of Alice’s current route. In this example, Alice’s service vehicle can pick up and serve Bob with minimal impact on Alice’s ride. Identifying low-effort opportunities to dynamically improve ridesharing groups would thus be of great practical use, and should thus be studied.

2.c) **Further exploring distributed ridesharing algorithms:** In high capacity ridesharing services, it is often the case in practice that enumerating the set of candidate ridesharing groups (i.e., the set of hyperedges of the hypergraph) is the most computationally expensive step. In this thesis we presented a Risk-Threshold-ag algorithm for the $k = 2$ cost minimization case, which is a decentralized algorithm which does not need to enumerate the entire edge set in order to construct its matching. The ability to avoid enumerating all candidate ridesharing groups presents attractive computational savings, however, one needs to make sure that the performance does not suffer from the shortcuts taken by such algorithms. To this end, generalizing Risk-Threshold-ag to higher capacity vehicles, empirical tests and algorithm tuning is likely required.

3. **More on Rider and System Cooperation:** In Chapter 5 we studied a rider-drive matching problem and showed how cooperation between riders and the system through rider reservations can lead to higher system efficiency. There are two main questions one can ask regarding that work. First, what other types of cooperation between customers and service providers can be incorporated? Related to this question, more sophisticated behavior models for users can be incorporated to make the model more realistic. The second question ties this work together with Part IV: Can the riders communicate with the service provider in a way that improves system efficiency in a way that preserves their privacy? Techniques from oblivious transfer may enable an anonymous hailing feature whereby a rider can authenticate themselves, hail and pay for a ride without the service knowing which of its customers requested the ride.
Appendix A

Background: Hypergraph Matching and Set Cover

A.1 Hypergraph Matching

Definition 27 ($k$-Max-Matching). The $k$-Max-Matching problem is to find a maximum weight matching in a rank $k$ hypergraph $H$. The maximum weight matching problem in graphs is a special case where $k = 2$.

The best known algorithms for $k$-Max-Matching [11, 15] are combinatorial in nature and are $2^k$-approximations. Convex relaxations are surveyed in [14] who show that the standard linear programming relaxation of $k$-Max-Matching can be rounded to produce a $1 - 1 + \frac{1}{k}$-approximation algorithm. Conversely, [39] prove that it is NP-hard to approximate $k$-Max-Matching better than $\Omega \left( \frac{\log k}{k} \right)$, showing that existing methods are optimal up to log factors.

Among all of the existing methods, we focus our discussion on a Greedy algorithm outlined in Algorithm 9. As described in Lemma 13, Greedy is appealing because its approximation ratio is within a constant factor of the best known result and also runs in nearly linear time.

Lemma 13 (Greedy is a $\frac{1}{k}$-approximation). The approximation ratio of Algorithm 9 is at least $\frac{1}{k}$ for $k$-Max-Matching. Furthermore, its runtime is $O(|E| \log |E|)$.

A.2 Set Cover

The set cover problem and its variants as defined below are crucial to our discussion on minimum cost matchings.
Algorithm 9: Greedy

1. **Input:** Weighted Hypergraph $H = (V, E, w)$
2. **Output:** A matching $M$ in $H$
3. Set $M \leftarrow \emptyset$
4. Sort edges in $E = \{e_j\}_{j=1}^{|E|}$ according to their weights, in decreasing order to obtain $\{\tilde{e}_j\}_{j=1}^{|E|}$.
5. for $1 \leq j \leq |E|$ do
6.     if $M \cup \{\tilde{e}_j\}$ is a matching then
7.         $M \leftarrow M \cup \{\tilde{e}_j\}$
8.     Return $M$

**Definition 28** (Weighted Set Cover ($\text{WSC}$)). Given a set $X$, a collection $S \subseteq 2^X$ of subsets of $X$ satisfying $\cup_{S \in S} S = X$ and a weight function $w : S \rightarrow \mathbb{R}_+$, the weighted set cover problem $\text{WSC}(X, S, w)$ asks the following minimization problem

$$\minimize_{\mathcal{E} \subseteq \mathcal{S}} \sum_{S \in \mathcal{E}} w(S)$$

s.t. $\cup_{S \in E} S = X$.

**Definition 29** (Set Cover ($\text{SC}$)). An instance of $\text{WSC}(X, S, w)$ is an unweighted set cover problem if $w(S) = 1$ for every $S \in S$. We denote such an instance as $\text{SC}(X, S)$.

**Definition 30** (Set Cover with a rank constraint ($k$-$\text{WSC}$)). An instance $\text{WSC}(X, S, w)$ is rank $k$ if $|S| \leq k$ for every $S \in S$. We denote the rank $k$ set cover problem as $k$-$\text{WSC}(X, S, w)$. If in addition, $w(S) = 1$ for every $S \in S$, then this is an instance of rank $k$ unweighted set cover, which we denote as $k$-$\text{SC}(X, S)$.

The set cover problem and its variants have been extensively studied in the computer science literature. A simple greedy algorithm was shown in [50, 60] to be a $(1 + \log n)$-approximation for $\text{SC}$, where $n = |X|$. It was later shown by [29] that this performance is optimal up to lower order terms. Namely, they showed that a polynomial-time $(1 - \epsilon) \log n$ approximation ratio is achievable only if NP has quasi-polynomial-time algorithms.

For $k$-$\text{SC}$, the greedy algorithm presented in [50, 60] is a $(1 + \log k)$-approximation. [17] extended this result to the weighted version of the problem $k$-$\text{WSC}$. [82] showed that this performance is optimal up to lower order terms by showing that it is NP-hard to approximate $k$-$\text{SC}$ better than $\log k - O(\log \log k)$. 
Appendix B

Proofs for Section 4.4

B.1 Proof of Lemma 6

For any (possibly randomized) algorithm $A$, we use the random variable $W(A)$ to denote the aptitude of the candidate chosen by $A$. Let $A_t$ be any algorithm with the optimal competitive ratio $\rho(A_t)$ for $t$-ASP. First, $\rho(A_t) \geq \frac{1}{t}$ by the trivial algorithm $A^*_t$ which ignores all scores, and commits to one of the $t$ secretaries uniformly at random. Indeed,

$$
E[W(A^*_t)] = \frac{1}{t} \sum_{\tau=1}^{t} w_{\tau} = \frac{||w||_1}{t} \geq \frac{||w||_\infty}{t} = \frac{\text{OPT}}{t}.
$$

Therefore, all we need to show is that $\rho(A_t) \leq \frac{1}{t}$, which we will do by induction.

**Base Case:** For $t = 1$ the problem is trivial; there is only one candidate, so hiring the candidate gives the optimal solution and the optimal competitive ratio is $\frac{1}{t} = \frac{1}{1} = 1$.

**Induction Step:** Suppose that $\rho(A_t) = \frac{1}{t}$ for some $t \geq 1$. For the $(t+1)$-ASP, at the first timestep, $A_{t+1}$ decides to hire the first candidate with some probability $p$. If the first candidate is hired, the algorithm collects utility $w_1$. If the first candidate is not hired, then the algorithm must solve a secretary problem on the remaining $t$ candidates. Therefore,

$$
E[W(A_{t+1})] = pw_1 + (1 - p)E[W(A_t)].
$$

Therefore the competitive ratio of $A_{t+1}$ is

$$
\rho(A_{t+1}) = \min_{w \in \mathbb{R}^{t+1}} \frac{pw_1}{||w||_{\infty}} + (1 - p) \frac{E[W(A_t)]}{||w||_{\infty}}.
$$

(B.1)
Let $z \in \mathbb{R}_+^t$ be a worst-case instance for $A_t$ so that $E[W(A_t)] = \frac{\text{OPT}_t}{t}$. Now let $[w_2, w_3, ..., w_{t+1}]^T = \alpha z$ for some $\alpha > 0$. Sending $w_1 \to \infty$, (B.1) converges to $p$. Sending $\alpha \to \infty$, eventually we have $\|w\|_\infty = \|z\|_\infty$, and (B.1) converges to $(1 - p)\rho(A_t) = \frac{1-p}{t}$. Therefore,

$$\rho(A_{t+1}) \leq \min \left( p, \frac{1-p}{t} \right).$$

The best choice of $p$ to maximize this upper bound is $p = \frac{1}{t+1}$, which yields $\rho(A_{t+1}) = \frac{1}{t+1}$. This completes the induction step, and thus completes the proof.
B.2 Visualizing the Reduction from $d$-ASP to Online-$(k,d)$-Max-Matching

Figure B.1: An example of a shareability hypergraph $G \in \mathcal{G}$ evolving in time. This example has $d = 9$. Vertices arrive one per timestep on the right, and depart one per timestep from the left. A vertex is red during the last timestep it is in the system. For each $t$, $e_t$ is revealed at time $t + d - 1$, when its last vertex appears. It also becomes critical at this time, since $t \in e_t$ is about to leave the system. Therefore, from timestep $d - 1$ onward, one edge, along with its weight, is revealed each timestep, and can only be included into the matching at that timestep. Furthermore, at most one edge can be in the matching, since all edges contain the node $d - 1 = 8$. Hence, the family $\mathcal{G}$ encodes the $d$-ASP problem.
B.3 Proof of Theorem 2

Proof Outline: Our proof has three main steps. First, we identify Property 1 such that for any \( \mathcal{A} \) satisfying Property 1, \( \text{Randomized-Batching}(\cdot, \mathcal{A}) \) is \( \frac{1}{d} \)-competitive. In Section B.3, we present the \( \text{Depth-} k \text{-Greedy} \) algorithm and prove that it satisfies Property 1. Finally, in Section B.3, we prove that for \( \mathcal{A} = \text{Depth-} k \text{-Greedy}, \text{Randomized-Batching}(\cdot, \mathcal{A}) \) runs in polynomial time.

Let \( M^* \) be a maximum weight matching for an instance \( H = (V, E, w) \) of \( \text{Online-}(k, d)\text{-Max-Matching} \). Partition \( M^* \) into two disjoint subsets \( M^*_s \cup M^*_\ell \) defined via
\[
M^*_s := \{ e \in M^* : \text{diam}(e) \leq d - k \}
\]
\[
M^*_\ell := \{ e \in M^* : \text{diam}(e) > d - k \},
\]
where \( M^*_s \) contains all of the edges of \( M^* \) whose diameter is at most \( d - k \), and \( M^*_\ell \) contains the other edges. First note that \( w(M^*) = w(M^*_s) + w(M^*_\ell) \). Recalling the definition of batches, for each \( z \in \{0, 1, ..., d - 1\} \) we use \( E_z := \bigcup_i E_{z,i} \).

Now suppose we have a procedure \( \mathcal{A} \) that satisfies the following property.

**Property 1.** Let \( M^* = M^*_s \cup M^*_\ell \) be a maximum weight matching in \( H \). For every pair \( z, i \), the algorithm \( \mathcal{A} \) can find a matching \( M_{z,i} \subset E_{z,i} \) so that
\[
w(M_{z,i}) \geq w(M^*_\ell \cap E_{z,i}) + \frac{1}{k} w(M^*_s \cap E_{z,i}). \tag{B.2}
\]

Next we show that if \( \mathcal{A} \) is a \( k\text{-Max-Matching} \) algorithm which satisfies Property 1, then Algorithm 3 using \( \mathcal{A} \) as a subroutine is \( \frac{1}{d} \)-competitive for \( \text{Online-}(k, d)\text{-Max-Matching} \).

\[
\mathbb{E}[w(M)] = \sum_{z=0}^{d-1} \mathbb{P}[Z = z] \mathbb{E}[w(M)|Z = z] \tag{B.3}
\]
\[
= \frac{1}{d} \sum_{z=0}^{d-1} \sum_{i=0}^{n/d} w(M_{z,i}) \]
\[
\geq \frac{1}{d} \sum_{z=0}^{d-1} \sum_{i=0}^{n/d} \left[ w(M^*_\ell \cap E_{z,i}) + \frac{1}{k} w(M^*_s \cap E_{z,i}) \right] \]
\[
= \frac{1}{d} \sum_{z=0}^{d-1} \left[ w(M^*_\ell \cap E_z) + \frac{1}{k} w(M^*_s \cap E_z) \right]
\]
\[
= \frac{1}{d} \sum_{z=0}^{d-1} \sum_{e \in M^*_\ell \cap E_z} w(e) + \frac{1}{kd} \sum_{z=0}^{d-1} \sum_{e \in M^*_s \cap E_z} w(e)
\]
\[ w(M_0) = w(M^*_\ell \cap E_{z,i}) + w(M) \]

\[ \geq w(M^*_\ell \cap E_{z,i}) + \frac{1}{k} \max_{M' \text{ is a matching in } W_{\text{res}}} w(M') \]

\[ \geq w(M^*_s \cap E_{z,i}) + \frac{1}{k} w(M^*_s \cap E_{z,i}), \]

where (a) is because Greedy is a \( \frac{1}{k} \)-approximation for \( k \)-Max-Matching, and (b) is because \( M_s \cap E_{z,i} \) is a matching in \( W_{\text{res}} \).

We cannot use this procedure directly because we do not know \( M^*_\ell \cap E_{z,i} \). However, if there are not too many candidates for what \( M^*_\ell \cap E_{z,i} \) can be, we can apply the above procedure to each of the candidates and return the resulting matching with the largest weight. To this end we have the following observation.

Observation 5. Since \( W_{z,i} \) is a batch, its vertices are ordered, so let \( L \) and \( R \) be the sets of the first and last \( k \) vertices of \( W_{z,i} \) respectively. For \( e \in M^* \cap E_{z,i} \), if \( e \cap L = \emptyset \) or \( e \cap R = \emptyset \), then \( \text{diam}(e) \leq d - k \), and hence \( e \notin M^*_\ell \). Equivalently, for any \( e \in M^*_\ell \cap E_{z,i} \), we have \( e \cap L \neq \emptyset \) and \( e \cap R \neq \emptyset \).

Hence \( M^*_\ell \cap E_{z,i} \) is a member of the following set:

\[ \mathcal{M}_{L,R}(W_{z,i}) := \{ M \text{ a matching in } W_{z,i} : e \cap L \neq \emptyset, e \cap R \neq \emptyset \forall e \in M \}. \]

In other words, \( M^*_\ell \cap E_{z,i} \) is a matching in \( W_{z,i} \) that is restricted to \( L, R \) where we define restricted
matchings below.

**Definition 31** (Restricted Matchings). Given a hypergraph \( H = (V, E) \) and two disjoint subsets \( L, R \subset V \), \( L \cap R = \emptyset \), a collection of edges \( M \subset E \) is a matching is restricted to \( L, R \)

1. \( M \) is a matching in \( H \).
2. For any \( e \in M \), \( e \cap L \neq \emptyset \) and \( e \cap R \neq \emptyset \).

Finally, we use \( \mathcal{M}_{L,R}(H) \) to denote the set of all matchings in \( H \) that are restricted to \( L, R \).

When \( L, R \) denote the first and last \( k \) vertices of a batch respectively, since \( |L| = |R| = k \) is a constant, a matching in \( \mathcal{M}_{L,R}(W_{z,i}) \) can have at most \( k \) edges. As a result, \(|\mathcal{M}_{L,R}(W_{z,i})| \leq \sum_{i=1}^{k} (|E_{z,i}|) \leq k|E_{z,i}|^k \). This motivates the *Depth-k-Greedy* algorithm which does the following: For each \( M \in \mathcal{M}_{L,R}(W_{z,i}) \), extend \( M \) to a maximal matching by adding edges via *Greedy*, and then among the \(|\mathcal{M}_{L,R}(W_{z,i})|\) resulting maximal matchings, return the one with the largest weight.

Using a calculation very similar to (B.4), we show that *Depth-k-Greedy* satisfies Property 1. To this end, for any matching \( M \) we use \( V(M) \) to denote the vertices that are matched by \( M \). If \( M_{z,i} \) is the matching returned by *Depth-k-Greedy*, we have

\[
w(M_{z,i}) = \max\limits_{M \in \mathcal{M}_{L,R}(W_{z,i})} w(M) + w(\text{Greedy}(V_{z,i} \setminus V(M), E_{z,i})) \\
\geq w(M^*_f \cap E_{z,i}) + w(\text{Greedy}(V_{z,i} \setminus V(M^*_f \cap E_{z,i}), E_{z,i})) \\
\geq w(M^*_f \cap E_{z,i}) + \frac{1}{k} \max_{M'^{z,i} \text{ is a matching in } W_{z,i}} w(M') \\
\geq w(M^*_f \cap E_{z,i}) + \frac{1}{k} w(M^*_s \cap E_{z,i}),
\]

where (a) is due to \( M^*_f \cap E_{z,i} \in \mathcal{M}_{L,R}(W_{z,i}) \), (b) is due to *Greedy* being a \( \frac{1}{k} \)-approximation for \( k\)-Max-Matching, and (c) is because \( (M^*_f \cap E_{z,i}) \cup (M^*_s \cap E_{z,i}) = M^* \cap E_{z,i} \) is a matching in \( W_{z,i} \). Hence *Depth-k-Greedy* satisfies Property 1. To complete the proof, we next show that the runtime of *Depth-k-Greedy* is polynomial in the size of the problem instance \( H \).

**Runtime of Randomized-Batching with Depth-k-Greedy:**

Finally, we need to show that the runtime of *Randomized-Batching* with \( \mathcal{A} = \text{Depth-k-Greedy} \) is polynomial in the size of \( H \). Since \( |L| = k \), any matching in \( \mathcal{M}_{L,R}(W_{z,i}) \) can have at most \( k \) hyperedges. From this, we conclude that

\[
|\mathcal{M}_{L,R}(W_{z,i})| \leq \sum_{i=1}^{k} (|E_{z,i}|) \leq k \left( \frac{|E_{z,i}|}{k} \right) \leq k \left( \frac{|E_{z,i}|}{e k} \right)^k
\]
APPENDIX B. PROOFS FOR SECTION 4.4

For a batch $W_{z,i}$, it takes $c|E_{z,i}|\log|E_{z,i}|$ time to sort the edges by weight, for some constant $c$. For each $M \in \mathcal{M}_{L,R}(W_{z,i})$, given that the edges are already sorted, extending $M$ to a maximal matching via \textbf{Greedy} requires $O(k|E_{z,i}|)$ time, since for each edge, it takes $O(k)$ time to check if it is disjoint from the current matching. Doing this for all members of $\mathcal{M}_{L,R}(W_{z,i})$ would take

$$c|E_{z,i}|\log|E_{z,i}| + |\mathcal{M}_{L,R}(W_{z,i})| \cdot k|E| = c|E_{z,i}|\log|E_{z,i}| + \frac{k^2}{(ek)^k} |E_{z,i}|^{k+1}$$

time. Therefore the expected runtime of \textsf{Randomized-Batching}(., $\mathcal{A}$) with $\mathcal{A} = \text{Depth-}k$-$\textbf{Greedy}$ is

$$\frac{1}{d} \sum_{z=0}^{d-1} \sum_i \left( c|E_{z,i}|\log|E_{z,i}| + \frac{k^2}{(ek)^k} |E_{z,i}|^{k+1} \right) = O \left( \frac{1}{d} \sum_{z=0}^{d-1} \sum_i |E_{z,i}|^{k+1} \right)$$

$$\leq O \left( \frac{1}{d} \sum_{z=0}^{d-1} |E_z|^{k+1} \right)$$

$$\leq O \left( \max_{0 \leq z \leq d-1} |E_z|^{k+1} \right)$$

$$\leq O \left( |E|^{k+1} \right).$$

This completes the proof.
Appendix C

Proofs for Section 4.5

C.1 Proof of Lemma [10]

We partition the set of maximal paths in $(V, M \cup M^*, w)$ into four categories, and treat each of the categories separately.

1. Length 1 paths.
2. Type 1 paths.
3. Even length paths.
4. Type 2 paths of length at least 3.

Where type 1 and type 2 paths are defined as follows:

Definition 32 (Path Types). A path in $(V, M \cup M^*, w)$ is type 1 if it has an odd length, and contains more edges from $M$ than from $M^*$. A path is type 2 if it has odd length and contains more edges from $M^*$ than from $M$.

Proving Lemma [10] for paths in Categories 1 and 2 are easy, so we will do this directly. For Categories 3 and 4, we will prove the lemma by induction on the path length.

C.1.1 Maximal Paths of Length 1

Consider a path of length 1 in $M \cup M^*$, i.e. an edge $e = (i, j) \in E$. There are two cases to consider: 1) $e \in M$ and 2) $e \in M^* \setminus M$.

Case 1: First note that either $(i, j) \in M^*$, or both $i, j$ are unmatched under $M^*$. To see this, if $(i, j') \in M^*$ for some $j \neq j'$, then the path $\{(j, i)\}$ would not be maximal since in particular the
path \{(j, i), (i, j')\} \subset M \cup M^*\). From this we can conclude that that \(v_i^+ \geq \theta_{ij} w_i, v_j^+ \geq \theta_{ij} w_j\). Finally,

\[ v_i + v_j = w(i,j) = \theta_{ij} w_i + \theta_{ij} w_j \leq v_i^+ + v_j^+. \]

**Case 2:** Now we consider the case where \(e \in M^* \setminus M\). We argue by contradiction that \(e \not\in E_\theta\).

Suppose \(e \in E_\theta\). We can immediately say that neither \(i\) nor \(j\) matched under \(M\), because if any of the vertices of \(e\) are matched under \(M\), then this path would have more than 1 edge. Since none of the vertices of \(e\) are matched in \(M\), then \(M \cup \{e\}\) is a matching in \((V, E_\theta)\). However, this contradicts the fact that \(M\) is a maximal matching in \((V, E_\theta)\). Therefore if \(e \in M^*\), it must be the case that \(e \not\in E_\theta\). From this we see that:

\[ v_i + v_j \leq w_i + w_j = \frac{w(i,j)}{\theta_{ij}} = \frac{v_i^+ + v_j^+}{\theta_{ij}} \leq \frac{3}{2} (v_i^+ + v_j^+). \]

Where \(\theta_{ij} \geq \theta\) is because \(e \not\in E_\theta\), and the last equality is due to \(\theta = \frac{2}{3}\).

**C.1.2 Type 1 Paths**

Let \(P\) be a type 1 path on the vertices \(\{1, 2, \ldots, 2\ell\}\) where \((2k + 1, 2k + 2)\}_{k=0}^{\ell-1} \in M\) and \((2k, 2k + 1)\}_{k=1}^{\ell-1} \in M^*\). Since \(P\) contains more edges from \(M\) than \(M^*\), the set of vertices in \(V(P)\) matched by \(M^* \cap P\) is a subset of vertices matched by \(M \cap P\). Concretely, note that \(M\) matches all vertices \(\{1, 2, \ldots, 2\ell\}\), whereas \(M^*\) matches all of the vertices except \(\{1, 2\ell\}\). We thus see that

\[
\sum_{k=1}^{2\ell} v_k = v_1 + v_{2\ell} + \sum_{k=2}^{2\ell-1} v_k \\
\leq v_1^+ + v_{2\ell}^+ + \sum_{k=2}^{2\ell-1} \theta w_k \\
\leq v_1^+ + v_{2\ell}^+ + \sum_{k=2}^{2\ell-1} 2\theta v_k^+ \\
\leq 2\theta \sum_{k=1}^{2\ell} v_k^+.
\]

We conclude by noting that \(\theta = \frac{2}{3}\) implies that \(2\theta = \frac{4}{3} \leq \frac{3}{2}\).

**C.1.3 Even Length Paths: Base Case**

We argue by induction on the length of even paths. The base case is when the path \(P = \{(a, b), (b, c)\}\) has 2 edges where \((a, b) \in M, (b, c) \in M^*\). We will use the following observation:

**Observation 6.** *(Fractions and convex combinations)* If \(x > y > 0\) and \(z > 0\), then \(\frac{x+z}{y+z} \leq \frac{x}{y}.*
Proof of Observation 4. This follows from the following algebra:

\[
x > y > 0, z > 0 \implies xz > yz \\
\implies xy + xz > xy + yz \\
\implies x(y + z) > y(x + z) \\
\implies \frac{x}{y} > \frac{x + z}{y + z}.
\]

With this, observe that

\[
\frac{v_a + v_b + v_c}{v_a^* + v_b^* + v_c^*} = \frac{w(a, b) + w_c}{w_a + w(b, c)} \leq \frac{w(a, b) + w_c}{w_a + \max(w_b, w_c)} \tag{C.1}
\]

If the numerator in (C.1) is smaller than the denominator, we are done. Therefore from this point onward assume that the numerator is larger. There are two cases. If \(w_c \leq w_b\), then we apply this bound to the numerator. If \(w_b < w_c\), then we apply Observation 6 with \(z = w_c - w_b\). In either case, we get

\[
\frac{v_a + v_b + v_c}{v_a^* + v_b^* + v_c^*} \leq \frac{w(a, b) + w_b}{w_a + w_b} \\
= \frac{\theta_{ab}w_a + \theta_{ab}w_b + w_b}{w_a + w_b} \\
= 1 + \frac{(\theta_{ab} - 1)w_a + \theta_{ab}w_b}{w_a + w_b} \\
\leq 1 + \frac{(\theta_{ab} - 1)\left(1 - \frac{\theta_{ab}}{\theta_{ab}}\right)w_b + \theta_{ab}w_b}{w_a + w_b} \\
= 1 + \frac{-(1 - \theta_{ab})^2w_b + \theta_{ab}^2w_b}{w_b} \\
= 1 + \theta_{ab}^2 - (1 - 2\theta_{ab} + \theta_{ab}^2) \\
= 2\theta_{ab} \leq 2\theta = \frac{4}{3} \leq \frac{3}{2}.
\]

We get (a) because the numerator is a decreasing function of \(w_a\) and the denominator is an increasing function of \(w_a\), which means the bound is maximized when \(w_a\) is minimized. Furthermore, \(\theta_{ab}w_a + \theta_{ab}w_b = w(a, b) \geq w_b\) implies that \(w_a \geq \left(\frac{1 - \theta_{ab}}{\theta_{ab}}\right)w_b\). This establishes the base case for even length paths.
C.1.4 Induction Step on the Path Length

Since we already bounded the cost contribution of type 1 paths in Section C.1.2, we focus on paths that are not type 1 in this section. In particular, we focus on paths that have at least as many edges from $M^*$ as they do from $M$. Note that any such path has the following property: Either the first edge in the path or the last edge in the path belong to $M^*$. In this section, we will without loss of generality assume that the first edge of the path belongs to $M^*$. If the first edge belongs to $M$, then it must be the case that the last edge belongs to $M^*$, and we can just reverse the ordering of the edges.

Now we begin the induction step. Assume that for all paths $P$ of length $n$ whose vertices are $\{0, 1, ..., n\}$ and whose first edge belongs to $M^*$, we have

$$w(M; P) \leq \frac{3}{2} w(M^*; P).$$

Now let $P'$ be a length $n + 2$ path whose vertices are $\{0, 1, 2, ..., n + 1, n + 2\}$ and whose first edge belongs to $M^*$. Define $P$ to be the path containing the edges $\{(i, i + 1)\}_{i=2}^{n+1}$. Note that the first edge of $P$ belongs to $M^*$, and the length of $P$ is $n$. We can write

$$w(M; P') = \sum_{i=0}^{n+2} v_i = v_0 + v_1 + (v_2 - w_2) + w_2 + \sum_{i=3}^{n+2} v_i$$

$$= v_0 + v_1 + (v_2 - w_2) + \underbrace{w(M; P)}_{\text{term 1}}$$

where term 1 is because of the following. Vertex 2 is unmatched in $P \cap M$, so it contributes a cost of $w_2$. However, $(1, 2) \in (P' \setminus P) \cap M$ reduces the cost contribution of vertex 2 from $w_2$ to $v_2$.

Similarly, we can write

$$w(M^*; P') = v_0^* + v_1^* + \sum_{i=2}^{n+2} v_i^*$$

$$= v_0^* + v_1^* + w(M^*; P).$$

Since $P$ is a length $n$ path whose first edge belongs to $M^*$, we can use the induction hypothesis to conclude that

$$w(M; P) \leq \frac{3}{2} w(M^*; P).$$

Next, we will show that

$$\frac{v_0 + v_1 + (v_2 - w_2)}{v_0^* + v_1^*} \leq \frac{3}{2}. \quad (C.2)$$
Once we show (C.2), we can conclude the proof as follows:

\[ w(M; P') = v_0 + v_1 + (v_2 - w_2) + w(M; P) \]
\[ \leq \frac{3}{2} (v_0^* + v_1^*) + \frac{3}{2} w(M^*; P) \]
\[ = \frac{3}{2} w(M^*; P') \]

Where the inequality is obtained by applying (C.2) and the induction hypothesis on \( P \). Therefore, all that remains is to prove (C.2). Observe that

\[ \frac{v_0 + v_1 + (v_2 - w_2)}{v_0^* + v_1^*} = \frac{w_0 + \theta_{12} w_1 + (\theta_{12} - 1) w_2}{\theta_{01} w_0 + \theta_{01} w_1} \leq \frac{w_0 + \theta_{12} w_1 + (\theta_{12} - 1) w_2}{\max(w_0, w_1)}. \]

We now use an approach similar to what was done in Section C.1.3. If the numerator is smaller than the denominator, then we are done. If it is not, then there are two cases. If \( w_0 \leq w_1 \), we apply this bound to the numerator. If \( w_1 < w_0 \), then we apply observation 6 with \( z = w_0 - w_1 \). In both cases, we get

\[ \frac{v_0 + v_1 + (v_2 - w_2)}{v_0^* + v_1^*} \leq w_1 + \theta_{12} w_1 + (\theta_{12} - 1) w_2 = w_1 + \theta_{12} w_1 + (\theta_{12} - 1) w_2 = \frac{\theta_{12} + \theta_{12}^2 - (1 - \theta_{12})^2}{\theta_{12}} \]
\[ = \frac{3\theta_{12} - 1}{\theta_{12}} \]
\[ \leq \frac{3\theta - 1}{\theta} = \frac{3}{2}. \]

Where \((a)\) is because the numerator is a decreasing function of \( w_2 \), so the bound is maximized when \( w_2 \) is minimized. Next, \( \theta_{12} w_1 + \theta_{12} w_2 = w(1, 2) \geq w_1 \) implies that \( w_2 \geq \left( \frac{1 - \theta_{12}}{\theta_{12}} \right) w_1 \). Finally, \((b)\) is because \( \theta_{12} \leq \theta \) since \((1, 2) \in M \subset E_\theta \), and the function \( \frac{3x-1}{x} \) is increasing on \([\frac{1}{2}, \frac{2}{3}]\).

### C.1.5 Base Case for Type 2 Paths of Length at least 3

Our base case is a path \( P = \{(a, b), (b, c), (c, d)\} \) where \((a, b), (c, d) \in M^* \) and \((b, c) \in M \). There are two cases we need to consider:

1. At most one of \((a, b)\) or \((c, d)\) is in \( E_\theta \).
2. \((a, b), (c, d) \in E_\theta \).
Remark 18. One may be tempted to use the result for type 2 paths of length 1 from Section C.1.1 together with the induction step in Section C.1.4 to establish a result for all type 2 paths. This, however, will not work. In Section C.1.1 when discussing type 2 paths of length 1, we used the fact that the edge $e \in M^*$ cannot be in $E_{\theta}$. Therefore applying the result from Section C.1.1 together with the induction step in Section C.1.4 will only prove the result for type 2 paths that contain at least one edge from $E \setminus E_{\theta}$. There can in general be type 2 paths whose edges are all contained in $E_{\theta}$. Case 2 of this section addresses such paths. Once we address case 2, then we can apply the induction argument from Section C.1.4 to establish the desired result for all type 2 paths.

We now prove the desired result for each of the two cases. This section, together with the induction argument from Section C.1.4 establishes the desired result for all type 2 paths.

Case 1: In this case, either $(a, b) \notin E_{\theta}$ or $(c, d) \notin E_{\theta}$. Without loss of generality assume that $(a, b) \notin E_{\theta}$ (otherwise simply relabel the vertices). Since $(a, b) \notin E_{\theta}$, we have $w((a, b)) \theta w_a + w_b > \theta$. We then observe

$$v_a + v_b \leq w_a + w_b \leq \frac{1}{\theta}w(i, j) \leq \frac{v_a^* + v_b^*}{\theta} = \frac{3}{2}(v_a^* + v_b^*).$$

We now have a path $P_0 = \{(a, b)\}$ where $w(M; P_0)$ is no more than $\frac{3}{2}w(M^*; P_0)$. Applying the induction argument from Section C.1.4, we can conclude that

$$\frac{w(M; P)}{w(M^*; P)} \leq \frac{3}{2}.$$ 

This completes the proof of Case 1.

Case 2: In this case we have a path $P = \{(a, b), (b, c), (c, d)\}$ where $(a, b), (c, d) \in M^* \cap E_{\theta}$ and $(b, c) \in M$. We first show that the first vertex among $\{a, b, c, d\}$ to become critical must be either $b$ or $c$. Note that if $a$ was the first vertex among $\{a, b, c, d\}$ to become critical, then in particular $a$ becomes critical before either $b, c$ becomes critical. Since $(b, c) \in M$, no vertex that becomes critical before $b$ and $c$ choose to match with either of them. Therefore, by the time $a$ becomes critical, both $b, c$ are available, meaning that $a$ should be matched, since in particular it can match to $b$. However, since $a$ is unmatched in $M$, it cannot be the case that $a$ is the first vertex among $\{a, b, c, d\}$ to become critical. The same argument concludes that $d$ cannot be the first to become critical. This means that the first among these vertices to become critical is either $b$ or $c$. Thus without loss of generality assume that $b$ is the first vertex to become critical. When $b$ becomes critical, both $a, c$ are available. Since $b$ chooses to match with $c$, we can conclude that $\theta_{bc} \leq \theta_{ab}$. 


We now observe that

\[
\frac{w(M; P)}{w(M^*; P)} = \frac{w_{a} + w(b, c) + w_{d}}{w(a, b) + w(c, d)}
\]

\[
= \frac{w_{a} + \theta_{bc}w_{b} + \theta_{bc}w_{c} + w_{d}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + \theta_{cd}w_{c} + \theta_{cd}w_{d}}
\]

\[
\leq \frac{w_{a} + \theta_{bc}w_{b} + \theta_{bc}w_{c} + w_{d}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + \max(w_{c}, w_{d})}
\]

If the numerator is smaller than the denominator, then we are done. So from now on we assume that the numerator is larger. There are two cases to consider. If \(w_{c} > w_{d}\), then we can bound the numerator by \(w_{a} + \theta_{bc}w_{b} + \theta_{bc}w_{c} + w_{c}\). If \(w_{d} > w_{c}\), then applying Observation 6 with \(z = w_{d} - w_{c}\), we get

\[
\frac{w_{a} + \theta_{bc}w_{b} + \theta_{bc}w_{c} + w_{d}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + \max(w_{c}, w_{d})} = \frac{w_{a} + \theta_{bc}w_{b} + \theta_{bc}w_{c} + w_{d} + (w_{d} - w_{c})}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c} + (w_{d} - w_{c})} \leq \frac{w_{a} + \theta_{bc}w_{b} + \theta_{bc}w_{c} + w_{d} + (w_{d} - w_{c})}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}}
\]

So in either case, we have

\[
\frac{w(M; P)}{w(M^*; P)} \leq \frac{w_{a} + \theta_{bc}w_{b} + \theta_{bc}w_{c} + w_{d}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}}
\]

There are two cases to consider. If \(w_{a} \geq w_{c}\), then

\[
\frac{w(M; P)}{w(M^*; P)} \leq \frac{w_{a} + \theta_{ab}w_{b} + \theta_{ab}w_{c} + w_{d}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}}
\]

\[
= 1 + \frac{(1 - \theta_{ab})w_{a} + \theta_{ab}w_{b}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}}
\]

\[
\leq 1 + \frac{(1 - \theta_{ab})w_{a}}{w_{a} + w_{c}} + \theta_{ab}\frac{w_{c}}{w_{a} + w_{c}}
\]

\[
\leq 1 + \frac{1 - \theta_{ab}}{2} + \frac{\theta_{ab}}{2} = \frac{3}{2}
\]

Where (a) is due to \(\theta_{bc} \leq \theta_{ab}\). The other case is that \(w_{c} > w_{a}\). In this case, we have

\[
\frac{w(M; P)}{w(M^*; P)} = \frac{w_{a} + \theta_{bc}w_{b} + \theta_{bc}w_{c} + w_{c}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}} \leq \frac{w_{a} + \theta_{ab}w_{b} + \theta_{bc}w_{c} + w_{c}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}}
\]

\[
= 1 + \frac{(1 - \theta_{ab})w_{a} + \theta_{bc}w_{c}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}}
\]

\[
= 1 + \frac{(1 - \theta_{ab})w_{a}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}} + \frac{\theta_{bc}w_{c}}{\theta_{ab}w_{a} + \theta_{ab}w_{b} + w_{c}}
\]

\[
= 1 + \frac{\theta_{ab}w_{a}}{w_{a} + \theta_{ab}w_{b} + w_{c}} \leq 1 + \frac{1}{2}
\]

Where (a) is obtained by bounding \(\theta_{bc}w_{b} \leq \theta_{ab}w_{b}\). We will show that term 1 is maximized when
$w_b$ takes its smallest possible value. To this end, note that term 1 is a convex combination of $\frac{1 - \theta_{ab}}{\theta_{ab}}$ and $\frac{\theta_{ab} w}{\theta_{ab} w + w_c}$. Since $w_b \geq \frac{1 - \theta_{bc}}{\theta_{bc}} w_c$ and $\theta_{bc} \leq \theta_{ab}$, we have

$$\frac{\theta_{bc} w_c}{\theta_{ab} w + w_c} \leq \frac{\theta_{bc} w_c}{\theta_{bc} w + w_c} \leq \frac{\theta_{bc} w_c}{\theta_{bc} w + w_c} = \frac{\theta_{bc}}{2 - \theta_{bc}} \leq \frac{\theta_{ab}}{2 - \theta_{ab}}. \quad (C.3)$$

Next, we note that since $\theta_{ab} \leq \theta = \frac{2}{3}$, we have $\frac{\theta_{ab}}{2 - \theta_{ab}} \leq \frac{1 - \theta_{ab}}{\theta_{ab}}$. So $\frac{1 - \theta_{ab}}{\theta_{ab}}$ is larger than $\frac{\theta_{ab} w}{\theta_{ab} w + w_c}$ for any valid value of $w_a, w_b, w_c$. Since $\frac{\theta_{ab} w}{\theta_{ab} w + w_c}$ and $\frac{\theta_{ab} w}{\theta_{ab} w + w_c}$ are decreasing functions of $w_b$, we see that for any valid values of $w_a, w_c$, term 1 is maximized when $w_b$ takes its smallest possible value, that is to say $w_b = \max \left( \frac{1 - \theta_{ab}}{\theta_{ab}}, \frac{1 - \theta_{bc}}{\theta_{bc}} w_c \right)$. From this we get:

$$\frac{w(M; P)}{w(M^*; P)} \leq 1 + \frac{(1 - \theta_{ab}) w_a}{\theta_{ab} w_a} \frac{\theta_{ab} w_a}{\theta_{ab} w_a + \theta_{ab} w_b + w_c} + \frac{\theta_{bc} w_c}{\theta_{bc} w_b + w_c} \frac{\theta_{bc} w_c}{\theta_{bc} w_b + w_c}$$

\[(a)\]

$$\leq 1 + \frac{(1 - \theta_{ab}) w_a}{\theta_{ab} w_a} \frac{\theta_{ab} w_a + \theta_{ab} \left( \frac{1 - \theta_{ab}}{\theta_{ab}} \right) w_a + w_c}{\theta_{ab} w_a + \theta_{ab} w_b + w_c} + \frac{\theta_{bc} w_c}{\theta_{bc} w_b + w_c} \frac{\theta_{bc} w_c}{\theta_{bc} w_b + w_c}$$

\[(b)\]

$$= 1 + \frac{(1 - \theta_{ab})}{\theta_{ab}} \frac{\theta_{ab} w_a}{w_a + w_c} + \frac{\theta_{ab}}{2 - \theta_{ab}} \left( \frac{1 - \theta_{ab}}{w_a + w_c} \right)$$

\[(c)\]

$$\leq 1 + \frac{(1 - \theta_{ab})}{\theta_{ab}} \frac{\theta_{ab} w_a}{w_a + w_c} + \frac{\theta_{ab}}{2 - \theta_{ab}} \left( \frac{1 - \theta_{ab} w_a}{w_a + w_c} \right)$$

$$= 1 + \frac{1 - \theta_{ab}}{2} + \frac{\theta_{ab}}{2} = \frac{3}{2}.$$ 

We get (a) from the chain of inequalities in (C.3). We get (b) and (c) by increasing the weight of $\frac{1 - \theta_{ab}}{\theta_{ab}}$ in the convex combination and reduce the weight of $\frac{\theta_{ab}}{2 - \theta_{ab}}$ by the same amount. This leads to an upper bound since $\frac{1 - \theta_{ab}}{\theta_{ab}} \geq \frac{\theta_{ab}}{2 - \theta_{ab}}$. This completes the base case for case 2.

\footnote{By this we simply mean $\theta_{bc} \leq \theta_{ab} \leq \theta$.}
Appendix D

Proofs for Section 5

D.1 Proof of Theorem 7

D.1.1 Setup

Given an instance $G$ of $\text{OBMRR}(p)$ and a set $R \subseteq B$, we use $(G, R)$ to denote the outcome where the set of vertices that choose to reserve is $R$. Our proof is based on a monotone property of $\text{Reservation-Priority}$: If $R \subseteq \bar{R}$, then the set of offline vertices matched in the instance $(G, R)$ is a subset of the offline vertices matched in $(G, \bar{R})$. This property is result of Lemma 12, which we prove in Appendix D.2.

We will also use the following observation to reason about the size of matchings.

**Observation 7.** Let $G$ be a graph with a perfect matching $M^*$ where $|M^*| = n$. For a maximal matching $M$ in $G$, let $\theta$ be the fraction of edges in $M^*$ whose endpoints are both matched under $M$. Then $|M| \geq n \frac{1+\theta}{2}$.

**Proof of Observation 7.** Since $M$ is a maximal matching, for each $e \in M^*$, at least one of the endpoints of $e$ is matched under $M$. Since $\theta n$ edges in $M^*$ have both of its endpoints matched, the total number of vertices matched by $M$ is $2\theta n + (1-\theta)n = n(1+\theta)$. Therefore the number of edges in $M$ must be at least $n \frac{1+\theta}{2}$. \qed

D.1.2 Proof

Let $M^*$ be a perfect matching in the bipartite graph $G = (A, B, E)$ and let $M$ be the matching produced by $\text{Reservation-Priority}$ on the instance $(G, R)$. Since $\text{Reservation-Priority}$ always outputs a maximal matching, it matches at least $\frac{n}{2}$ vertices on the instance $(G, \emptyset)$. Let $A_0$ be a set of $\frac{n}{2}$ such vertices. By the monotone property, we know $A_0 \subseteq A_R$. Consider $a \in A_0$. If $M^*(a)$ is matched, then $(a, M^*(a))$ is an edge in $M^*$ whose endpoints are both matched by $M$. 

126
Since Reservation-Priority matches all reserved vertices, $M^*(a)$ is guaranteed to be matched if $M^*(a) \in R$. Thus the fraction of edges in $M^*$ whose endpoints are both matched by $M$ is at least $\frac{1}{n} |R \cap M(A_0)|$. Letting $R_1 := R \cap M(A_0)$, Observation 7 implies that $|M| \geq \frac{n}{2} + \frac{|R_1|}{2}$.

This means that $A_{R_1}$ must contain at least $\frac{|R_1|}{2}$ vertices outside of $A_0$. Denote such vertices by $C_1$. Because $R_1 \subset R$, we have $A_{R_1} \subset A_R$. Therefore, every reserve vertex in $R \cap M^*(A_{R_1})$ leads to an edge in $M^*$ whose endpoints are both matched under $M$. Therefore, the fraction of edges in $M^*$ whose endpoints are both matched under $M$ is at least

$$|R \cap M^*(A_{R_1})| = |R \cap M(A_0)| + |R \cap M(C_1)|,$$

where the equality is because $A_0, C_1$ being disjoint implies that $M^*(A_0), M^*(C_1)$ are also disjoint. Applying Observation 7 we see that

$$|M| \geq \frac{n}{2} + \frac{|R \cap M(A_0)|}{2} + \frac{|R \cap M(C_1)|}{2}.$$  

We now repeat the above arguments: letting $R_2 := R \cap M^*(A_{R_1})$, $A_{R_2}$ must have at least $\frac{1}{2} |R \cap M^*(A_1)|$ vertices outside of $A_{R_1}$, which we denote $C_2$. Any reserve vertices in $M^*(C_2)$ will increase the size of the matching.

In general, we define $\{R_k\}_{k=1}^\infty, \{C_k\}_{k=1}^\infty$ according to the following rules:

$$R_{k+1} = R \cap M^*(A_{R_k})$$
$$C_{k+1} = A_{R_{k+1}} \setminus A_{R_k},$$

where $|C_{k+1}| \geq \frac{1}{2} |R \cap M^*(C_k)|$ (due to Observation 7) with $R_1 = R \cap M(A_0), C_1 = A_{R_1} \setminus A_0$ as specified above. Note that by construction, the sets $\{C_k\}_{k=1}^\infty$ are disjoint and are disjoint from $A_0$. Thus the number of edges in $M^*$ whose endpoints are both matched under $M$ is at least

$$|R \cap M^*(A_{R_\infty})| = |R \cap M(A_0)| + \sum_{k=1}^\infty |R \cap M^*(C_k)|.$$  

Since every vertex in $B$ reserves independently with probability $p$, $|R \cap M(A_0)|$ has distribution $\text{Bin}(\frac{n}{2}, p)$, and $|R \cap M(C_1)|$ given $C_1$ has distribution $\text{Bin}(|C_1|, p)$. Similarly, for $k > 1$, $|R \cap M(C_k)|$ given $C_1, \ldots, C_k$ has the distribution $\text{Bin}(|C_k|, p)$. From this we see that $E [|R \cap M^*(C_k)| | C_1, \ldots, C_k] \geq$
\[ p|C_k| = \frac{p}{2} |R \cap M^*(C_{k-1})|. \] Therefore, by the tower property of expectation,

\[ \mathbb{E}[|R \cap M^*(C_k)|] \geq \left( \frac{p}{2} \right)^k \mathbb{E}[|R \cap M^*(A_0)|] = \left( \frac{p}{2} \right)^k \frac{n}{2} = n \left( \frac{p}{2} \right)^{k+1}. \]

Therefore the expected number of edges in \( M^* \) whose endpoints are both matched under \( M \) is at least

\[ \mathbb{E}\left[ |R \cap M(A_0)| + \sum_{k=1}^{\infty} |R \cap M^*(C_k)| \right] \geq \frac{np}{2} + \sum_{k=1}^{\infty} n \left( \frac{p}{2} \right)^{k+1} = \frac{n}{2} \frac{p/2}{1-p/2} = \frac{n}{2} \frac{p}{2-p}. \]

Invoking Observation \( \ref{obs:threshold} \) gives

\[ \mathbb{E}[|M|] \geq \frac{n}{2} + \frac{n}{2} \frac{p}{2-p} = \frac{n}{2-p}. \]

From this we conclude that Reservation-Priority is \( \frac{1}{2-p} \)-competitive for OBMRR\( (p) \).

### D.2 Proof of Lemma \( \ref{lem:reservation-priority} \)

We will show that \( A_R \subset A_{R'} \) when \( R' = R \cup \{ b \} \) for any \( b \notin R \). The result of Lemma \( \ref{lem:reservation-priority} \) is achieved by applying this result repeatedly for every \( b \in \bar{R} \setminus R \).

To this end, let \( R' = R \cup \{ b \} \) for some \( b \in \bar{R} \setminus R \). Let \( M^R, M^{R'} \) denote the matchings constructed by Reservation-Priority on the instances \( (G, R) \) and \( (G, R') \) respectively. Furthermore, let \( M^R_t, M^{R'}_t \) denote the intermediate matchings produced by the algorithm up to the \( t \)th timestep for the instances \( (G, R) \) and \( (G, R') \) respectively.

Let \( t_0 := \min \left\{ \tau \in \mathbb{N} : M^R(b_\tau) \neq M^{R'}(b_\tau) \right\} \) be the first time the algorithm makes a different action on instances \( (G, R) \) and \( (G, R') \). Since the algorithm makes the same matches up to time \( t_0 - 1 \), the set of unmatched vertices in \( A \) at time \( t_0 \) is the same for both instances. We can thus conclude that \( N_{R',t_0}(b_{t_0}) \subset N_{R,t_0}(b_{t_0}) \). Since the algorithm makes a different decision at time \( t_0 \), we have \( M^R(b_{t_0}) \in N_{R,t_0}(b_{t_0}) \setminus N_{R',t_0}(b_{t_0}) \). By definition of \( N_{R',t_0}(b_{t_0}) \), this means one of the vertices in \( R' \) is unmatched by \( M^R \). Since \( M^R \) is guaranteed to match all vertices in \( R \), this implies that \( b \) is unmatched in \( M^R \).

Note that \( M^R \triangle M^{R'} \) is a disjoint union of cycles and paths. Since \( b \) is matched under \( M^{R'} \) but not matched under \( M^R \), the connected component of \( M^R \triangle M^{R'} \) containing \( b \) is a path which we denote \( P_b := (V_b, E_b) \). Furthermore, \( b \) is one of the endpoints of the path.
Next we show that $M^R \triangle E_b$ is a matching in $G$ that matches every vertex in $R'$. First, since $b$ is matched under $M^R$ and is unmatched under $M^R$, we can conclude that $b$ is matched under $M^R \triangle E_b$. To ensure that all vertices in $R$ are still matched under $M^R \triangle E_b$, there are two possibilities to consider. If $P_b$ has an odd number of edges, then every vertex matched in $M^R$ is also matched in $M^R \triangle E_b$. If $P_b$ has an even number of edges, then the last vertex in $P_b$ is some $b' \in B$, and thus every online vertex matched under $M^R$ with the exception of $b'$ is also matched under $M^R \triangle E_b$. We conclude by observing that $b' \notin R$, since every vertex in $R \cap V_p$ has degree 2 in $M^R \triangle M^{R'}$ (one edge for each matching $M^R$ and $M^{R'}$) and therefore cannot be the endpoint of a path. Since in both cases all vertices in $R \cup \{b\}$ are matched under $M^R \triangle E_b$, we see that $M^R \triangle E_b$ is a matching that matches all vertices in $R'$.

We are now ready to show $A_R \subset A_{R'}$, there are three cases to consider.

Case 1: ($a \in A_R \cap V_b$) Since $b$ is one of the endpoints of $P_b$ and is matched by $M^{R'}$ but not by $M^R$, it is easy to check that every $a \in A_R \cap V_b$ is also matched under $M^{R'}$.

Case 2: ($a \in A_R \setminus V_b$ and $a$ is matched before time $t_0$) Since the algorithm behaves the same for both instances before time $t_0$, $a$ will have the same partner under $M^R$ as it does in $M^R$. In particular, $a$ is also matched in $M^{R'}$.

Case 3: ($a \in A_R \setminus V_b$ and $a$ is matched at time $\geq t_0$) We will show by induction that vertices in the set $\{a \in A_R \setminus V_b : (a, b_{\tau}) \in M^R, \tau > t_0\}$ have the same partners in $M^{R'}$ as they do in $M^R$.

Induction Step: For $t' > t_0$, $b_{\tau} \notin V_b$, suppose that for every $b_{\tau} \notin V_b$, $t_0 < \tau < t'$, we have $M^R(b_{\tau}) = M^{R'}(b_{\tau})$. This means that the current matching is still a subset of $M^R \triangle E_b$. Therefore $M^R(b_{\tau}) \in N_{R', \tau'}(b_{\tau})$. $b_{\tau}$ fails to match with $M^R(b_{\tau})$ only if it is instead matched to a higher ranked alternative $a'$ that is available. This happens only if $M^R(a') \neq M^{R'}(a')$. There are two possibilities to consider:

1. The first possibility is if $M^R(a')$ arrives before $b_{\tau}$. Since $M^R(a') \neq M^{R'}(a')$ we can conclude that $M^R(a')$ must be in $P_b$, since by the induction hypothesis every vertex in cases 2 and 3 that arrives before time $t'$ has the same partner under $M^R$ and $M^{R'}$. However this would imply that $b_{\tau} \in P_b$ which is a contradiction.

2. The second possibility is if $M^R(a')$ arrives after $b_{\tau}$. This would mean $a'$ was unmatched upon the arrival of $b_{\tau}$ in the instance $(G, R)$. We will show that $(a', b_{\tau}) \in N_{R, \tau}(b_{\tau})$. By the induction hypothesis, the algorithm assigned the same matches to all vertices in $B \setminus V_p$ that
Since both possibilities lead to contradiction, no such \( a' = M^R(b_v) \), and \( b_v \notin V_p \), we have \( (a', b_v) \notin E_b \). From this we have

\[
\{(a', b_v)\} \cup (M^R_{t-1} \cap (E \setminus E_b)) = \{(a', b_v)\} \cup (M^R_{t-1} \cap (E \setminus E_b)) \\
= (M^R_{t-1} \cup \{(a', b_v)\}) \cap (E \setminus E_b) \subset M^R \cap (E \setminus E_b)
\]

We also have \( M^R_{t-1} \cap E_b \subset M^R \cap E_b \). Since \( P_b \) is a connected component of \( M^R \setminus M^R' \), there are no edges in \( M^R \cup M^R' \) that have exactly one endpoint in \( V_p \). This implies that \( \tilde{M} := \left( M^R \cap (E \setminus E_b) \right) \cup (M^R \cap E_b) \) is a matching in \( G \). \( \tilde{M} \) contains \( \{(a', b_v)\} \cup (M^R_{t-1} \cap (E \setminus E_b)) \) and \( M^R \cap E_b \), so it contains their union, which is \( M^R_{t-1} \cup \{(a', b_v)\} \).

Since \( M^R \) matches all vertices in \( R \), \( M^R' \cap (E \setminus E_b) \) matches all vertices in \( R \setminus V_p \). Since \( M^R \) matches all vertices in \( R \), \( M^R \cap E_b \) matches all vertices in \( R \cap V_p \). From this we conclude that all vertices in \( R \) are matched under \( \tilde{M} \).

To summarize, there exists a matching \( \tilde{M} \) which contains \( M^R_{t-1} \cup \{(a', b_v)\} \) and matches all vertices in \( R \). This means \( a' \in N_{R,t'}(b_v) \). This leads to a contradiction since \( M^R(b_v), a' \in N_{R,t'}(b_v) \) implies that \( M^R(b_v) \) has a higher rank than \( a' \), which contradicts the assumption above that \( a' \) has the higher rank.

Since both possibilities lead to contradiction, no such \( a' \) can exist, and therefore it must be the case that \( M^R(b_v) = M^R(b_v) \). This completes the induction proof.

Therefore \( A_R \subset A_{R'} \).

### D.3 Proof of Theorem 8

In this section we prove that the competitive ratio of \textbf{Reservation-Priority} is at most \( \frac{1}{2-p} \). We will do this by constructing instances for which \textbf{Reservation-Priority} obtains a matching with expected size no more than \( \frac{1}{2-p} \) times the size of the optimal matching.

#### D.3.1 Constructing a Worst Case Instance for Reservation-Priority

Let \( p \in [0, 1] \). For any \( m \in \mathbb{N} \), let \( n = \left( 2 + \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^2 m}} \right) m \) and consider the following bipartite graph \( G = (A, B, E) \) (shown in Fig. D.1 with offline vertices \( A = \{a_1, ..., a_n\} \) and online vertices \( B = \{b_1, ..., b_n\} \). We further subdivide both sides of the graph as follows:

\[
A = A_1 \cup A_2 \cup A_3 \text{ where} \\
A_1 = \{a_1, ..., a_m\}
\]
\[ A_2 = \{a_{m+1}, \ldots, a_{2m}\} \]
\[ A_3 = \{a_{2m+1}, \ldots, a_n\} \]

Similarly,
\[ B = B_1 \cup B_2 \cup B_3 \]
\[ B_1 = \{b_1, \ldots, b_m\} \]
\[ B_2 = \{b_{m+1}, \ldots, b_{2m}\} \]
\[ B_3 = \{b_{2m+1}, \ldots, b_n\} \]

Let \( \pi \) be the ranking used by Reservation-Priority. The ordering of \( A \) and \( B \) is chosen so that vertices in \( A_2 \) have higher rank than vertices in \( A_1 \), i.e., \( \pi(a') > \pi(a) \) for every \( a' \in A_2 \) and every \( a \in A_1 \).

The edge set \( E \) has three components:

- A perfect matching between \( A \) and \( B \) of the form \( \{(a_i, b_i)\}_{i=1}^n \).
- All possible edges between \( A_2 \) and \( B_1 \). Concretely, this is \( \{(a_i, b_j) : m < i \leq 2m, 1 \leq j \leq m\} \).
- All possible edges between \( A_3 \) and \( B_2 \). Concretely, this is \( \{(a_i, b_j) : 2m < i \leq n, m < j \leq 2m\} \).

**D.3.2 Performance of Reservation-Priority on \( G \)**

We begin with the following observation:

**Observation 8.** If \( |R \cap (B_2 \cup B_3)| \leq |A_3| \), then for any \( t \leq m \), \( N_{R,t}(b_t) \) will contain all vertices in \( A_2 \) that have yet to be matched.

**Proof.** To see why this is true, note that:

- Any \( b_{t'} \in R \cap (B_1 \cup B_3) \) that has not yet been matched (i.e., \( t' > t \)) can be matched to \( a_{t'} \).
  We know \( a_{t'} \) is unmatched because its only neighbor is \( b_{t'} \) which has not yet been matched.

- Since \( |R \cap (B_2 \cup B_3)| \leq |A_3| \), the number of vertices in \( A_3 \) that are not matched to a vertex in \( B_3 \) is larger than \( |R \cap B_2| \). Since \( B_2 \) is fully connected to \( A_3 \), all vertices in \( R \cap B_2 \) can be matched to vertices in \( A_3 \).

Since we have constructed a matching which matches all vertices in \( R \) to \( A_1 \cup A_3 \), all unmatched vertices in \( A_2 \) are in \( N_{R,t}(b_t) \).

Using Observation 8 and the fact that the vertices in \( A_2 \) have higher ranks than those in \( A_1 \), we conclude that Reservation-Priority will match \( B_1 \) to \( A_2 \) if \( |R \cap (B_2 \cup B_3)| \leq |A_3| \).
Figure D.1: A visualization of the graph $G$. $A_1, A_2, B_1, B_2$ each have $m$ vertices while $A_3, B_3$ each have \( \left( \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^4 m}} \right) m \) vertices. The light gray area represents the perfect matching between $A$ and $B$. The dark gray areas represent the fully connected subgraphs between $(A_2,B_1)$ and $(A_3,B_2)$. 
Since $N(A_1) = B_1$, all vertices in $A_1$ will go unmatched, and hence the matching produced by Reservation-Priority will have at most $|A_2| + |A_3|$ vertices. Letting $M$ be the matching produced by Reservation-Priority and defining the random variable $S = 1[|R \cap (B_2 \cup B_3)| \leq |A_3|]$, we then have:

$$\mathbb{E}[|M|] = \mathbb{E}[|M| | S = 1] \mathbb{P}[S = 1] + \mathbb{E}[|M| | S = 0] \mathbb{P}[S = 0]$$

$$\leq (|A_2| + |A_3|)\mathbb{P}[S = 1] + n\mathbb{P}[S = 0]$$

$$= \left(1 + \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}} \right) m \mathbb{P}[S = 1] + \left(2 + \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}} \right) m \mathbb{P}[S = 0]$$

$$= \left(1 + \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}} \right) m + m \mathbb{P}[S = 0]$$

We can bound $\mathbb{P}[S = 0]$ by observing that

$$|R \cap (B_2 \cup B_3)| \sim \text{Bin}(|B_2| + |B_3|, p)$$

and using the following Chernoff bound:

For $Y \sim \text{Bin}(N, p)$, $\mathbb{P}[Y > Np(1 + \epsilon)] \leq \exp \left(-Np \frac{\epsilon^2}{2(1 + \epsilon)} \right)$.

Applying the Chernoff bound to our setting, we see that

$$\mathbb{P}[S = 0] = \mathbb{P}[|R \cap (B_2 \cup B_3)| > |A_3|]$$

$$= \mathbb{P} \left[ |R \cap (B_2 \cup B_3)| > \left(\frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}} \right) m \right]$$

$$\leq \mathbb{P} \left[ |R \cap (B_2 \cup B_3)| > \left(\frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}} \right) m \right]$$

$$= \mathbb{P} \left[ |R \cap (B_2 \cup B_3)| > \left(\frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}} \right) m \left(1 + \frac{(1-p)\sqrt{\frac{3 \log m}{(1-p)^3 m}}}{\frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}}} \right) \right]$$

$$\leq \exp \left(-\frac{1}{3} \left(\frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}} \right) m \left(1 + \frac{(1-p)\sqrt{\frac{3 \log m}{(1-p)^3 m}}}{\frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}}} \right)^2 \right)$$

$$= \exp \left(-\frac{1}{3} \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}} m \left(1 + \frac{(1-p)\sqrt{\frac{3 \log m}{(1-p)^3 m}}}{\frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}}} \right)^2 \right)$$
≤ \exp\left(-\frac{1}{3} \frac{1}{1-p} + 1\right) m \left(1 - p\right) \left(\frac{3 \log m}{(1-p)^3 m}\right)^2
= \exp\left(-\frac{1}{3} (1 - p)m \left(1 - p\right) \left(\frac{3 \log m}{(1-p)^3 m}\right)^2\right)
= \frac{1}{m}.

Applying this bound, we see that

\[ \frac{E[|M|]}{|M^*|} \leq \frac{\left(1 + \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}}\right) m + mP[S = 0]}{n} \]
\[ = \frac{\left(1 + \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}}\right) m + 1}{\left(2 + \frac{p}{1-p} + \sqrt{\frac{3 \log m}{(1-p)^3 m}}\right) m} \]
\[ = \frac{\left(1 + (1-p) \sqrt{\frac{3 \log m}{(1-p)^3 m}}\right) m + p}{\left(2 - p + (1-p) \sqrt{\frac{3 \log m}{(1-p)^3 m}}\right) m} \]
\[ = \frac{1 + O\left(\frac{\log m}{m}\right)}{2 - p + O\left(\frac{\log m}{m}\right)}. \]

Since this analysis holds for all \( m \in \mathbb{N} \), send \( m \to \infty \) to obtain \( \frac{E[|M|]}{|M^*|} \leq \frac{1}{2^p} \), which is the desired result.
Appendix E

Proofs for Section 6

E.1 Proof of Theorem 9

We prove Theorem 9 by proving the following Lemma:

Lemma 14. For any \( s \in \mathbb{R}_+ \), the following inequality

\[
\frac{|\tau_e(s+Z) - \tau_e(s)|}{\tau_e(s)} \leq \delta
\]

is satisfied with probability at least \( 1 - p \) when \( Z \sim \mathcal{L}_e \).

The Theorem follows by applying the result with \( s = s_e(k \Delta t) \). We will prove Lemma 14 by considering two exhaustive cases:

Case 1: \( s < \log \frac{1}{\epsilon \delta} \),

Case 2: \( s \geq \log \frac{1}{\epsilon \delta} \).

E.1.1 Proving Lemma 14 in Case 1

In Case 1 we have \( s < \frac{\log \frac{1}{\epsilon \delta}}{\epsilon \delta} \). By the condition in Theorem 9 \( F_e(c_{e, \delta}) \geq \frac{1}{\epsilon} \left( \frac{1}{\delta} + 1 \right) \log \frac{1}{p} \), and thus \( s \leq F_e(c_{e, \delta}) \).

Next, note that

\[
P[s + Z \geq F_e(c_{e, \delta})] = P[Z \geq F_e(c_{e, \delta}) - s]
\]
\[
\leq P \left[ Z \geq \left( \frac{1}{\epsilon} \left( \frac{1}{\delta} + 1 \right) \log \frac{1}{p} \right) - \frac{\log \frac{1}{p}}{\epsilon \delta} \right]
\]
\[
= P \left[ Z \geq \frac{\log \frac{1}{p}}{\epsilon} \right]
\]
\[(a) \leq \frac{1}{2} \exp \left( -\epsilon \frac{\log \frac{1}{p}}{\epsilon} \right) = \frac{p}{2}, \]

where \((a)\) is due to the formula for the cumulative distribution function for the Laplace distribution. Therefore, with probability at least \(1 - \frac{p}{2}\), both \(s, s + Z\) are less than \(F_e(c_{e,\delta})\). For the remainder of the Case 1 discussion we will condition on the high probability event that both \(s, s + Z\) are less than \(F_e(c_{e,\delta})\).

By Assumption 2 we know that \(f_e\) is non-decreasing, which means that \(F_e\) is non-decreasing and invertible. From this we can conclude that \(F_e^{-1}(s) \leq c_{e,\delta}\) and \(F_e^{-1}(s + Z) \leq c_{e,\delta}\). Since \(f_e\) is non-decreasing this means

\[f_e(0) \leq f_e(s), f_e(s + Z) \leq f_e(c_{e,\delta}).\]

From this we can deduce that

\[|f_e(s + Z) - f_e(s)| \leq f_e(c_{e,\delta}) - f_e(0) = (1 + \delta)f_e(0) - f_e(0) = \delta f_e(0) \leq \delta f_e(s),\]

which establishes Lemma 14 in Case 1.

### E.1.2 Proving Lemma 14 in Case 2

For Case 2 we have \(s \geq \frac{\log \frac{1}{c_{e,\delta}}}{\epsilon^2}\). Our analysis for this case will involve \(\frac{d}{dy} F_e^{-1}(y)\) and \(\frac{d}{dy} \tau_e(y)\), so we will compute them using chain rule:

\[1 = \frac{d}{dy} y = \frac{d}{dy} F_e(F_e^{-1}(y)) = F_e'(F_e^{-1}(y)) \left( \frac{d}{dy} F_e^{-1}(y) \right) \]

\[\implies \frac{d}{dy} F_e^{-1}(y) = \frac{1}{F_e'(F_e^{-1}(y))}.\]
Using this, we can now compute \( \frac{d}{dy} \tau_e(y) \) using chain rule:

\[
\frac{d}{dy} \tau_e(y) = \frac{d}{dy} f_e(F_e^{-1}(y)) = f'_e(F_e^{-1}(y)) \left( \frac{d}{dy} F_e^{-1}(y) \right) = \frac{f'_e(F_e^{-1}(y))}{F'_e(F_e^{-1}(y))}
\]

Defining \( x_y := F_e^{-1}(y) \), we see that

\[
\frac{d}{dy} \tau_e(y) = f'_e(x_y) = \frac{f'_e(x_y)}{x_y} + f_e(x_y).
\]

We make an observation that will be useful later:

**Observation 9.** Since \( f_e \) is non-negative under Assumption 2, we have \( \frac{d}{dy} \tau_e(y) \leq \frac{1}{x_y} \).

**Observation 10.** \( t_e(y) = \frac{y}{x_y} \). This is because since \( t_e(y) = f_e(F_e^{-1}(y)) \), we have \( F_e^{-1}(y)t_e(y) = F_e^{-1}(y)f_e(F_e^{-1}(y)) = F_e(F_e^{-1}(y)) = y \). Therefore \( t_e(y) = \frac{y}{F_e^{-1}(y)} = \frac{y}{x_y} \).

With this setup in hand, we are now ready to prove the Lemma. First note that

\[
\mathbb{P}[|Z| \geq \delta s] \overset{(a)}{=} \exp(-\epsilon \delta s) \\
\leq \exp(-\epsilon \delta \log \frac{1}{p}) = p,
\]

where \( (a) \) is due to the fact that \( |Z| \) has the exponential distribution with parameter \( \frac{1}{\epsilon} \), and this distribution has the cumulative distribution function \( \mathbb{P}[|Z| \geq t] = e^{-\epsilon t} \mathbb{1}[t \geq 0] \). Thus with probability at least \( 1 - p \), we have \( |Z| \leq \delta s \). For the remainder of the Case 2 discussion we will condition on the high probability event that \( |Z| \leq \delta s \).

By the fundamental theorem of calculus,

\[
|\tau_e(s + Z) - \tau_e(s)| = \left| \int_s^{s+Z} \left( \frac{d}{dy} \tau_e(y) \right) dy \right| \\
\leq \int_s^{s+Z} \left| \frac{d}{dy} \tau_e(y) \right| dy \\
\overset{(a)}{\leq} \int_s^{s+Z} \frac{1}{|x_y|} dy \\
\overset{(b)}{\leq} \int_s^{s+Z} \frac{1}{x_{\min(s,s+Z)}} dy
\]
\[(c) \leq \int_s^{s+Z} \frac{1}{x(1-\delta)s} \, dy = \frac{|Z|}{x(1-\delta)s} \leq \frac{\delta s}{x(1-\delta)s} = \frac{\delta}{1-\delta} \frac{(1-\delta)s}{x(1-\delta)s} \]
\[\leq \frac{\delta}{1-\delta} \tau_e((1-\delta)s) \leq \frac{\delta}{1-\delta} \tau_e(s)\]

where (a) is due to Observation 9. Since \(x_y\) was defined to be \(F_{e^{-1}}(y)\), (b) is due to the fact that \(F_e\) is increasing, therefore \(x_y\) is an increasing function of \(y\) and hence \(\frac{1}{x_y}\) is a decreasing function of \(y\). (c) is because \(\min(s, s+Z) \geq (1-\delta)s\) since we are in the event that \(|Z| \leq \delta s\). (d) is due to Observation 10 and (e) is because \(t_e\) is an increasing function, since it is \(f_e\) composed with \(F_{e^{-1}}\), which are both increasing. Since \(\frac{\delta}{1-\delta} = \delta + \frac{\delta^2}{1-\delta}\), we have

\[|\tau_e(s+Z) - \tau_e(s)| \leq (\delta + O(\delta^2)) \tau_e(s)\]

which proves Lemma 14 in Case 2.
Appendix F

Proofs for Section 2

F.1 Proof of Lemma 1

Given \( f, g : X \rightarrow \mathbb{R} \) define \( x_f := \arg \min_{x \in X} f(x) \) and \( x_g := \arg \min_{x \in X} g(x) \). Let \( ||f - g||_{\infty} := \sup_{x \in X} |f(x) - g(x)| \). We then see that

\[
f(x_g) \leq g(x_g) + ||f - g||_{\infty} \\
    \leq g(x_f) + ||f - g||_{\infty} \\
    \leq [f(x_f) + ||f - g||_{\infty}] + ||f - g||_{\infty} \\
    = f(x_f) + 2||f - g||_{\infty},
\]

which is the desired result.

F.2 Proof of Lemma 2

First recall the definition of the \( \chi^2 \)-divergence \( \chi^2(P||Q) \) between two probability distributions \( P, Q \).

\[
\chi^2(P||Q) := \mathbb{E}_Q \left[ \left( 1 - \frac{dP}{dQ} \right)^2 \right] \quad \text{if } P \ll Q, +\infty \text{ else.}
\]

The function \( \chi^2(P||Q) \) is non-negative and is zero if and only if \( P = Q \). Since \( (\cdot)_+ \) is 1-Lipschitz, we have:

\[
F(x, w) - \mathbb{E}_{\hat{P}} \hat{F}_K(x, w)
\]
APPENDIX F. PROOFS FOR SECTION 2

\[ = \sum_{ijt} \sum_{\lambda \in \mathbb{N}} (\bar{\lambda} + w_{ijt} - x_{ijt})_+ (P_{ijt}(\bar{\lambda}) - \hat{P}_{ijt}(\bar{\lambda})) \]

Define \( \ell_{ijt}(\bar{\lambda}) := (\bar{\lambda} + w_{ijt} - x_{ijt})_+ - \mathbb{E}_{P_{ijt}}(\bar{\lambda} + w_{ijt} - x_{ijt})_+ \), we have:

\[ F(x, w) - \mathbb{E}_{P_{ijt}} \mathbb{E}_K(x, w) = \sum_{ijt} \sum_{\lambda \in \mathbb{N}} \ell_{ijt}(\bar{\lambda})(P_{ijt}(\bar{\lambda}) - \hat{P}_{ijt}(\bar{\lambda})) \]

Since \( \sum_{\lambda \in \mathbb{N}} C(P_{ijt}(\bar{\lambda}) - \hat{P}_{ijt}(\bar{\lambda})) = 0 \) for any constant \( C \), we let \( C = \mathbb{E}_{P_{ijt}}(\bar{\lambda} + w_{ijt} - x_{ijt})_+ \).

\[ \sum_{ijt} \sum_{\lambda \in \mathbb{N}} \ell_{ijt}(\bar{\lambda})(P_{ijt}(\bar{\lambda}) - \hat{P}_{ijt}(\bar{\lambda})) \]

\[ = \sum_{ijt} \sum_{\lambda \in \mathbb{N}} \ell_{ijt}(\bar{\lambda}) \sqrt{P(\bar{\lambda})_{ijt}} \left( \sqrt{P(\bar{\lambda})_{ijt}} - \frac{\hat{P}_{ijt}(\bar{\lambda})}{\sqrt{\hat{P}_{ijt}(\bar{\lambda})}} \right) \]
\[ \leq \sum_{ijt} \ell_{ijt}(\bar{\lambda})^2 P_{ijt}(\bar{\lambda}) \sum_{\lambda \in \mathbb{N}} \left( 1 - \frac{\hat{P}_{ijt}(\bar{\lambda})}{\hat{P}_{ijt}(\bar{\lambda})} \right)^2 \]
\[ = \sqrt{\text{Var}_P(||\bar{\lambda} + w - x||_2)} \sqrt{\sum_{ijt} \chi^2(\hat{P}_{ijt}||P_{ijt})} \]
\[ \leq \sqrt{\text{Var}_P(||\bar{\lambda}||_2)} \sqrt{||\chi(\hat{P}||P)||_2} \]

The first inequality is due to Cauchy-Schwarz in \( L_2^P \), and the second inequality is due to the fact that \( \text{Var}(X_+) \leq \text{Var}(X) \) for any random variable \( X \) by the following calculations:

\[ \text{Var}(X_+) := \mathbb{E}[(X_+ - \mathbb{E}[X_+])^2] \]
\[ \leq \mathbb{E}[(X_+ - \mathbb{E}[X_+])^2] + (\mathbb{E}[X_+] - \mathbb{E}[X])^2 \]
\[ = \mathbb{E}[(X_+ - \mathbb{E}[X_+])^2] \]
\[ \leq \mathbb{E}[(X - \mathbb{E}[X])^2] \]
\[ = \text{Var}(X) \]

The second inequality is because \((\cdot)_+ \) is a 1-Lipschitz function. It is also possible to control the model error using the RMSE of the generative model \( \hat{P} \), but that bound is weaker than what is presented here. For the standard deviation bound, we use the concentration of measure for sub-exponential random variables. A random variable \( X \) is sub-exponential if there exists parameters \( \sigma^2, b \) so that for any \( |\lambda| \leq b^{-1} \), we have

\[ \text{Var}(X) \leq \mathbb{E}[(X - \mathbb{E}[X])^2] \]

The second inequality is because \((\cdot)_+ \) is a 1-Lipschitz function. It is also possible to control the model error using the RMSE of the generative model \( \hat{P} \), but that bound is weaker than what is presented here. For the standard deviation bound, we use the concentration of measure for sub-exponential random variables. A random variable \( X \) is sub-exponential if there exists parameters \( \sigma^2, b \) so that for any \( |\lambda| \leq b^{-1} \), we have

\[ \text{Var}(X) \leq \mathbb{E}[(X - \mathbb{E}[X])^2] \]
\[
\log \mathbb{E}[e^{\lambda (X - \mathbb{E}X)}] \leq \frac{\lambda^2 \sigma^2}{2}
\]

The following probability bounds for sub-exponential random variables are well known: If \(X\) is \((\sigma^2, b)\) sub-exponential, then:

\[
\mathbb{P}[|X - \mathbb{E}X| > t] \leq \exp \left( -\frac{t^2}{2\sigma^2} \right) \quad \text{if } t \leq \frac{\sigma^2}{b}
\]
\[
\mathbb{P}[|X - \mathbb{E}X| > t] \leq \exp \left( -\frac{t}{2b} \right) \quad \text{otherwise}
\]

Let \(\{\lambda^1, \ldots, \lambda^K\}\) be samples from \(\hat{\lambda}\) used to form the objective function. Let \(\hat{F}_{x,w}(\lambda) := \sum_{ijt}(\lambda_{ijt} + w_{ijt} - x_{ijt})_+\). 1-Lipschitz functions of sub-exponential random variables are also sub-exponential with the same parameters, thus \(\{\hat{F}_{x,w}(\lambda^k)\}_{k=1}^K\) are i.i.d. \((\sigma^2, b)\)-sub-exponential random variables. Thus, the objective of (2.4), \(\frac{1}{K} \sum_{k=1}^K \hat{F}_{x,w}(\lambda^k)\) is \((\frac{\sigma^2}{K}, \frac{b}{K})\) sub-exponential. Applying the first part of the probability bound, we see that:

\[
\mathbb{P} \left( \left| \frac{1}{K} \sum_{k=1}^K \hat{F}_{x,w}(\lambda^k) - \mathbb{E}\hat{F}_{x,w}(\lambda) \right| > t \right) \leq \exp \left( -\frac{Kt^2}{2\sigma^2} \right)
\]

for any \(t < \frac{\sigma^2}{b/K} = \frac{\sigma^2}{b}\). For any \(\delta > 0\) error tolerance, setting \(t = \frac{2\sigma}{\sqrt{K}} \sqrt{n^2T \log(m) + \log(\delta^{-1/2})}\), for sufficiently large \(K\) the bound evaluates to \(\delta(m)^{-n^2T}\). However this inequality only applies to a particular pair of \(x, w\). Since \(x \in \mathbb{R}^{n^2T}\) and \(||x||_\infty \leq m\), \(x\) can take at most \(|m|^{n^2T}\) many distinct values. Note that if \(w_{ijt} > m\) we can always set \(w_{ijt} = m\) without affecting performance because the system cannot pick up more than \(m\) waiting customers at any time. Thus, we also have \(||w||_\infty \leq m\) and hence \(w\) can take at most \(m^{n^2T}\) values. Thus there are at most \(m^{2n^2T}\) possible plans \((x, w)\). Taking a union bound over all possible \(x, w\) gives, with probability at least \(1 - \delta\),

\[
\left\| \frac{1}{K} \sum_{k=1}^K \hat{F}_{x,w}(\lambda^k) - \mathbb{E}\hat{F}_{x,w}(\lambda) \right\|_\infty \leq \frac{2\sigma}{\sqrt{K}} \sqrt{n^2T \log(m) + \log(\delta^{-1/2})}
\]

Applying lemma with this bound yields the desired result.
F.3 Proof of Lemma 3

Here we use sub-exponential concentration inequalities to obtain a bound on the maximum and minima of i.i.d. sub-exponential random variables. Lemma 3 is related to the distribution of maxima of sub-exponential random variables. Let \( X_1, \ldots, X_n \) be i.i.d. zero mean \((\sigma^2, b)\) sub-exponential random variables. We proceed by the standard Chernoff bounding technique. For any \( 0 < \lambda \leq b^{-1} \), we have:

\[
P \left[ \max_{1 \leq i \leq n} X_i \geq t \right] = P \left[ e^{\lambda \max_{1 \leq i \leq n} X_i} \geq e^{\lambda t} \right] \\
= P \left[ \max_{1 \leq i \leq n} e^{\lambda X_i} \geq e^{\lambda t} \right] \quad \text{Markov Inequality} \\
\leq e^{-\lambda t} E \left[ \max_{1 \leq i \leq n} e^{\lambda X_i} \right] \\
\leq e^{-\lambda t} E \left[ \sum_{1 \leq i \leq n} e^{\lambda X_i} \right] \\
\leq n \exp \left( -\lambda t \right) \exp \left( \frac{\lambda^2 \sigma^2}{2} \right) \\
= \exp \left( -\lambda t + \frac{\lambda^2 \sigma^2}{2} + \log n \right)
\]

If \( t > \frac{\sigma^2}{b} \) then setting \( \lambda = \frac{t}{b} \) gives the tightest upper bound, in which case we have:

\[
P \left[ \max_{1 \leq i \leq n} X_i \leq t \right] \leq \exp \left( -\lambda t + \frac{\sigma^2}{2b^2} + \log n \right)
\]

but recall that \( t > \frac{\sigma^2}{b} \implies \frac{\sigma^2}{2b^2} \leq \frac{t}{2b} \), meaning

\[
P \left[ \max_{1 \leq i \leq n} X_i \leq t \right] \leq \exp \left( -\frac{t}{2b} + \log n \right)
\]

thus for any \( \delta > 0 \), setting \( t = 2b \log \frac{n}{\delta} \), the upper bound is equal to \( \delta \). Hence,

\[
P \left[ \max_{1 \leq i \leq n} X_i \geq 2b \log \frac{n}{\delta} \right] \leq \delta
\]

The concentration of the minimum is analogous, by noting that \(-X\) is also sub-exponential, and applying the above argument. Applying this to our problem, if the demand \( \lambda_{ijt} \) for each \((i, j, t)\)
APPENDIX F. PROOFS FOR SECTION 2

143

is \((\sigma^2_{ijt}, b)\) sub-exponential, and \(K\) samples \(\lambda^1_{ijt}, ..., \lambda^K_{ijt}\) are observed, then by the above argument, with probability at least \(1 - \frac{\delta}{n^2T}\) all samples fall in the interval

\[
\left[ \mathbb{E}[\lambda_{ijt}] - 2b \log \frac{Kn^2T}{\delta}, \mathbb{E}[\lambda_{ijt}] + 2b \log \frac{Kn^2T}{\delta} \right].
\]

But since these samples are integer valued, if they lie in an interval of size \(O(\log K)\), then there can be at most \(O(\log K)\) distinct samples of the demand. Taking a union bound over all tuples \((i, j, t)\), we have that with probability at least \(1 - \delta\), for each tuple \((i, j, t)\), the number of unique elements in \(\{\lambda^k_{ijt}\}_{k=1}^K\) is at most \(4b \log Kn^2T\). Summing over all \((i, j, t)\) we then have that the total number of decision variables is at most \(4bn^2T \log Kn^2T\). The number of decision variables is trivially at most \(Kn^2T\), therefore taking the better of the two bounds yields the result.

F.4 Proof of Lemma 4

Define \(u = [u^1, ..., u^K] \in \mathbb{R}^{n^2TK}\) so that the decision variable for (2.7) is \(z = [x, u]\). To show that (2.7) is totally unimodular, it is necessary and sufficient to show that all extreme points of the constraint polyhedron are integer vectors. Recall that a point \(z^\ast = [x^\ast, u^\ast]\) is an extreme point if and only if the matrix of active constraints \(B(z^\ast)\) has rank \(n^2T(K + 1)\), where the active constraint matrix is the matrix whose rows are the equality constraints and active inequality constraints of the problem at \(z^\ast\). Since \(z \in \mathbb{R}^{n^2T(K+1)}\), a point \(z^\ast\) is extreme if and only if \(B(z^\ast)\) has full column rank. For notational convenience, we write \(x = [x_+ x_0]\) where \(x_0\) are the entries of \(x\) for which the constraint \(x \geq 0\) is active. Similarly, we write \(u = [u_+ u_0]\) where again \(u_0\) denotes the entries of \(u\) for which the constraint \(u \geq 0\) is active. We can express the active constraints as:

\[
\begin{bmatrix}
A_+ & A_0 & 0 & 0 \\
0 & I & 0 & 0 \\
B_+ & B_0 & I & 0 \\
0 & 0 & 0 & I
\end{bmatrix}
\begin{bmatrix}
x_+^* \\
x_0^* \\
u_+^* \\
u_0^*
\end{bmatrix}
= \begin{bmatrix}
b \\
0 \\
0 \\
\lambda
\end{bmatrix}
\]

where \(A_+, A_0, b\) are chosen so that \(A_+x_+^* + A_0x_0^* = b\) is equivalent to the network flow constraints specified by (2.1). Network flow problems are known to be totally unimodular, so the matrix \(A := [A_+ A]\) is a totally unimodular matrix. \(B_+, B_0\) are chosen so that \(B_+x_+^* + B_0x_0^* = \lambda\) represents the active inequality constraints \(u_{ijt}^k = \lambda_{ijt}^k - x_{ijt}\). Note that the column space of \(B(z^\ast)\) is the same
as the column space of the following matrix

\[
C(z^*) := \begin{bmatrix}
A_+ & A_0 & 0 & 0 \\
0 & I & 0 & 0 \\
0 & 0 & I & 0 \\
0 & 0 & 0 & I 
\end{bmatrix}
\]

We have \(A_+x_+^* + A_0x_0^* = b\) and since \(x_0^* = 0\), we have \(A_+x_+^* = b\). \(z^*\) is an extreme point of the constraint set if and only if \(B(z^*)\) has full column rank. \(B(z^*)\) has full column rank only if \(C(z^*)\) has full column rank. It is clear from its definition that \(C(z^*)\) has full column rank if and only if \(A_+\) has full column rank. Because \(A_+\) is a submatrix of a totally unimodular matrix, its left inverse exists and is an integer matrix. Therefore, \(x_+ = A_+^{-1}b\). Since \(A_+^{-1}, b\) both have integer entries, this proves that \(x_+^*\) is integer, meaning \(x^* = [x_+^*, x_0^*]\) is integer. For each tuple \((i, j, t, k)\), the decision variable \(u_{ijt}^k\) is subject to exactly two constraints: \(u_{ijt}^k \geq \lambda_{ijt}^k - x_{ijt}\), and \(u_{ijt}^k \geq 0\). In any extreme point, at least one of these constraints is active. This can be shown easily via contradiction. If \(z^*\) is extreme and for some \(u_{ijt}^k\), both (i.e. all) constraints are inactive, then define \(I\) to be the index of \(u_{ijt}^k\) in \(z^*\). Since there are no active constraints involving \(u_{ijt}^k\), the \(I\)th column of \(B(z^*)\) is zero, hence \(B(z^*)\) cannot have full column rank. Therefore \(u_{ijt}^k \in \{0, \lambda_{ijt}^k - x_{ijt}\}\). Since we showed \(x^*\) is integer, and \(\{\lambda_{ijt}^k\}_{k=1}^K\) are integer, this implies that \(u^*\) must be integer, finally implying that \(z^*\) is integer. Since all extreme points are integer valued, (2.7) is a totally unimodular linear program.
Bibliography


[35] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A
completeness theorem for protocols with honest majority. In STOC 1987, New York, New

[36] Yanmin Gong, Chi Zhang, Yuguang Fang, and Jinyuan Sun. Protecting location privacy for
task allocation in ad hoc mobile cloud computing. IEEE Transactions on Emerging Topics in

[37] Mehmet Gursoy, Ling Liu, Stacey Truex, and Lei Yu. Differentially private and utility pre-
serving publication of trajectory data. IEEE Transactions on Mobile Computing, 18(10):2315–
2329, 2018.

[38] Lino Guzzella and Antonio Sciarretta. Vehicle Propulsion Systems. Springer Berlin Heidelberg,
2007.

[39] Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set

[40] Brian He and Joseph Chow. Optimal privacy control for transport network data sharing.

[41] Sebastian Hörl, Claudio Ruch, Felix Becker, Emilio Frazzoli, and Kay Axhausen. Fleet control
algorithms for automated mobility: A simulation assessment for zurich. In Annual Meeting of
the Transportation Research Board, 2018.

[42] Sebastian Hörl, Claudio Ruch, and Emilio Frazzoli. Amodeus, a simulation-based testbed for
autonomous mobility-on-demand systems. In Proc. IEEE Int. Conf. on Intelligent Transportation


IEEE Conf. on Robotics and Automation, Brisbane, Australia, May 2018.


[52] Jon Keegan and Alfred Ng. There’s a multibillion-dollar market for your phone’s location data. Available online at https://themarkup.org/privacy/2021/09/30/theres-a-multibillion-dollar-market-for-your-phones-location-data.


