Enhancing Ride-Pooling Operations: Algorithms, Heuristics and Simulation-Based Approaches

by

Alexander Sundt

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Civil Engineering)
in the University of Michigan
2024

Doctoral Committee:

Professor Yafeng Yin, Chair
Professor Xiuli Chao
Professor Henry Liu
Associate Professor Neda Masoud

Alexander Sundt
asundt@umich.edu
ORCID iD: 0000-0003-0334-5963

DEDICATION

To Mom, Dad, Christina and Mandy

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor, Dr. Yafeng Yin, for his guidance and mentorship throughout all five years. I am sincerely grateful for his patience and understanding during many obstacles, and his encouragement and help in pushing through are likely the sole reason I completed this PhD. I truly admire his commitment to and genuine care for his students and for being such an incredible role model both professionally and personally.

I am deeply grateful to my committee members Dr. Henry Liu, Dr. Neda Masoud, and Dr. Xiuli Chao for their teaching and insight during my studies. I enjoyed learning from you all and really appreciate your willingness to answer questions and offer suggestions. Professor Masoud especially was a great mentor, advisor, and leader during my time on the board of the Michigan Transportation Student Organization (MiTSO) and I thank her greatly for her support.

I would especially like to thank my co-author and collaborator Dr. Qi Luo, the driving force behind Chapter 4 and his contributions in Chapter 3, and other collaborators Mehrdad Shahabi and John Vincent for their assistance, feedback, and support (on Chapters 2, 3, and 4) done for Ford Mobility. Their help pushed this research along and made it far better than I could do alone.

I owe a massive thank you to all of my professors and mentors from undergraduate and graduate school, including but not limited to Professors Jerry Lynch, Alexandre Bayen, Robert Harley, Scott Moura, and Dr. Alex Keimer. Your mentorship and incredible teaching led me to discover my own passions in civil engineering, transportation, and research that led me on this path and you helped me to develop incredibly useful skills that continue to benefit me in my academic and professional career.

I am deeply grateful to the friends I have made through Dr. Yin and the LIMOS lab. Zhengtian Xu, Daniel Vignon, Xiaotong Sun, Zhibin Chen, Tianming Liu, Zhichen Liu, Minghui Wu, Moji Abdolmaleki, Tara Radvand, Sina Bahrami, Tian Mi, Guoyang Qin, Manzi Li, and others: your questions and research have been both inspiring and extremely helpful in expanding my interests and deepening my knowledge. I treasure the light-hearted discussions we've had outside of research too, they brought joy to many days during my PhD.

To my MiTSO board members and other University of Michigan friends: Xingmin Wang, Zhen Yang, Xintao Yan, Yiyang Wang, Amir Tafreshian, Zachary Jerome, Lily Craighead, Ethan Zhang, Jisoon Lim, Iason Liagkas, Haowei Sun, Yan Zhao: You made my time at Michigan full of memories I will cherish. I'm proud of the work we accomplished with MiTSO and I'm glad to see it thriving and expanding its reach even farther. I hope to stay in touch so I can continue to be impressed by what you all accomplish.

Special thanks as well to the Center for Connected and Automated Transportation (CCAT) and especially Debbie and Calvin for their support of MiTSO, and for creating opportunities and financial support for transportation students like myself.

I would not be finishing this degree today except for my dear friends, whose support and company has kept me sane and entertained over the many years of college and grad school. Thank you Annie Shi, Amanda Zeng, Dennis Bradford, Ben Pridonoff, Mitch Deans, Zac Johnson, Maggie Yeh, Kiera Nissen, Henry Hammel, Alexis Flores, Bhavna Gopal, and so many more that I've been able to share time and memories with. Your friendships have brought light, humor, and empathy to these 5 years and I only hope that I can continue to repay in kind.

Last but not least, I am forever grateful to my loving parents and family who have always supported and encouraged me to pursue my passions and education. Mom, Dad, Christina: thank you for your unwavering love and for always being willing to listen and understand.

The work described in this thesis was partly supported by research grants from the Ford Motor Company, CCAT, the National Science Foundation, Rackham Graduate School and the University of Michigan.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES viii

LIST OF TABLES x

LIST OF APPENDICES xi

LIST OF ACRONYMS xii

ABSTRACT xiv

CHAPTER

1 Introduction 1

1.1 Background and Motivation 1
1.2 Challenges 3
1.3 Contributions and Outline 4

2 Mobility Profiles for Community-Based Ridesharing 5

2.1 Introduction 5
2.2 Literature Review 6
2.3 Data and Construction of Mobility Profile 7
2.3.1 Trajectory Mining: Inferring Work and Home Locations 9
2.3.2 Departure Time Distributions 10
2.4 Dynamic Time Warping 11
2.4.1 Dynamic Time Warping (DTW) Modifications 13
2.4.2 Binning and Distribution 15
2.5 Data Driven Matching 16
2.5.1 Mathematical Formulation for Matching Problem 16
2.5.2 Robust Data Driven for Matching Problem 17
2.5.3 Selecting Values for γ 18
2.5.4 Results 19
2.6 Conclusion 20

3 Heuristics for Customer-focused Ride-pooling Assignment 21

3.1 Introduction . . . 21
3.1.1 Main Contributions . . . 22
3.2 Literature Review . . . 23
3.3 Performance Measures of Ride-pooling Systems . . . 25
3.4 Ride-pooling Assignment Heuristics . . . 27
3.4.1 Preliminaries: Benchmark Ride-pooling Methods . . . 27
3.4.2 Restricted Subgraph Method: a Customer-Focused Heuristic . . . 30
3.5 Numerical Simulation on Real-World Data . . . 33
3.5.1 Simulation Environment . . . 33
3.5.2 Data Description . . . 34
3.5.3 Results and Discussion . . . 35
3.5.4 System Metrics . . . 36
3.5.5 Customer Metrics . . . 38
3.5.6 Statistical Test Results for Heuristics . . . 41
3.5.7 Results Takeaways . . . 41
3.6 Conclusion . . . 43

4 Efficient Algorithms for Stochastic Ride-pooling Assignment with Mixed Fleets . . . 44

4.1 Introduction . . . 44
4.1.1 Main Results and Contribution . . . 47
4.1.2 Organization and General Notation . . . 49
4.2 Literature Review . . . 49
4.3 Problem Description . . . 52
4.3.1 Basic Setting . . . 52
4.3.2 Reduction to Sample-Average Estimate . . . 56
4.3.3 Hardness and Properties of SRAMF . . . 58
4.4 Approximation Algorithms for SRAMF . . . 60
4.4.1 Local Search Algorithm for Mid-Capacity SRAMF . . . 60
4.4.2 Max-Min Online Algorithm for High-Capacity SRAMF . . . 66
4.4.3 Extensions to SRAMF under Partition Constraints . . . 71
4.5 Numerical Experiments . . . 71
4.5.1 Data Description and Experiment Setup . . . 71
4.5.2 Numerical Results for Mid-Capacity SRAMF . . . 76
4.5.3 Sensitivity analysis. . . . 79
4.5.4 Numerical Results for High-Capacity SRAMF . . . 79
4.6 Conclusion . . . 85

5 Conclusions and Future Research . . . 87

5.1 Research Summary and Findings . . . 87
5.1.1 Customer Preferences and Comfort (Chapters 2 and 3) . . . 87
5.1.2 Performance Metrics and Optimality (Chapters 3 and 4) . . . 88
5.1.3 Scalability to Large-Scale Real-Time Operations (Chapters 3 and 4) 88
5.2 Directions for Future Work . . . 89
5.2.1 Variance Across Cities . . . 89

5.2.2 Time Period Length and Rolling Horizon 89
5.2.3 Integration with Pricing Models 90
5.2.4 Transit System Coordination 90

APPENDICES 91

BIBLIOGRAPHY 105

LIST OF FIGURES

FIGURE

2.1 Density of trips in Beijing in the Microsoft Geolife dataset, with emphasis on the amount of similar routine trips among users 8
2.2 Trips in dataset plotted in space and time. Many trips are clustered around the same area at common times, allowing for pooling to be a feasible option. 9
2.3 Plotted location history for one user in the Geolife dataset, with lighter colors indicating a higher density of location points. Note that work and home locations are clearly identifiable based on density and clustering of GPS points, as well as the most preferred route. 10
2.4 Departure time distribution for one user's commuting trips 11
2.5 Example of DTW minimum cost warping path for two time-series. Reproduced from Mueen and Keogh (2016) 12
2.6 DTW with Sakoe-Chiba band constraint. Reproduced from Mueen and Keogh (2016) 14
2.7 Similar trajectories from two users found by DTW to have low distance cost . . 15
2.8 Two pairs of user trajectory history matched by the presented Robust Data Driven Optimization (RDDO) framework. User #30 was matched with user #1 and user #3 was matched with #35. Trips in similar areas are highlighted by the blue circles. 19

3.1 Target occupancy heuristic 29
3.2 Counterexamples for benchmark heuristics; $o_i - d_i, i = 1, 2, \ldots$ are origins and destinations of trip requests on a network, and edge values are travel costs. M is an arbitrarily large positive number. 30
3.3 Restricted subgraph heuristic 31
3.4 Visualization of the agent-based ridesourcing simulation in Manhattan, New York 35
3.5 Summary of results for system metrics 37
3.6 Summary of results for customer metrics 39
3.7 P-value of pairwise t-test. The smaller value indicates that the difference of system throughput using two algorithms is significant. The colorbar is in logarithm scale. 42

4.1 Example of ride-pooling with automated vehicles (AVs) and conventional (human-driven) vehicles (CVs). The first-stage decision involves repositioning AVs in dedicated regions; the second-stage decision is to solve a Generalized Assignment Problem (GAP). 46

4.2 The illustration of Stochastic Ride-pooling Assignment with Mixed Fleets (SRAMF) procedure per step. $S_B = \{s_1^B, s_2^B\}$ is the basis set (e.g., CVs) and $S_A = \{s_1^A, s_2^A\}$ is the augmented set (e.g., AVs). Figure 4.2a represents the algorithm's input, including the current locations of S_A and S_B, and obtains demand forecast. Figure 4.2b constructs a shareability graph for each scenario, where each trip is a clique containing one vehicle and multiple matchable requests. Figure 4.2c solves the SRAMF problem by approximation algorithms, in which one or more ride requests are assigned to a selected vehicle in each scenario ξ. Figure 4.2d implements the computed decisions and updates the system state. 54
4.3 Road-map for the performance analysis on SRAMF algorithms; the approximation ratios on arrows refer to the results in this chapter; S_O is the optimal selection of vehicles and S_R is the section of vehicles generated by approximation algorithms. 56
4.4 An example for non-submodularity of function $v^*(S_R)$. 59
4.5 Illustration of mapping $\Delta_d(e, f)$ with $E(\xi) = \{e_1, e_2, e_3\}$. 63
4.6 Mid-capacity mixed autonomy traffic experiment in Manhattan, NYC. 74
4.7 Optimal trip assignment and routes in the mid-capacity scenario. 78
4.8 Impact of K on computation time and optimality gap in the mid-capacity scenario. 80
4.9 High-capacity mixed autonomy traffic experiment in Manhattan, NYC. 81
4.10 Optimal trip assignment and routes in mixed autonomy, high-capacity SRAMF. 82
4.11 Impact of input distribution on computation time and optimality gap. 84

A.1 Diagram of relationships between classes in ride-pooling simulator 92

B.1 Topological relationship between cliques of matchable requests. In this example, (2,3,4) is not a valid combination of requests because the (2,4) combination was not valid. 99

LIST OF TABLES

TABLE

3.1 Summary of heuristic evaluation metrics 26
3.2 Summary of heuristics compared in numerical experiments 33
3.3 Summary of simulation scenarios 36
3.4 Summary of customer metric results, in minutes, for varying demand levels . . 40

4.1 Parameters in numerical experiments 74
4.2 Summary of Numerical Results for Mid-Capacity SRAMF 77
4.3 Summary of Numerical Results for High-Capacity SRAMF 83
4.4 Impact of vehicle capacity on computation time and optimality gap 84
4.5 Impact of sample size on computation time and optimality gap 85

B.1 Summary of notation and acronyms 95

LIST OF APPENDICES

A Appendix for Chapter 3 . . . . . **91**

B Appendices for Chapter 4 . . . . . **95**

LIST OF ACRONYMS

AV automated vehicle

CV conventional (human-driven) vehicle

CAV connected and automated vehicle

DTW Dynamic Time Warping

GAP Generalized Assignment Problem

GPS Global Positioning System

IP integer program

KNN k-Nearest Neighbor

LCSS Longest Common Subsequence

LP linear program

LSLPR Local-Search LP-Relaxation

MIP mixed-integer program

MMO max-min online

MoD mobility-on-demand

NYC New York City

O/D origin-destination

OSM OpenStreetMap

RDDO Robust Data Driven Optimization

RL reinforcement learning

SAA sample-average approximation

SRAMF Stochastic Ride-pooling Assignment with Mixed Fleets

TNC transportation network company

TSP Travelling Salesman Problem

VMT vehicle miles traveled

VRP Vehicle Routing Problem

ABSTRACT

The massive growth of ride-hailing and mobility-on-demand (MoD) platforms like Uber, Lyft, and DiDi, and advances in connected and automated vehicle (CAV) technology over the past decade have brought promising alternatives to car ownership within reach for many city residents. However, these services come with potentially severe negative effects on cities, such as increasing congestion and emissions due to empty miles. A useful tool to reduce these drawbacks is the introduction and promotion of ride-pooling, which accommodates multiple passenger requests in a single trip. Adopting ride-pooling on a real-time city-wide scale though has significant challenges including customer preferences, computational complexity, and demand uncertainty, all of which affect the benefits of the service. This dissertation aims to examine and address these issues in the context of ride-pooling operations.

The success of ride-pooling platforms hinges on whether customers will accept it over other alternatives; without enough demand, the system loses the efficiency of multiple customers per vehicle. To this end, we propose a community-based ride-sharing scheme where a system operator recommends travelers with similar travel patterns in order to address concerns about delay and safety while promoting a shared-vehicle environment. By leveraging trajectory data from routing apps, smartphones and CAVs, we can gather information about consumer preferences and construct a mobility profile for them. We modify a traditional Dynamic Time Warping (DTW) algorithm to compare trajectories in users' profiles and use the resulting measures as a basis for offline recommendation matching. We demonstrate this framework on data from the Microsoft Geolife Dataset.

Most ride-pooling platforms operate in a real-time environment, rather than offline, so it is important to consider operational challenges as well as those presented by demand. Most notably, solving a matching problem to not only pair riders but also assign groups of requests to vehicles is incredibly complex and time-consuming at scale. In the remaining sections of the dissertation, we introduce and analyze methods that are designed to be operable for a large-scale city. First, we develop a set of heuristic methods for ride-to-ride and ride-to-vehicle assignment that improves the customers' ride-pooling experience. In order to evaluate how changes in these methods and platform decisions affect all aspects of the system including customer waiting time and delay, we propose a family of metrics for evaluating

ride-pooling performance. We show that the proposed heuristics for the ride-pooling assignment are scalable and easily implementable methods and can be substitutes for centralized optimization in many scenarios, with only minor sacrifices in platform performance.

Second, while heuristics can achieve good performance in many scenarios, they provide no guarantees on performance in the worst cases. To address this, we formulate a joint vehicle repositioning and ride-pooling assignment problem as a two-stage stochastic integer program and expand it to the dual- or multi-source scenario in which the service provider can use different fleets of vehicles. Two approximation algorithms are proposed that provide competitive bounds on worst case performance. We then evaluate these approximation algorithms on real-world data using a simulator, demonstrating that these algorithms can parallelize computations and achieve solutions with small optimality gaps (typically within 1%).

The algorithms, frameworks, and takeaways presented in this dissertation were derived and evaluated for ride-pooling specifically, but many are generalizable to other shared mobility and multi-modal use cases.

CHAPTER 1

Introduction

1.1 Background and Motivation

With increasing concerns about climate change and our effect on the environment, encouraging carpooling and ride-sharing to shift away from single-occupancy vehicles is a necessary step in reducing carbon emissions and congestion on roadways. Recent studies in on-demand ride-pooling systems have shown a massive potential for ride-sharing leading to reduced vehicle miles traveled (VMT) and emissions in cities (Alonso-Mora et al. 2017b, Santi et al. 2014, Simonetto et al. 2019). The adoption of connected and autonomous vehicles can also help with reducing emissions via electrification and by smoothing traffic flow. However the benefit of CAVs largely depends on how and by who they are adopted (Kopelias et al. 2020). A model in which most people own a personal CAV can likely increase VMT and energy usage, by encouraging more and longer trips and even empty trips with no one in the car. On the other hand, a model with primarily on-demand shared rides features many more benefits and reductions in energy use (Greenblatt and Shaheen 2015). Thus it is imperative to pursue a modal shift toward carpooling and ride-sharing.

Though carpooling has been promoted many times throughout the past decades, an oft-cited reason for why participation has been limited is hesitancy about giving up the freedom of a personal vehicle (Greenblatt and Shaheen 2015). With the recent growth of transportation network companies like Uber and Lyft and the ease of requesting a ride on these platforms, this is less of a worry. The recently observed drop in car ownership means that relying on these services is not only feasible but also more accepted. However some notable barriers still remain to mass adoption of these services. Cost of these platforms can be an issue, and though sharing is cheaper, users are worried about safety when sharing rides with strangers. Thus, it may still be preferable for users to engage in a more traditional carpooling scenario, where passengers typically know each other well and costs are split among them long term. This, however, is limited by whether compatible carpools exist in

a passenger's social network. In this dissertation we introduce a framework for introducing travelers to others with similar mobility history, with the goal of expanding their network and encouraging pooling.

Unfortunately for many users, carpooling won't work for some of their trips due to inflexible timing or specific use cases. However we now have on-demand ride-sourcing as an option to fill in these gaps. The ridesourcing industry has recently grown tremendously due to the rise of transportation network companies (TNCs) such as Uber, Lyft, and DiDi Chuxing. Already serving billions of passenger trips per year, this industry has the potential to reshape cities, reduce the need for parking, and fortify the transportation ecosystem Wang and Yang (2019), Tafreshian et al. (2020). On-demand mobility services offered by TNCs also improve accessibility for those living in transit deserts with limited mobility choices.

While ridesourcing has revolutionized the ground passenger transportation market, it has also come with undesirable consequences. The increase in ridesourcing trips has drawn riders from public transit and increased the congestion in urban areas Henao and Marshall (2019), Hall et al. (2018), Luo et al. (2019). In San Francisco alone, the County Transportation Authority (SFCTA) found that TNCs were responsible for more than half of the 60 % increase in traffic congestion between 2010 and 2016 Hawkins (2019). The negative congestion externality of ridesourcing is primarily due to the increase in vehicular traffic demand. The convenience and flexibility of these services have both induced travel and encouraged a modal shift. The shift increases vehicular traffic demand if the original modes of transportation are non-motorized such as walking or biking. It also yields additional traffic demand if the original modes have higher occupancy, such as public transit and private driving. Compared to these modes, the average occupancy of ridesourcing vehicles is much lower because of the massive amount of empty miles traveled in the system. These are distances that ridesourcing vehicles travel when searching for or picking up a request, and would not happen if the trip was made in a personal vehicle. In summary, ridesourcing services have yielded an increase in VMT, causing congestion on city streets. The inefficient operations of these ridesourcing platforms can have severe environmental impacts due to increased energy consumption and emissions.

The potential for vast growth in the ridesourcing industry must be managed efficiently without slowing cities' traffic to a halt. Both planners and the services themselves have considered strategies to govern this. Transportation authorities may implement external regulations, such as capping the number of TNC vehicles or congestion pricing to stimulate behavioral changes in the ridesourcing market Luo et al. (2019), Erhardt et al. (2019). TNCs, internally motivated to enhance system performance by reducing the empty miles Braverman et al. (2019), have also implemented tactical policies, such as introducing meeting points so

that passengers can walk and shorten the pickup distance Stiglic et al. (2015). In addition, encouraging shared rides (termed as "ride-pooling") when possible is another effective way for TNCs to increase the utilization of ridesourcing vehicles. It can also help these systems be more financially stable, allowing the platform to charge cheaper fares, pay drivers more, and increase profit by serving more demand. However, the benefit of ride-pooling is dependent on a number of factors, including using efficient assignment and routing algorithms. In each assignment, we need to allocate multiple rides with compatible routes to one available vehicle and the assigned vehicle needs to determine the best route to pick up and drop off these rides. Thus, it is critical to study how to use ride-pooling services to efficiently offset the excess VMT and limit the negative externalities of conventional ridesourcing services. This dissertation examines the effect of various operational decisions on ride-sourcing systems, including matching algorithms and heuristics, and relocation choices.

1.2 Challenges

There are a number of challenges that carpooling and ride-pooling must overcome in order to be sustainable and profitable in the long term and widely accepted by the population. This dissertation aims to address the following:

- **Passenger comfort:** In order to encourage travelers to pool their rides with others and solidify a modal change in the future, their major concerns must be addressed. These typically consist of safety concerns about sharing with strangers, flexibility, and cost. We tackle these in a number of ways. By recommending other users with similar trajectory history, we expand users' social networks in ways they may be more comfortable with. Additionally, we cover metrics and heuristics that emphasize customer satisfaction and limit delays.

- **Scalability of algorithms:** When operating in real-time on a city scale, any proposed algorithm must be able to be handle thousands of requests per minute, leading to as many as 800,000 trips served per day in NYC for example. Many current optimization-based methods in the literature are too complex or too intensive to operate on such a large scale with short turnaround windows. We focus our methods on heuristics and approximation algorithms that scale better and achieve near optimal performance.

- **Stochastic nature of demand:** Due to the real-time nature of many ride-hailing platforms, it is uncertain where the next demand will exactly come from. This not only affects the decision of where to locate vehicles for low pick-up times, but also

how to pair trips for sharing. By determining fleet repositioning that maximizes the chances of pooling and makes the most use out of the available supply, we can potentially serve more demand than otherwise and reduce pick-up times. We develop algorithms that choose repositioning locations based on sampled future demand and provide performance guarantees for them.

1.3 Contributions and Outline

The rest of this dissertation is organized as follows. Chapter 2 focuses on the offline carpool matching problem, with the goal of helping travelers expand their social network to facilitate car-pooling groups. Using DTW, we measure differences in users' trajectories while accounting for traffic delays and time constraints. We then present a RDDO framework for using this historical trajectory information to find suggestions for pre-arranged ride-sharing partners.

In Chapter 3 we study the online vehicle-trip assignment problem commonly solved in ride-hailing systems like Uber and Lyft. We discuss various metrics for measuring the performace of these systems specifically in the case of ride-pooling, with respect to both customer and platform goals. A number of scalable heuristic methods are presented, tested, and compared using an agent-based simulator developed for this purpose.

Chapter 4 provides a combination of this online matching setting with fleet relocation, aiming to solve both the trip-vehicle assignment and relocation problem for stochastic demand. In order to make the problem more general, we also introduce the concept of mixed fleets. This allows us to handle autonomous and human-driven vehicle fleets, as well as luxury and standard fleets. More importantly, we use this setting to develop approximation algorithms with provable performance guarantees with respect to optimal.

With the exception of Chapter 2, algorithms and heuristics in this dissertation are tested on NYC taxicab data to demonstrate performance and scalability. Finally, Chapter 5 concludes this dissertation.

CHAPTER 2

Mobility Profiles for Community-Based Ridesharing

2.1 Introduction

With the rise and adoption of CAVs, car manufacturers and TNCs potentially have access to a plethora of new data. This data can reveal new insights into consumers' travel habits and help design new services. It can also be a really important tool for communities and companies to encourage carpooling and ride-sharing. Instead of simple origin-destination (O/D) information from traditional travel surveys, we can instead extract a time-sampled trajectory of users' routes, which is needed because users may not always prefer the shortest route from origin to destination. This could be due to other obligations along the route, such as dropping children off at school and picking up a coffee, or personal preference, like driving on local roads instead of a rush hour highway. Whatever the reason, accommodating these preferences is important when trying to encourage users to switch to carpooling instead of personal vehicles. The more the suggestions take into account these preferences, the more comfortable users will be with the service and more willing to change their travel mode.

Additionally, shared autonomous vehicles specifically allow for the combination of trips in ways that are not possible with traditional carpooling. Currently, the traveler that owns the vehicle is the most important trip, and other travelers that want to carpool with them must be picked up en route to the destination and dropped off either at the same destination as the driver or at a destination along the route. However with the introduction of autonomous vehicles, similar to the model of TNCs we have seen introduced in cities, this second requirement is no longer the case. Since the vehicles will be owned by the service or driven autonomously, it is possible to combine trips that overlap. The traveler who is picked up first can now be dropped off at any time, including before passengers who were picked up later. This allows for more efficient use of the vehicle.

However, we lack a comprehensive framework to take full advantage of this new data and sharing potential. This chapter presents a RDDO framework for using historical trajectory information to find suggestions for pre-arranged ride-sharing partners.

First, we construct a mobility profile for users based on their historical trajectory information. In order to compare these mobility patterns, we use a modified DTW algorithm to generate a distribution of distances between trajectories. This serves as a proxy for trip similarity: the shorter the distance between trajectories, the more similar the trips are in terms of route and departure time. This distribution is then used as a basis for a robust matching framework where matching is attempted based on a likelihood constraint for similarity.

We test the DTW algorithm on trajectories from the Microsoft GeoLife dataset, as well as perform robust matching on a small test case of 5 users.

The rest of this chapter is organized as follows: We first review previous literature in this area and discuss how it is lacking for this scenario. We then briefly describe our vision for a mobility profile and show how trajectory data can be used to fill out this profile. Next we introduce the dynamic time warping algorithm and modifications for comparing users historical trajectories, with examples of performance. Finally we formulate the robust matching model for matching ridesharing partners.

2.2 Literature Review

As mentioned earlier, traditional carpooling has been researched and promoted in the past with varying degrees of success. With the rise of TNCs making living without a car more accessible, and hesitancy about sharing with strangers still lingering, community-based carpooling seems well positioned for growth. Efficient matching for carpooling, however, needs location histories.

The availability of trajectory data from mobile phones spurred a lot of research in this direction. Importantly, it was shown that the vast majority of individuals typically follow a routine from day to day, and thus their trajectories and locations are not at all random (González et al. 2008). Location history data has been shown to be tremendously useful in predicting future behavior. Song et al. (2010) showed that it is possible to predict a user's location with 93% probability given their history.

While this research with mobile phone data is illuminating, mobile phone data can also be noisy, especially since data points often consist of simply which tower the phone is connected to. This may not even always be the closest tower. Additionally, the data can lack granularity especially with respect to route preference data, as multiple streets may lead to a connection to the same cell tower. Connected and automated vehicles have the potential to provide much

more accurate and granular data, especially since most modern vehicles come equipped with on board Global Positioning System (GPS) units. This data will not only be much more accurate with respect to routes traveled, but also with respect to trip start and end, as now it is no longer necessary to infer that from cell phone activity behavior, the car simply shuts off or starts when the trip starts.

There has been a plethora of research on trajectory clustering using all varieties of methods including euclidean distance, Longest Common Subsequence (LCSS), k-Nearest Neighbor (KNN), etc. (Zheng 2015). However trajectory clustering typically focuses on a goal classification or identification of future trajectories, rather than comparison for matching.

Recent research has focused heavily on on-line matching for TNC scenarios (Santi et al. 2014, Alonso-Mora et al. 2017b, Simonetto et al. 2019). While useful, these methods overlook the usefulness of historical data in predicting specific users trips. Additionally, some users are still hesitant to share vehicles with random strangers. This chapter emphasizes long-term pairings for commuter trips based on similar route preference to alleviate these fears. This can be further developed into a social network feature where users can find and befriend people based on similar activities and route locations. For example, if a user takes their child to school in the morning before traveling to work, they may find via this platform that their trip is similar another school parent who works nearby to them, expanding social networks and allowing users to become more familiar with potential carpooling partners.

There have been a couple papers relevant to this work. Ying et al. (2010) proposed a social network recommendation system for finding users with similar semantic trajectories. However these semantic trajectories exclusively consisted of activity patterns, rather than route similarity. Lee and Liang (2011) approached the problem with a similar idea as this chapter, recommending carpool partners based on similar routes, but used complex trajectory processing combined with a longest common prefix subsequence (LCPS) method due to noisy cell phone data. Additionally, none of these papers considered travelers' multiple different trajectories over their history in the matching.

2.3 Data and Construction of Mobility Profile

The data used in this chapter comes from the Microsoft Geolife Dataset Xie et al. (2009). Consisting of trajectory history data for 178 people over the course of 4 years, most in and around Beijing, this is one of the only large scale trajectory history datasets publicly available. Participants were asked to carry gps-loggers and phones, which recorded location for most trajectories every 5 seconds. This dataset contains trajectory data for a wide range of movements and activities, including many modes of transport. Figure 2.1 shows the trip

density in Beijing and the potential for trip sharing among users' frequent trips.

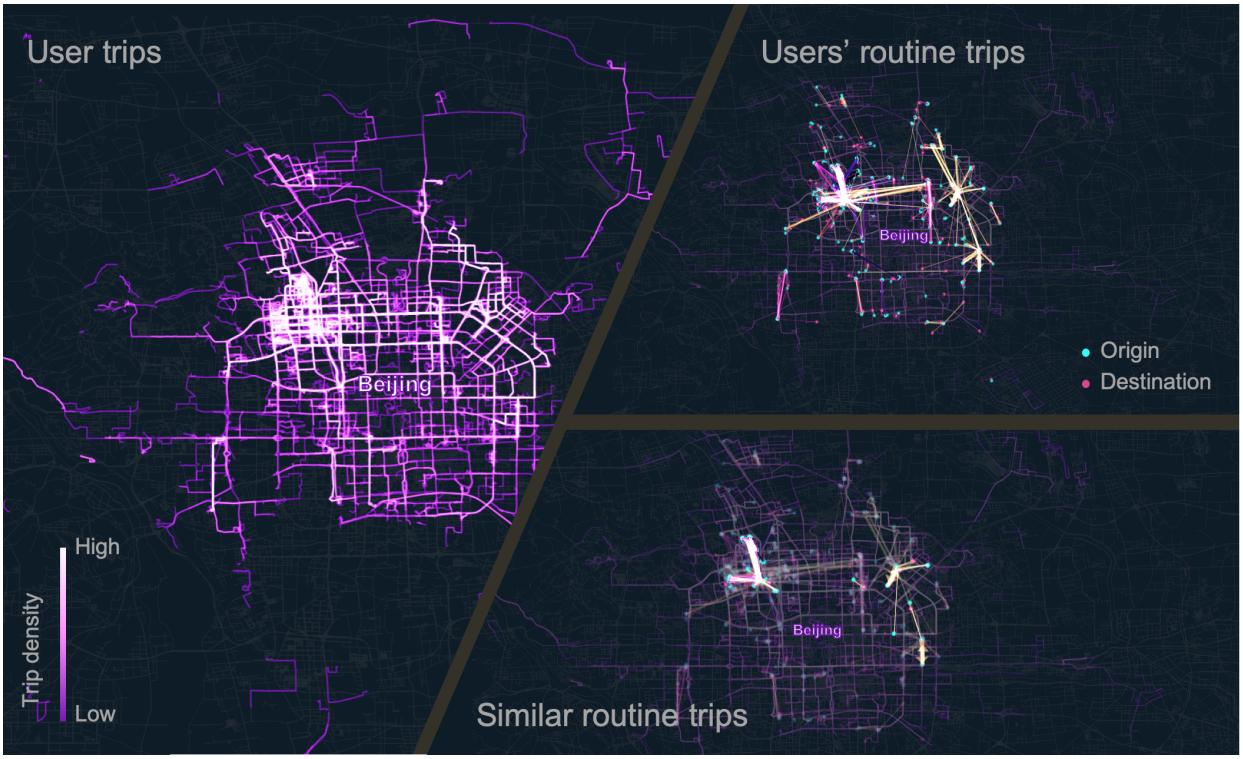


Figure 2.1: Density of trips in Beijing in the Microsoft Geolife dataset, with emphasis on the amount of similar routine trips among users

Figure 2.2 shows the distribution of trips in the data set both spatially and temporally. Note that there is a fair amount of potential for shared trips but that many happen over vastly different time periods. Thus it is important to take that into account when comparing trajectories for compatibility.

In this chapter we focus mainly on commuting trips between work and home, as these are the most frequent and predictable trips by travelers and are easiest to replace with carpooling rides. Given the data, we are able to construct a mobility profile with this information and filter for a trajectory history for trips from home to work or work to home. This mobility profile can be expanded to include information like preferred mode of transportation, and most visited places. This profile can also be merged with other sources of data, including activity check-in data from networks like Facebook and Twitter. Other papers (Xie et al. 2009) have demonstrated using location history data for friend recommendations and travel suggestions for new places to visit.

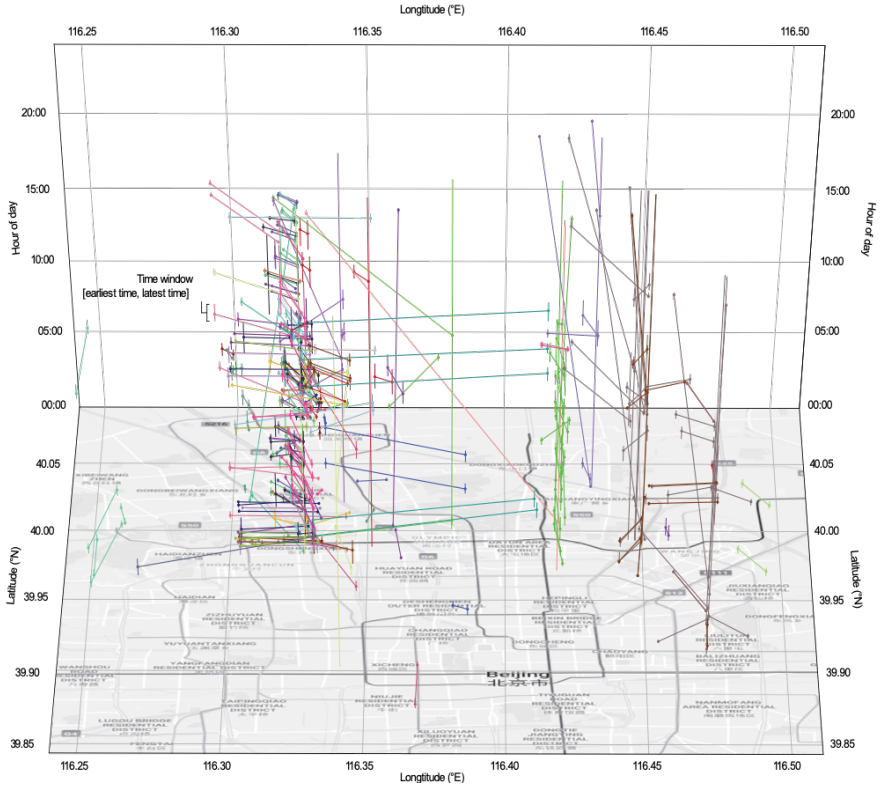


Figure 2.2: Trips in dataset plotted in space and time. Many trips are clustered around the same area at common times, allowing for pooling to be a feasible option.

2.3.1 Trajectory Mining: Inferring Work and Home Locations

Using trajectory data mining techniques, such as stay-point detection and clustering, we can easily identify users' home and work locations. Stay-point detection (as well as monitoring for dropped data points) is used to split up a continuous daily trajectory into trips and stops. These stops can then be clustered and counted to reveal travel patterns. Home and work locations are where users spend the most time, and these are often the 2 most visited places. Due to privacy constraints, note that these are not a specific point or address, but rather an area of small radius. Figure 2.3 shows a plot of all data points for one user in the data set, with lighter colors representing higher density of points. The home and work locations, as well as the most preferred route, can be easily identified using the density of points. To distinguish between home and work, we examine the time of day spent in each location, assuming the user spends the night at home. We then use these locations to look specifically for trips between home and work in our analysis.

Other techniques for trajectory data mining are summarized in Zheng (2015), including map-matching, noise filtering, and outlier detection. While these are useful tools when analyzing trajectories, they require significantly more analysis and computation. We present

Figure 2.3: Plotted location history for one user in the Geolife dataset, with lighter colors indicating a higher density of location points. Note that work and home locations are clearly identifiable based on density and clustering of GPS points, as well as the most preferred route.

our framework on data that has not been map matched or noise-reduced in anyway and note that these techniques will only serve to improve the accuracy of the framework.

2.3.2 Departure Time Distributions

It is also important to understand the distribution of traveler departure times for trips of interest in the data. Travelers prefer to leave around a certain time as dictated by trip activity, whether it is a requirement at the destination or due to conditions on the route. This is another aspect of user preference that is important to match in for users to be comfortable with and willing to accept suggested matches and trips in a carpooling scenario.

Figure 2.4 displays the departure time variation for one user's commuting trips between home and work. A large portion of the trips in the data set occur at or near a university in the northwestern portion of Beijing. If the participants in the study are students or professors, this might explain a wider distribution in departure times than expected given the nature of a changing class schedule each semester. Still, we observe the most common departure times fall within a 30-40 minute window. around 8:10am.

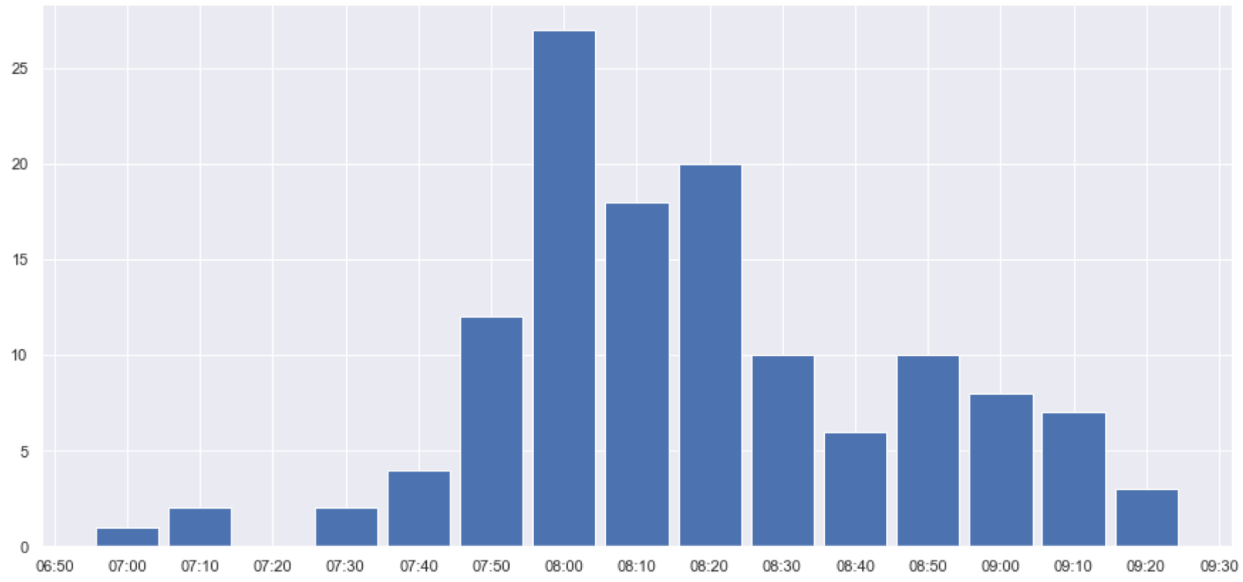


Figure 2.4: Departure time distribution for one user's commuting trips

2.4 Dynamic Time Warping

In order to compare mobility profiles for matching, we need to establish a method for comparing location history data. Dynamic Time Warping (DTW) is a dynamic programming algorithm for comparing time-series Senin (2008), Mueen and Keogh (2016). Initially used for speech recognition, DTW has been adapted and applied to many fields with time-series data, including motion-detection and finance. Contrary to euclidean one-to-one matching, where each time point is matched and compared to one other point corresponding in time, DTW allows for "warping", where many points in a series can be matched to one point in the other. In speech pattern matching, this is important to match the same word said fast or slow. In our setting, this is also important because two people taking the same route can have different travel times or speeds due to traffic or stoplights. A simple one-to-one comparison would introduce larger distances at points where one vehicle had a green light while the other had to stop, leading to an inaccurate comparison in route.

A warping path, w, is an assignment of points in one series to points in the other series. The basic "naive" dynamic time warping algorithm chooses a warping path that minimizes overall cost. Let $f_c(i, j)$ be the cost function for two inidices, i series Q and j in series C. In this case, cost between two points is the distance between them on earth's surface, as each point is a latitude/longitude coordinate. Let $D(i, j)$ be the cumulative cost so far for the minimum cost warping path matching all indices 0 to i in series Q to indices 0 to j in C. Since all points must be included on

$$D(i,j) = f_c(i,j) + \min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases} \tag{1}$$

This is a recursive formula that can also be calculated iteratively through careful ordering of the cumulative distance matrix. The boundary conditions for $i = 0$ and/or $j = 0$ are $D(i,j) = f_c(i,j)$. The iterative algorithm is summarized below. For a more in depth summary, see Mueen and Keogh (2016).

A visualization of the process can also be found in figure 2.5 from Mueen and Keogh (2016). Note the ability of DTW to match many points to one, and find a warping path that minimizes the overall cost (in this case, distance on the y-axis).

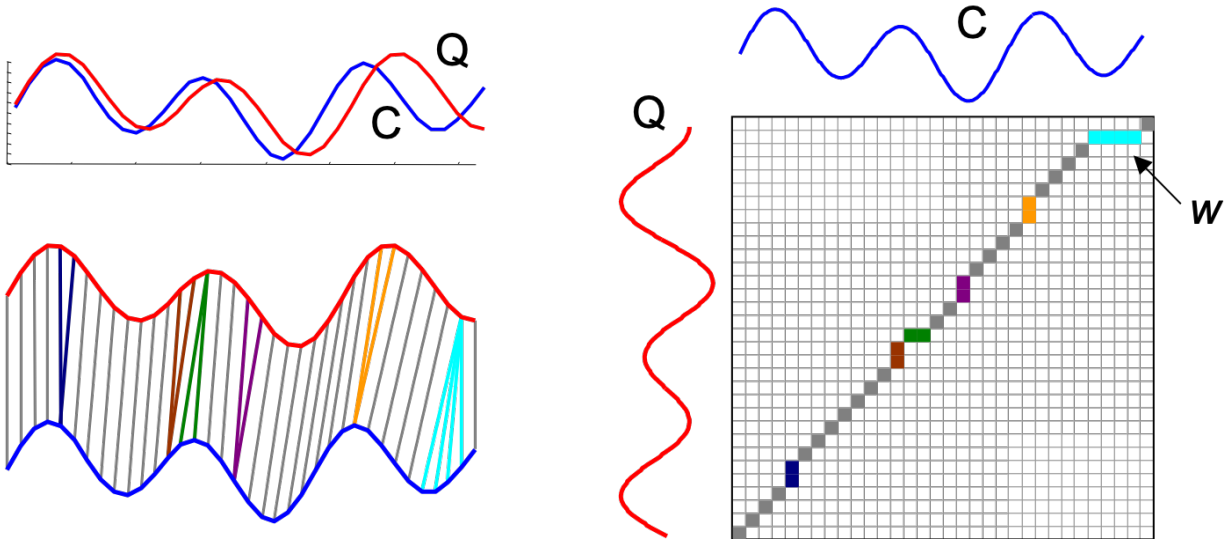


Figure 2.5: Example of DTW minimum cost warping path for two time-series. Reproduced from Mueen and Keogh (2016)

It is worth noting that we chose the DTW method presented in this chapter based on our desired behavior for a similarity function on the data we had. The benefits of DTW over a euclidean distance function have been summarized above, but other distance metrics can be used instead. A commonly proposed alternative for DTW is the Longest Common Subsequence (LCSS) method, based on the dynamic programming edit distance algorithm. While the LCSS method has potential benefits, like the ability to omit points from the warping path which limits the effects of outliers and noise, the cost function is typically limited to a constant based on a threshold value for distance. If the two points are within the threshold distance, they can be matched (given a cost of 1), while those farther apart have

a cost of 0. The total cost of a warping path is then maximized, finding the longest matching subsequence. While this has been used for trajectory clustering and can identify which parts of a trajectory are similar, it doesn't fully quantify the distance between the two trajectories outside the threshold difference. In addition, choosing a threshold distance is difficult and depends on the expected noise from the sensor. Instead, we address this method's sensitivity to noise by adjusting the histogram in the binning section of this report.

2.4.1 DTW Modifications

The naive DTW algorithm has a few notable shortcomings with respect to our scenario that need to be addressed:

- Doesn't consider timestamp associated with data point
- Doesn't account for departure time preference/flexibility
- Enforced pairing of end points of trajectories

These three points are actually closely intertwined and can be solved with just minor modifications. Naive DTW is mainly used for comparing the similarity of time-series in terms of their measured values, not necessarily the timestamps when they occurred. DTW is very good at identifying similar sequences, patterns, subsequences, etc., which is great if one just wants to identify or cluster similar trajectories and routes. However, the temporal aspect of these trajectories is also important when identifying ridesharing partners; the trips have to be compatible and occur in the same timeframe so that either user isn't delayed too much. This isn't as simple as comparing start and end times of the trip, as trips that overlap or are subsets only need to align in time at the points where the trips match, which can happen at any point in the middle of the trip, not necessarily at the start. Similarly, the historical trajectories in the data set capture departure time preferences of the users, and users would not necessarily be able to change their departure time by too much.

In order to address these two problems, we must first align the two trajectories with respect to time of day. Assuming that some amount of the trajectories overlaps in time (otherwise it is pointless to compare as the trajectories would be incompatible in a carpooling scenario), we can add points to the starts and end of these trajectories at the same sampling rate as the trajectories until both are the same length and start and end at the same time of day. Points added to the start of a trajectory should have the same location as the initial point of the original trajectory; points added to the end of a trajectory should be replicates of the original end point. This is the equivalent of a trajectory where a traveler is waiting at

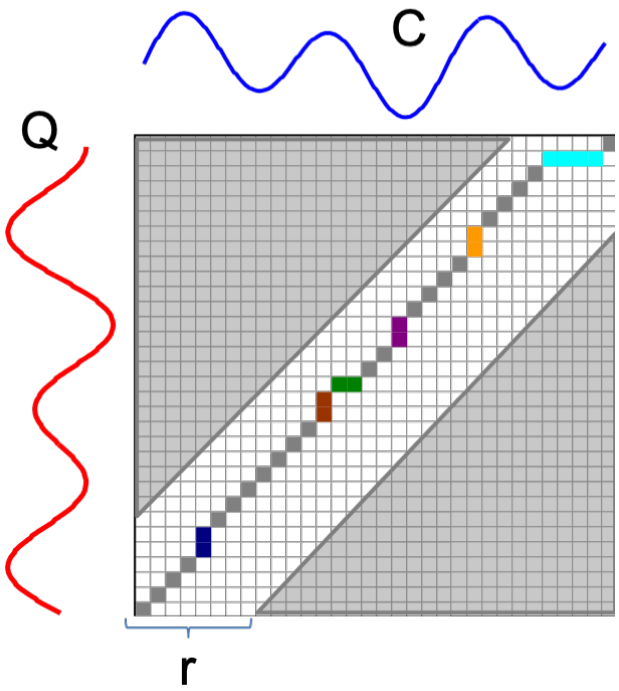


Figure 2.6: DTW with Sakoe-Chiba band constraint. Reproduced from Mueen and Keogh (2016)

their origin before the trip or at their destination afterwards. Note that because these points are replicas of start and end points, this does not affect the warping path of the original trajectories when using naive DTW.

It then becomes an easy modification to solve the departure time and delay flexibility problem using slope-limited dynamic time warping. This method limits the difference in indices that can be paired by the warping path. It functions as an extra global constraint limiting the space for the warping path so that trips cannot link up at points that are too far apart in index. This is also called a Sakoe-Chiba band in literature (Mueen and Keogh 2016), visualized in Figure 2.6. Since the indices are now exactly aligned in time, restricting the indices is equivalent to restricting the delay or difference in time between the trajectories. The index difference r can be calculated based on the sampling frequency and the desired departure time window. If the sampling frequency was once every five seconds and the departure time window was $\pm$ 10 minutes, this would yield $r = 120$.

However due to the enforced pairing of end points, this can potentially increase the distance returned due to having additional points that need to be paired. Although there are methods in the literature for flexible starting and ending points, they do not mesh well with the Sakoe-Chiba band introduced earlier. If we return a warping path from the DTW

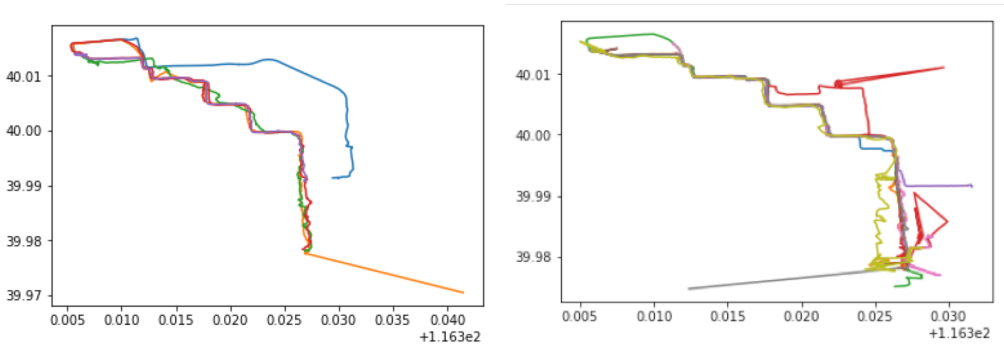


Figure 2.7: Similar trajectories from two users found by DTW to have low distance cost

algorithm, instead of just a distance, we can actually recalculate the total cost based on the start and end indices of the original trajectories. We use this value instead.

In summary, we propose and put into practice the following modifications for Dynamic Time Warping (DTW). For each comparison, we first check that if the trajectories overlap but do not start or end at the same time. If so we augment each trajectory with start points or end points equivalent to the original start and end points until both are same length. We then run DTW with a slope-limited constraint equal to departure time flexibility. From the returned optimal warping path, we recalculate the actual distance based on assignments of original (not augmented) trajectories.

An example set of trajectories returned using these methods is shown in Figure 2.7. These are trajectories from 2 different users that were identified to have low distances to both other trajectories from that user and to trajectories from the other user. Trips that are longer and also trips that are shorter are shown to be included as low distance.

2.4.2 Binning and Distribution

Because we assume users will have generated a number of historical routes with variations (not the same route each time), it is important to take all of these generated trajectories into account. So DTW is run on all possible pairs when comparing two users' trajectories, generating a distribution of distances.

This distribution then needs to be binned in a histogram to input into the robust matching formulation. This binning does not need to be regular; in fact it might be beneficial to use smaller bins for short distances and large ones for longer, as above some distance we can treat the trajectories as incompatible. The added granularity at smaller distances can help distinguish between better matches.

The exact choice of bins should depend on the magnitude of the trajectory noise and

the network topology. For example, the grid street network of Manhattan means there are possibly many ways to reach a destination. Since it is not significantly different if someone travels the same route but two blocks over, it might but useful to consider binning based on an average distance of 2 manhattan blocks. This will also need some trial and error to see what parameters perform the best.

2.5 Data Driven Matching

The goal of this section is to provide a Robust Data Driven Optimization (RDDO) framework to find the optimal ride matching partners which can be utilized in better planning for ridesharing systems. Once the distance between every two user's activities is determined, an RDDO model with likelihood bounds is employed to find the best matching partners for every two users. The model conservativeness and robustness can be controlled by tuning the likelihood threshold parameter in the data-driven uncertainty set of RDDO models.

We first provide a general formulation for a matching problem followed by the discussions on Robust Data Driven Optimization model for the matching problem and finally we propose a distributed optimization approach for solving the RDDO model for large scale problems.

2.5.1 Mathematical Formulation for Matching Problem

In this section we introduce the general mathematical formulation of the matching problem. Let C represent the set of all of the users, $x_ij \in \{0, 1\}$, $i, j \in C$ be the set of binary variables: if $x_ij = 1$ then user i is matched with user j, otherwise $x_ij = 0$. If two users are matched a certain cost of c_ij is incurred. Finally the mathematical formulation for the matching problem is presented as below:

$$\min_{x_{ij}} \sum_{i,j} c_{ij} x_{ij} \tag{2}$$

$$\text{s.t.} \sum_{i} x_{ij} = 1 \qquad \forall i \in C \tag{3}$$

$$\sum_{j} x_{ij} = 1 \qquad \forall j \in C \tag{4}$$

$$x_{ij} \in 0, 1 \qquad \forall i, j \in C \tag{5}$$

The goal of the matching problem above is to minimize the total matching costs (Equ. 1) while ensuring each user is assigned to only one. According to the combinatorial optimization literature the matching problem has a unimodular constraint structure therefore the

continuous relaxation of the binary constraints (4) and solving the resulting linear program would yield the optimal solution.

2.5.2 Robust Data Driven for Matching Problem

This section proposes mathematical formulations for a data-driven optimization modeling framework. To build the robust optimization model, we adopted a min-max perspective which minimizes the distance between two uses with respect to worst-case realization of the observed distances (Equ.5) which belongs to the data-driven uncertainty set. The uncertainty set is constructed through the likelihood function of distances between two users activities observed in multiple days.

Data Driven Uncertainty Set: The space where the uncertain parameter belongs to and is defined as the likelihood robust distribution set where the observed samples achieve an empirical likelihood level of γ. Mathematical equations for the uncertainty set is provided in below:

$$D(\gamma) = \left\{ p = (p_{ij}^1, p_{ij}^2, \dots p_{ij}^t) \mid \sum_t N_{ij}^t \log p_{ij}^t \geq \gamma,\ \sum_t p_{ij}^t = 1 \right\} \tag{6}$$

In the above equation p_{ij}^t is the probability that the matching cost for user i and j belongs to interval t with frequency of occurrence of N_{ij}^t. Given the uncertainty set $D(\gamma)$ the Robust Data-Driven Matching Problem is introduced as the following nonlinear programming formulation.

$$\min_{x_{ij}} \sum_{i,j} \left(\max_{p \in D(\gamma)} \sum_t p_{ij}^t c_{ij}^t x_{ij} \right) \tag{7}$$

$$\text{s.t.} \sum_t N_{ij}^t \log(p_{ij}^t) \geq \gamma \qquad \forall i, j \in C \tag{8}$$

$$\sum_t p_{ij}^t = 1 \qquad \forall i, j \in C \tag{9}$$

$$\sum_i x_{ij} = 1 \qquad \forall i \in C \tag{10}$$

$$\sum_j x_{ij} = 1 \qquad \forall j \in C \tag{11}$$

$$x_{ij} \in \{0, 1\}, \quad p_{ij}^t \in \{0, 1\} \qquad \forall i, j \in C \tag{12}$$

In the above formulation, Eq.(5), minimize the total matching cost while making sure the worst-case distribution achieved form the observed data reaches a desired level of likelihood.

Wang et al. (2016) provided the details on how to convert the min-max optimization above to an equivalent convex minimization which is readily solvable by the off-the-shelf commercial optimization solvers. In order to write the reformulation, we need to provide a closed form solution equation for inner maximization problem. We begin by writing the Lagrangian function of the inner this problem with respect to p:

$$L(p, \lambda, \mu) = \sum_{i,j} \Bigg(\sum_{t} p_{ij}^{t} c_{ij}^{t} x_{ij} + \lambda_{ij} \Big(\gamma - \sum_{t} N_{ij}^{t} \log(p_{ij}^{t}) \Big) + \mu_{ij} \Big(1 - \sum_{t} p_{ij}^{t} \Big) \Bigg) \tag{13}$$

Computing the KKT conditions for above equation and solving $\nabla L(p, \lambda, \mu) = 0$ would yield the optimal value for p_{ij}^{t} as below:

$$p_{ij}^{t} = \frac{\lambda_{ij} N_{ij}^{t}}{c_{ij}^{t} x_{ij} - \mu_{ij}} \tag{14}$$

Substituting the optimal value calculated for p_{ij}^{t} (Eq. 12) into equation (11) would provide an equivalent convex minimization reformulation for the problem.

$$\begin{aligned}
\min_{x_{ij}, \lambda_{ij}, \mu_{ij}} \quad & \Bigg(\mu_{ij} + \sum_{ij} \lambda_{ij} \Big(\gamma + N - \sum_{t} N_{ij}^{t} \log(N_{ij}^{t}) \Big) \\
& - \sum_{ij} N \lambda_{ij} \log \lambda_{ij} + \sum_{ij} \lambda_{ij} \sum_{t} N_{ij}^{t} \log(\mu_{ij} - c_{ij}^{t} x_{ij}) \Bigg) && (15) \\
\text{s.t.} \quad & \sum_{i} x_{ij} = 1 \qquad \forall i \in C && (16) \\
& \sum_{j} x_{ij} = 1 \qquad \forall j \in C && (17) \\
& \mu_{ij} - c_{ij}^{t} x_{ij} \geq 0 \qquad \forall i, j, t \in C && (18) \\
& \mu_{ij} \geq 0, \lambda_{ij} \geq 0 \qquad \forall i, j, t \in C && (19) \\
& x_{ij} \in \{0, 1\} \qquad \forall i, j \in C && (20)
\end{aligned}$$

2.5.3 Selecting Values for γ

Considering α as the reliability factor and $p = (p_{ij}^{1}, p_{ij}^{2}, \ldots, p_{ij}^{t})$ with $p_{ij}^{t} \geq 0$. The optimal value for likelihood value is achieved through the following equation:

$$P(p \in D(\gamma)) = 1 - \alpha \tag{21}$$

In the above equation P is the probability measure on p and the goal is to find a likelihood value γ by which the probability of p belonging to the uncertainty set $D(\gamma)$ is at least $1-\alpha$. Wang et al. (2014) showed that γ is calculated according to the following equation:

$$\gamma_{ij} = \sum_{t=1} N_{ij}^t \log\left(\frac{N_{ij}^t}{N}\right) - \frac{1}{2}\chi^2_{t-1,1-\alpha} \qquad \forall i,j,t \in C \tag{22}$$

Where $\chi^2_{t-1,1-\alpha}$ is the $1-\alpha$ quantile of a χ^2 distribution with t degrees of freedom.

2.5.4 Results

The algorithm and matching formulation were tested on a 5 person subset of the GeoLife dataset, chosen for having a fairly large amount of trips during rush hour

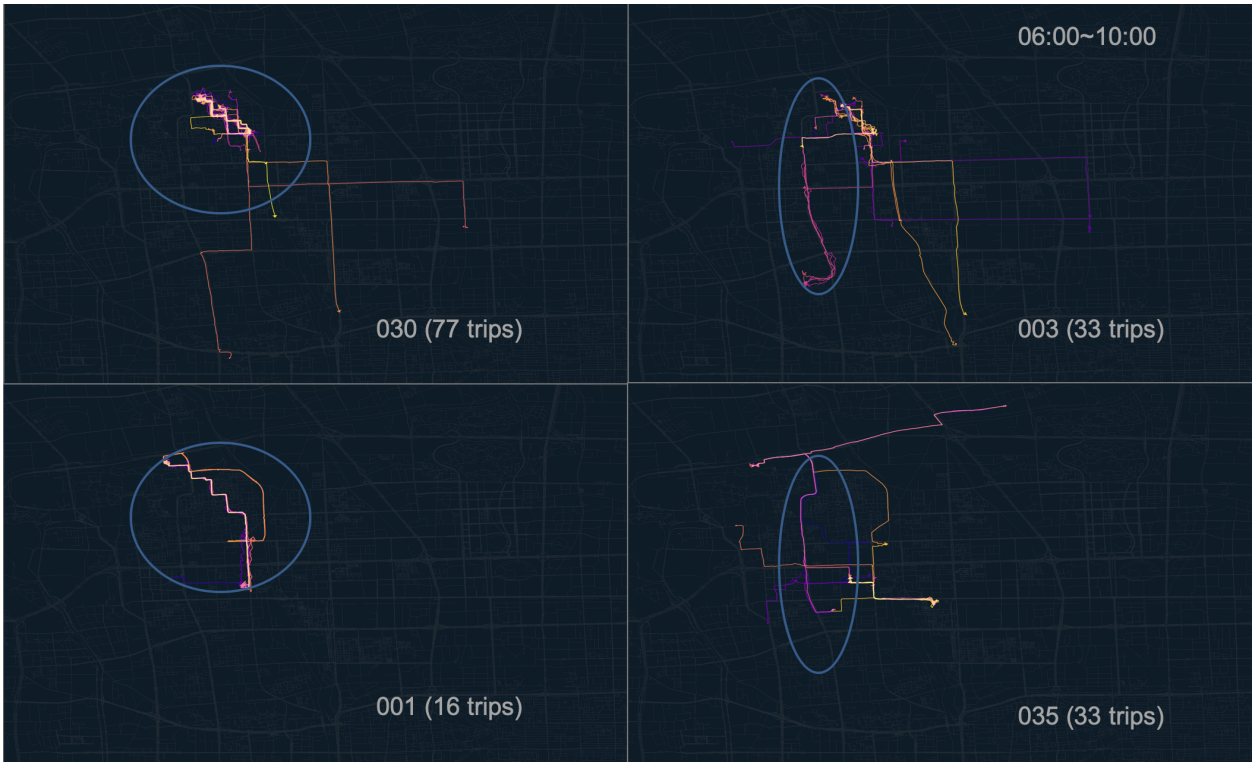


Figure 2.8: Two pairs of user trajectory history matched by the presented RDDO framework. User #30 was matched with user #1 and user #3 was matched with #35. Trips in similar areas are highlighted by the blue circles.

As seen in figure 2.8, the matching provides reasonable assignments. Note that the trips of users 1 and 30 line up often within the circled area, making them reliable candidates for sharing rides. Although user 3 also shares some of these same trips, they are a much less

reliable match. Instead, they share some less common but still frequent enough trips in the circled area on the right.

2.6 Conclusion

In this chapter we proposed a robust data-driven optimization framework to best use historical location and trajectory data to recommend community ride-sharing partners. In order to build the model, first the raw trajectory data which represents user's traces over time is processed. Such data is a multi-day high frequency GPS trajectories, by which one's frequent activities are revealed. In particular, in this work a trajectory mining approach is implemented which allows us to retrieve activities that happen frequently. Such information includes, the potential location for home and work and also the information regarding the recurrent trips for every user. With this information on hand a mobility profile for every traveler is constructed which forms the basis to understand every user's travel patterns. Finally, by comparing the mobility profiles of every two users the similarity of behavior between users activities can be established. This similarity is computed via a pairwise dynamic time warping algorithm with slope-constraints representing departure time flexibility. The pairwise comparison generates a distribution of distances, which is used to define an uncertainty set in a robust matching formulation. We demonstrate that the proposed DTW comparison and matching formulation captures efficient and useful carpooling assignments.

CHAPTER 3

Heuristics for Customer-focused Ride-pooling Assignment

3.1 Introduction

The ridesourcing industry has recently grown tremendously due to the rise of transportation network companies (TNCs) such as Uber, Lyft, and DiDi Chuxing. Already serving billions of passenger trips per year, this industry has the potential to reshape cities, reduce the need for parking, and fortify the transportation ecosystem Wang and Yang (2019), Tafreshian et al. (2020). On-demand mobility services offered by TNCs also improve accessibility for those living in transit deserts with limited mobility choices.

As mentioned previously, the benefits of ridesourcing are often outweighed by a host of negative externalities, including increased emissions and VMT. Facilitating ride-pooling is a useful method to offset these externalities and retain many benefits of ride-hailing. However ride-pooling introduces numerous other problems including making operation more complex at scale. It is well-known that the trip-vehicle assignment problem for ride-pooling is notoriously difficult, and most solutions involve computationally-heavy optimizations. Previous papers have quantified the benefit of ride-pooling by solving large-scale multi-vehicle dial-a-ride problems or approximate dynamic programming Santi et al. (2014), Alonso-Mora et al. (2017a), Yu and Shen (2019), Simonetto et al. (2019). Note that some of these papers assume an offline setting where trip requests are known in advance.

Solving ride-pooling assignment to global optimality for real-time operations is even more challenging, if not impossible. More practical alternative approaches include online approximation algorithms, reinforcement learning (RL), and heuristics (including metaheuristics). Online approximation algorithms can provide provable guarantees on the computed solution with respect to the optimum Ashlagi et al. (2018), Bei and Zhang (2018) in the assignment of trip requests arriving incrementally over time. However, these settings are restricted to

certain instances and are unrealistic because existing studies focus on analyzing the performance of these approximation algorithms in the worst case. These algorithms' performance on real-time spatial data from cities show various negative results. For example, (Tong et al. 2016) discovered that theoretically competitive algorithms might have worse average-case performance than a simple greedy assignment algorithm. Reinforcement learning is a purely data-driven approach Jindal et al. (2018) that has been used extensively in industry. Nevertheless, reinforcement learning may suffer from lacking adequate real-world data in early adoption and new areas (leading to poorly trained models) or may be lead to unpredictable behavior in unforeseen scenarios. This work aims to examine potential rule-based heuristic vehicle-trip assignment methods. These heuristic methods are not computationally intensive, easily scalable, and prioritize sharing rides over single rides.

Evaluating heuristics for ride-pooling can be difficult and subjective due to the lack of well-balanced performance measurements. Most heuristic methods in the prior work fall into the "platform-focused" catalogue Pelzer et al. (2015), Tong et al. (2018). The objective is serving the most trips in the fewest VMT. Nevertheless, implementing poorly designed heuristics may significantly increase customers' travel time compared with the conventional single-ride service. The platform must then compensate the customers by offering a discount. This work proposes a new "customer-focused" heuristic method to reduce the service degradation due to ride-pooling. This method guarantees the detours from the customers' trip plans are restricted, hence called the restricted subgraph method. This novel heuristic method could potentially achieve similar performance to optimization-based methods while simplifying calculations.

3.1.1 Main Contributions

The main contributions of this chapter include:

1. Suggest a set of well-balanced performance measures for evaluating ride-pooling systems.

2. Propose a new, simple but effective heuristic and examine a family of heuristic methods for ride-to-ride and ride-to-vehicle assignment that improves the customers' ride-pooling experience.

3. Build a data-driven agent-based simulator upon which to evaluate and compare different heuristics.

The remainder of the chapter is organized as follows. In Section 3.2, we review the relevant literature on offline and real-time ride-pooling assignment algorithms. In Section 3.3,

we suggest a comprehensive set of metrics to evaluate the performance of ridesourcing assignment methods. In Section 3.4, we propose a customer-focused heuristic based on evaluating restricted subgraphs of trip requests. In Section 3.5, we describe how the numerical experiments are conducted and discuss the numerical results. We draw the conclusion in Section 3.6.

3.2 Literature Review

There have been burgeoning studies to investigate the ride-pooling assignment problem. The goal is to dispatch ride-hailing vehicles to enable customers to share rides with others going in the same direction in order to offset travel costs with multiple passengers in the vehicle. Much of the prior optimization-based work assumes that trip requests are known ahead of time, and therefore the proposed approach is inherently offline. Others instead receive knowledge of trip requests in batches and perform computationally demanding optimizations to determine routes and matches. On the other hand, those for pure online implementation with unknown demand were designed against the worst case. In contrast, this work seeks to approach the middle ground for easily computable, real-time ride-pooling assignment methods.

Previous studies assuming that the platform has partial or full knowledge of travel demand focused on developing centralized, optimization-heavy approaches to improve the overall system performance. The resulting benefit of ride-pooling is over-optimistic, as real-time ridesourcing operations often do not know about future requests or have time to perform intensive calculations. (Alonso-Mora et al. 2017a) has indicated the substantial potential of high-capacity ride-pooling. This alternative on-demand transportation system can serve 98% of the demand with 15% of the fleet size when using a fleet of small busses with a capacity of 10 people. They used the idea of "shareability graphs" (vehicle-trip-request graphs) to optimize among potential matches of multiple trips and vehicles. This approach requires solving the Travelling Salesman Problem (TSP) repeatedly for each potential pairing of trips to check if constraints regarding pick up time and delay are satisfied and then requires solving an integer program to optimize the miles traveled. Both of these problems are highly computationally intensive and can scale exponentially with the number of requests. (Simonetto et al. 2019) improved the computational efficiency of the algorithm as a linear assignment problem and distributing the computation to multiple platforms. The results achieved a similar quality of service and ran up to four times faster than in the prior work. Since the assignment algorithm's performance varies significantly when the information about the next demand is unknown, this stream of literature can be treated as an upper bound on

any assignment policies in real-time/online ride-pooling systems.

On the other hand, there is a growing body of approximation algorithm literature that is derived from a worst-case analysis. These analyses are over-conservative because they examine the performance of algorithms when requests arrive in an adversarial order, which is designed to elicit poor results. Regarding the conventional single-ride vehicle assignment (i.e., each vehicle is matched with one trip request), (Tong et al. 2016) revealed that a simple greedy assignment algorithm, in which requests are matched on arrival to the closest driver, outperformed randomized online matching algorithms such as the permutation algorithm and the hierarchically separated tree algorithm. A potential reason is because those worst cases are rare in practice. (Ashlagi et al. 2018) proposed a passenger-to-passenger matching algorithm in ride-pooling assignment systems where each customer has either a constant or a random deadline. Their algorithm, based on the adversarial arrival order assumption, achieved 1/4-competitive for fixed deadline and 1/8-competitive when the demand process is memoryless. (Azar et al. 2017) initiated the studies of online matching with delays. This model captured the trade-off between the market thickness and information in the dynamic matching systems. In the context of ride-pooling, the longer customers stay unmatched, the higher probability that a better match can be found. For those works, ridesourcing platforms may pose the same question as to the single-ride case – how do they perform on the real-world spatial data? This chapter aims to cast light on this question.

Work in the area of real-time operation of ride-pooling systems has become more prevalent in recent years. The vehicle-trip assignment problem in ridesourcing is a variation of the dial-a-ride problem, i.e., designing efficient vehicle routes to serve the users' pickup and delivery requests between origins and destinations. Easily computable heuristic methods to solve the vehicle-trip assignment in ridesourcing were examined by (Hyland and Mahmassani 2018). This showed the potential of different methods of prioritization and matching, as well as considering varying scenarios of driver statuses. However, (Hyland and Mahmassani 2018) only examined the single-trip-per-ride case and did not consider the potential for pooling or sharing. (Herminghaus 2019) analyzed the efficiency of ride-pooling by using a mean-field approach in different urban settings. The ride-pooling market is characterized by aggregate variables such as the distribution of demand, the average delay of ride-pooling, and the traffic network structure. The model explained why mobility-on-demand is more competent in dense urban areas and revealed a break-even point for its deployment in urban areas.

3.3 Performance Measures of Ride-pooling Systems

Designing real-time ride-pooling heuristics starts with a clear goal in mind, but a consistent and objective measure of ride-pooling effectiveness is lacking in this avenue of research. The following performance measures are widely used to guide the design and the operations of ridesourcing services:

1. Maximize the utilization of ridesourcing vehicles Cramer and Krueger (2016).
2. Maximize average vehicle occupancy Di et al. (2017).
3. Maximize the platform's throughput Masoud and Jayakrishnan (2017).

However, these measures do not fully reflect the saved vehicle miles traveled (VMT) due to ride-pooling, which is the main benefit of its deployment. Utilization specifically does not distinguish between time spent detouring (increased VMT) and time spent en-route with multiple passengers (decreased VMT). Designing heuristics around these measures alone may also lead to unintended consequences. For example, if an algorithm cherry-picks short-haul trips, the system's overall throughput can significantly outperform other algorithms. Alternatively, if vehicles stay occupied when picking up new passengers, the vehicle utilization will appear high. Passengers may not benefit from these resulting ride-pooling algorithms due to being treated unequally or being forced on long detours and increasing VMT. To better evaluate the performance of ride-pooling systems, we propose a family of performance metrics summarized in Table 3.1.

Below we elaborate on the system throughput and system efficiency, as other performance metrics in Table 3.1 are self-explanatory and easily measured. For a study period, say, three hours, we discretize it into smaller time intervals of, e.g., five minutes each. At a given time interval τ, we let V_τ denote the set of in-service vehicles at the interval and K_τ^v denote the set of trips delivered by vehicle $v \in V_\tau$ during the time window $[\tau, \tau + \Delta\tau]$ where $\Delta\tau$ is the length of the moving window, e.g., 10 minutes.

We first introduce system throughput, which is defined as the number of passengers delivered to their destinations per unit of time. If real-time passenger dropoff data are available, as is the case in simulations in this chapter, the throughput Q_τ can be simply measured by the number of passengers dropped off within the window $[\tau, \tau + \Delta\tau]$ divided by $\Delta\tau$. Otherwise, it can be approximated by the ratio between the total number of passengers on board and the average in-vehicle time of passengers. As per Little's law, this approximation is exact if the system is in a steady state. Let O_τ denote the average occupancy of vehicles in V_τ. Mathematically, the throughput can be computed as follows:

Table 3.1: Summary of heuristic evaluation metrics

Evaluation metrics		Definition
Platform performance metrics	System throughput	Number of passengers delivered to their destinations per unit of time
	System efficiency	Average passenger-hours served per unit labor hour
	Time-based vehicle occupancy	Time-averaged number of passengers on board
	Distance-based vehicle occupancy	Distance-averaged number of passengers on board
Customer performance metrics	Matching time	Average time between when a trip request is sent and a driver is dispatched to the trip
	Pickup time	Average time between when a driver is dispatched and the customer is picked up
	Detour time	Average difference of in-vehicle time using ride-pooling relative to that of using single-ride service

$$Q_\tau = \frac{|V_\tau| O_\tau}{\sum_{v \in V_\tau} \sum_{k \in K_v^\tau} t_k^{\text{in-veh}} / \sum_{v \in V} |K_v^\tau|}, \tag{1}$$

where $t_k^{\text{in-veh}}$ is the in-vehicle trip time experienced by the passenger trip $k \in K_v^\tau$ served by vehicle $v \in V_\tau$ and $|\cdot|$ represents the size of a set. In the equation, the numerator is the total number of passengers on board while the denominator computes the average in-vehicle trip time.

System throughput, as defined above, is a good performance measure of the productivity or output of a ridesourcing system. However, it does not reflect the resource, i.e., labor hours, utilized to produce such an output. This is important in a ride-pooling scenario to quantify how much benefit is derived from sharing rides. More importantly, throughput does not reflect the length of the trips served. As previously mentioned, a matching algorithm that maximizes system throughout will tend to cherry-pick to serve short-haul trips. To address these two issues, we propose a system efficiency metric. The adjusted productivity or output at time interval τ will be the sum of the lengths of all trips (measured by their travel times if being served by single-ride service) delivered during the time window $[\tau, \tau + \Delta\tau]$. Mathematically, the system efficiency is defined as follows:

$$E_\tau = \frac{\sum_{v \in V_\tau} \sum_{k \in K_v^\tau} t_k}{\overline{V}_\tau \cdot \Delta\tau}, \tag{2}$$

where $\overline{V}_\tau$ is the average number of in-service vehicles in $[\tau, \tau+\Delta\tau]$, and thus the denominator assesses the total labor input. E_τ indicates the system outcome produced per unit of labor hours through ride-pooling. Note that for systems without ride-pooling, this metric reduces to the well-known system utilization metric, i.e., the percent time-in-service vehicles being occupied.

Note that Q_τ and E_τ, as defined, yield time-varying metrics of how a ride-pooing system performs during a study period. For the whole period, we report the averages of these time-varying metrics over all time intervals. Later we conduct statistical tests to determine how significant differences in these averages are over many simulation runs.

3.4 Ride-pooling Assignment Heuristics

In this section, we first introduce several basic versions of ride-pooling assignment methods widely adopted in industry and from literature. Note that most of these methods are platform-focused so customers may experience substantial degradation of service quality due to waiting and detour. Thus, we propose a novel customer-focused heuristic called the restricted subgraph matching method.

3.4.1 Preliminaries: Benchmark Ride-pooling Methods

We present three mainstream ride-pooling heuristics adopted in the industry or from the existing literature Yang et al. (2020).

Origin-destination (O/D) grouping or path clustering is a similarity-based algorithm that combines trips for ride-pooling based on the proximity of their origins and destinations. The advantage of the grouping method is that this type of trip similarity is an intuitive indicator for shareable rides without excessive detour time. Thus, a ride-hailing vehicle can serve matched trip requests by solving a shortest-path problem. On the other hand, two potential disadvantages emerge. First, ride-pooling efficiency is highly dependent on the thickness and homogeneity of the pooling market. If the market is not thick enough, the pooling will likely fail. Otherwise, passengers have to experience much longer matching time to be successfully pooled. Second, the system needs to determine sequences of pickup and dropoff after collecting all trip requests. The option for vehicles to accommodate new demand once

their route is determined is ruled out, missing potential compatible requests in the future.

A second class of heuristic is occupancy-based.

Algorithm 1: Target Occupancy Matching

Data: Passenger queue P, target occupancy $target$, idle drivers V_{idle}, occupied drivers V_{occ}, and acceptable matching radius r

Result: Assignments of passengers to en-route or idle vehicles

```
for p ∈ P do
    v* ← ∅ ;
    for v ∈ V_occ do
        if occupancy(v) < target and dist(p_o, v) < r then
            v* = v ;
            assign(p,v*);
    end
    if v* = ∅ and |V_idle| > 0 then
        v* ← arg min_{v∈V_idle} dist(p_o, v);
        assign(p,v*)
    end
end
```

The target occupancy heuristic aims to maintain a certain number of passengers per vehicle by alternating pickups and dropoffs. The main idea of the target occupancy heuristic is demonstrated in Figure 3.1. As requests are received, they are assigned to the nearest vehicle whose occupancy is below the target. This heuristic should benefit the ridesourcing platform, as it is a greedy algorithm that attempts to maximize the utilization of vehicles. However, since this simplified target occupancy heuristic does not consider requests and vehicle destinations, customers might experience substantial trip delays. To address this issue, we limit the assignment to vehicles within a certain matching radius. The geometric matching radius is effective for limiting both the delay for passengers currently in the vehicle and the pickup time for the assigned request. It is widely used in the ridesourcing industry Yang et al. (2020), Zha et al. (2018).

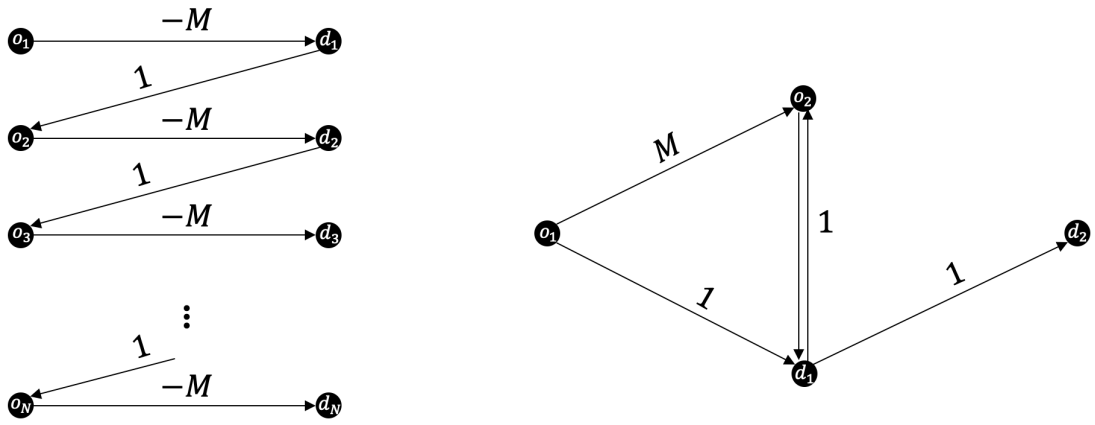
The final class of the ride-pooling method is bipartite matching or assignment regarding batched requests and vehicles. This method first constructs a bipartite graph in which available drivers (empty cars or cars with vacant seats) and trip requests sit on two sides, respectively. Since ride-pooling allows multiple requests to be assigned to the same vehicle, the system is solving a general assignment problem to minimize the total pickup time (or maximize the value of assignments). Efficient approximation algorithms Williamson and

Figure 3.1: Target occupancy heuristic

Shmoys (2011) can solve the large-scale bipartite matching (conventional) or assignment (ride-pooling) problem. The batch size is controlled by how frequently such a bipartite matching is conducted. The advantage of this approach is achieving the local optimum for the metrics of interest, in this case total system pickup time, while matching as many requests as possible. However, bipartite matching or assignment may cause additional waiting costs if the batch size of matching is extensive, and the detours for requests depend the method is used to determine trip compatibility.

The following counterexamples show how these these heuristics can be arbitrarily bad in specific networks.

We assume that all trip requests are revealed at time 0 and the platform operator controls one vehicle with capacity of two to fulfill these demand. The objective is to minimize the total cost of transporting passengers. In the first counterexample, the operator adopting the O-D grouping algorithm will assign the vehicle to an arbitrary trip to collect a total cost of $-M$, while the obvious optimal path is traveling a sequence of $o_1 \to d_1 \to o_2 \to d_2 \to \cdots \to d_N$ with the total cost of $N(1-M)-1$. The optimality ratio is an unbounded N. In the second counterexample, the operator adopting the target occupancy algorithm (with a target of 2) will choose the path $o_1 \to o_2 \to d_1 \to d_2$ with a total cost of $M+2$. A better path is $o_1 \to d_1 \to o_2 \to d_1 \to d_2$ with a total cost of 4. The optimality ratio is $(M+2)/4$, which is also unbounded when we drive $M \to \infty$. Note that in the second counterexample, the first passenger experience extreme detour from the use of target occupancy algorithm. These



(a) O-D grouping

(b) Target occupancy

Figure 3.2: Counterexamples for benchmark heuristics; $o_i - d_i, i = 1, 2, \ldots$ are origins and destinations of trip requests on a network, and edge values are travel costs. M is an arbitrarily large positive number.

examples motivate us to develop a family of customer-focused algorithms in what follows.

3.4.2 Restricted Subgraph Method: a Customer-Focused Heuristic

This section proposes a new customer-focused heuristic built upon the concept of a restricted subgraph. For each request received and each vehicle in transit, a matchable subgraph is created for the driver and the customer, respectively. For a vehicle with passengers, each of these subgraphs contains nodes that can be reached along its route within a reasonable delay; Similarly, for a customer seeking a ride-pool, their restricted subgraph contains nodes within an acceptable delay from their initial route.

The matchable node in a subgraph is defined as follows. A request going from node A to node C can reach node B if the sum of the travel time from A to B and the travel time from B to C is less than the original travel time from A to C plus an allowable delay δ as a function of the original travel time $\delta(tt_{AC})$, i.e., $tt_{AB} + tt_{BC} \leq tt_{AC} + \delta(tt_{AC})$.

The allowable delay function can be simple constants or reflect more complex, operational considerations. In our numerical results, we have chosen $\delta(tt_{AC}) = \sqrt{tt_{AC}}$, as this implies that the acceptable detour time grows with the length of the original trip, but also that for longer trips this delay should not grow at the same rate. For example, a 5-minute detour might be acceptable for a 10-minute trip, but a 30-minute detour would unlikely be

Figure 3.3: Restricted subgraph heuristic

acceptable for an hour-long trip.

As can be seen in Figure 3.3, constructing this restricted subgraph allows for a concise way of determining whether trips are compatible for pooling. This figure shows that the red origin falls inside the blue trip's subgraph, meaning the red trip can be picked up without too much delay for blue. Similarly, the blue destination falls within the red subgraph, suggesting that the blue trip can be dropped off without too much delay for red. If either trip did not satisfy these subgraph constraints (e.g., the yellow triangle on the map), it would be too far out of the way to yield a reasonable ride-pooling assignment.

Given the travel time (tt_{AC}) from A to C, this method considers all nodes within an ellipse of a certain focal distance from the origin and destination. This restricted subgraph method is easy to implement because the node-to-node distances can be pre-calculated. A similar method has been used to assign peer-to-peer carsharing Masoud and Jayakrishnan (2017) and check the detour propensity for ride-pooling in the mean-field setting Herminghaus (2019).

Algorithm 2: Restricted Subgraph Matching

Data: Passenger queue P with passenger origin p_o and destination p_d, idle drivers V_{idle}, occupied drivers V_{occ} en-route to destination v_d, and subgraphs G_i for passengers and occupied drivers

Result: Assignments of passengers to en-route or idle vehicles

```
for p ∈ P do
    v* ← ∅ ;
    for v ∈ V_occ do
        if p_o ∈ G_v then
            if v_d ∈ G_p then                                   /* Trips overlap */
                assign(p, v);
                v* ← v;
                Break ;
            else if p_d ∈ G_v and dist(p_d, v_d) < dist(p_o, v_d) then
                assign(p, v) ;                                  /* Passenger trip is a */
                v* ← v ;                                        /* subset of vehicle trip */
                Break;
            end
        end
    end
    if v* == ∅ and |V_idle| > 0 then                            /* p can be assigned */
        v^i ← arg min_{v∈V_idle} dist(p_o, v_o) ;               /* to idle vehicle */
        assign(p,v^i);
        V_idle ← V_idle \ v^i ;                                 /* Remove vehicle from idle set */
    V_occ ← V_occ ∪ v*;
    else
        V_occ ← V_occ \ v* ;                                    /* Remove vehicle from en-route set */
    end
end
```

Once the subgraphs are created, finding matches is a simple membership test. If a passenger, p, can share with a vehicle, v, carrying a different passenger en route to their destination, it must first be true that the vehicle can pick up p without delaying v too much. So p's origin, p_o, must be in v's subgraph, G_v. Additionally, one of the two following cases must be true: p's destination, p_d, must be in G_v OR v's destination, v_d must be in p's subgraph, G_p. The first case corresponds to the scenario where passenger p's trip is a subset of the vehicle's trip,

which means p will be picked up and dropped off before it reaches its original destination. The second case corresponds to when the two trips overlap. After passenger p is picked up, the vehicle will travel to its original destination, drop that passenger off, and travel to passenger p's destination. In our experiments with real-world spatial data, this proposed heuristic has performed well in maximizing the systems' throughput while keeping detours small.

In addition to the restricted subgraph heuristics, all the other pooling methods tested in this chapter are summarized in Table 3.2.

Table 3.2: Summary of heuristics compared in numerical experiments

Heuristics		**Description**	**Pros**	**Cons**
Customer-focused Heuristics	Restricted subgraph	Matching with earliest arrival passengers in subgraph	Easy-to-implement	Large pickup time
	Restricted subgraph greedy method	Matching with the nearest driver in subgraph	Small pick-up time	Long detour time in the worst case
	Bipartite	Matching to minimize delays	Obtain minimum total pick-up time	Incur waiting cost
Platform-focused Heuristics	Target occupancy with radius Yang et al. (2018)	Maintain occ. by adding rides in matching radius	Similar to industrial practice	Potential long trip time
	Target occupancy with one-sided subgraph	Combination of target and sub-graph heuristics	Trade-off between pick-up and in-vehicle time	−−
Baseline	Path clustering O/D grouping Hong et al. (2017)	Matching similar trips by O/D location	Near-optimal for offline assignment	Unsuitable for real-time assignment
	Single-ride Schreieck et al. (2016)	−−	−−	No ride-pooling

3.5 Numerical Simulation on Real-World Data

3.5.1 Simulation Environment

In order to compare the performance of these heuristics and evaluate their ability to pool requests, we develop an agent-based simulation environment based on a real-world taxicab dataset. Figure A.1 is a sketch of the architecture of the simulator. At each timestep,

all agents in the simulation are updated, and new trip requests are collected or generated. Drivers are updated first, during which they drop-off or pickup customers if appropriate. Next, waiting customers are updated and determine whether they leave the queue based on their threshold waiting time. Similarly, idle drivers can move randomly or based on given information about supply and demand. Finally, the platform is updated with all the new requests and driver statuses at every matching interval. The platform then performs a new matching assignment and updates customers and drivers with the results. Detailed statistics on miles traveled, driver status and occupancy, and waiting times are updated at each timestep.

This object-oriented simulation environment is designed to be flexible and allows for varied driver, customer, and platform behaviors. In this chapter, we assume idle drivers cruise randomly throughout the network when unassigned. However, other behaviors such as navigating towards areas of high demand or platform repositioning are also supported. Additionally, this chapter only places constraints on customer matching waiting time behavior and does not investigate different customer pickup waiting time behavior, but this is also supported. The flexibility of this environment (in Python 3.7) allows for potential testing of a multitude of different scenarios with minimal changes. Further information on the capabilities of the simulation and information on its development can be found in Appendix A. The testing of the various heuristics in this chapter is done simply by changing the platform matching function and the randomly sampled demand. Other tests can be done by changing the network, number of drivers, or other parameters easily.

3.5.2 Data Description

The data used for the simulation are drawn from multiple sources:

1. Trip request data: the NYC Taxicab data set available from the NYC Taxi and Limousine Commission Dias et al. (2019).

2. Speed data: average street speed data from the Uber movement project Aryandoust et al. (2019).

3. Street network data and subgraph network representation: OpenStreetMap (OSM) road network is converted to a graph in Networkx.

The NYC taxicab data consists of pickup and dropoff latitude and longitude locations for every taxi ride in NYC for 2011-2014. Data from 2013 was used specifically as it is widely used in other papers Alonso-Mora et al. (2017a), Simonetto et al. (2019) and it is the most recent NYC data containing latitude and longitude coordinates instead of more general zone

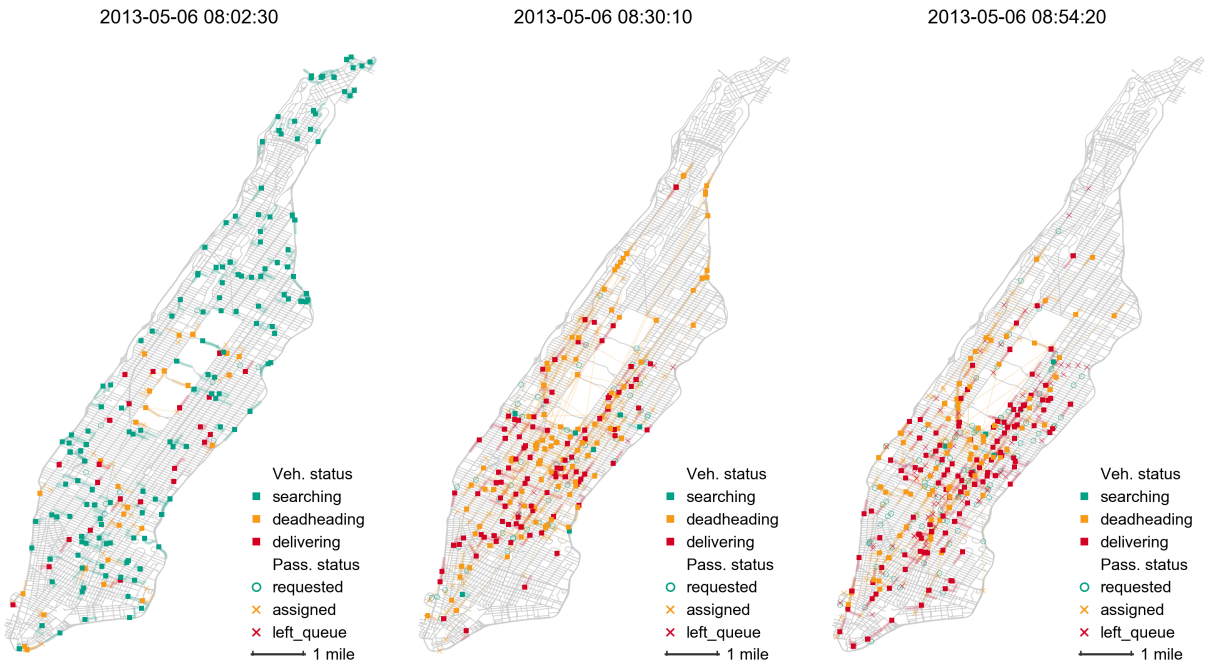


Figure 3.4: Visualization of the agent-based ridesourcing simulation in Manhattan, New York

information. In addition, the data contain fare and distance information for the trip, which are not used in the simulation. Since this chapter considers low capacity matching for a ridesourcing service, we assume each request to be two or fewer people per request (as is commonly required in TNC pooling services). All heuristics in this chapter support larger request sizes. However, a capacity constraint check would need to be added to make sure that vehicle capacities would not be violated in the matching assignment.

Network average travel times are obtained from the Uber Movement speeds dataset, averaged over 2017-2019 for all links, and synced with the underlying subgraph. Links without any information are given average speeds according to their OSM road type (i.e., highway, primary, residential).

3.5.3 Results and Discussion

In order to compare multiple heuristics by statistical tests, we need to specify how to randomly sample supply and demand in the simulator.

We define a scenario as a sampled supply and demand profile and an initial condition for ride-pooling. In each scenario, the demand is randomly sampled from the NYC taxicab data from May 6th, 2013, 7:30 to 9:00 am. In each test, the fleet size is set to 500 vehicles with

Table 3.3: Summary of simulation scenarios

Percentage of demand	# of scenarios per heuristic	# of vehicles	Demand-supply balance
7%	10	500	over-supply
10%	10	500	nearly-balanced
12%	10	500	under-supply

random initial locations in the road network, and the simulation is run for a warm-up period from 7:30 to 8:00 am. The timestep for updates is every five seconds. The matching interval is also set to five seconds for all heuristics except bipartite and group methods, which use a one-minute interval. Additionally, the target occupancy in the relevant heuristic is two requests per vehicle, and the restricted subgraph methods use an allowable delay function as described earlier. All groups of heuristics are examined with the same initial conditions, but the system evolves differently once trip assignments are made.

Table 3.3 summarizes the different scenarios run in simulation. Three different demand cases are examined by sampling the demand at 7%, 10%, and 12% of the total Manhattan demand over the simulation period. Because the fleet size remains constant, these demand levels represent oversupply (enough vehicles to serve all trips with one request per vehicle), nearly balanced (most strategies can manage the demand), and severe under-supply (where many customers leave before being served, regardless of matching strategy), respectively.

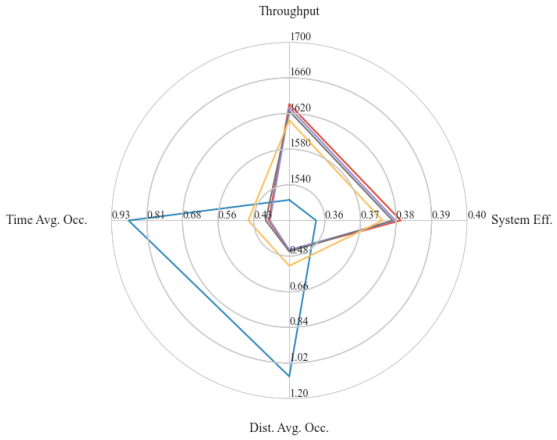
In order to directly compare performance across heuristics, we run ten simulations at each demand level. In each run, a new set of requests are sampled from the taxicab dataset, and new initial vehicle positions are generated. The demand and initial positions are kept the same across the various heuristics so that the runs could be statistically compared.

In accordance with the proposed performance measures, we report the comparisons of heuristics by system and customer metrics.

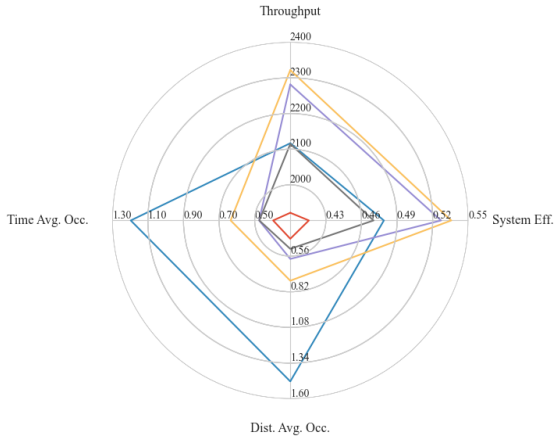
3.5.4 System Metrics

We examine the system performance metrics of these methods in closer detail in those three scenarios. These metrics are plotted for comparison in the radar charts in Figure 3.5.

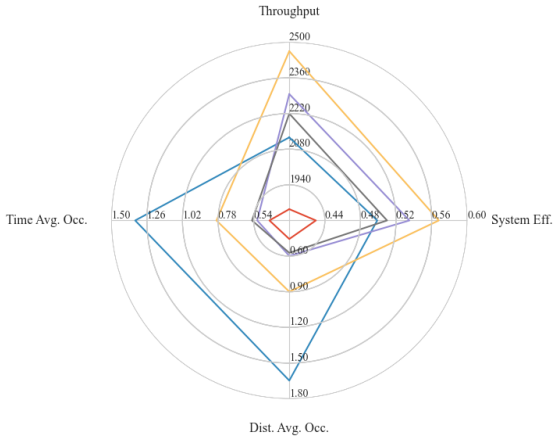
In the oversupply scenario (7% demand), because all requests can be served without ride-pooling, the single ride and greedy restricted subgraph methods outperform the others in both throughput and system efficiency, with no significant difference between them. As seen in Figure 3.5a, there is a less than 5% difference between the throughput of other methods except for the target occupancy method. The target occupancy method performs quite



(a) Oversupply (7% demand)



(b) Nearly-balanced (10% demand)



(c) Under-supply (12% demand)

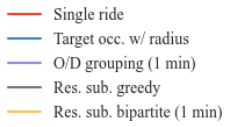


Figure 3.5: Summary of results for system metrics

poorly while maintaining high occupancy because it may cause long detours by forcing rides to share and low throughput.

In the nearly-balanced scenario (10% demand), a much larger variation can be observed between the methods. Origin-destination (O/D) grouping and bipartite methods, due to their longer matching window or centralized optimization, outperform the other methods in throughput by about 10% (200 requests per hour). However, this is a trade-off with the additional matching time incurred. Shorter matching windows yield lower overall throughput due to less time to group or consider requests for optimality. It is also important to note that the performance of the O/D grouping method is highly dependent on the spatial distribution of demand. It is very effective only when many requests share similar origins and destinations.

As vehicle supply becomes exceptionally constrained in the 12% demand scenario, the advantage of the restricted subgraph method and its extensions becomes clearer. Though statistically significant, the O/D grouping platform's average throughput is only ten requests per hour more (or less than 1%). Meanwhile, the subgraph method depends less on the distribution of demand and has a lower matching time. In most cases, we observe a correlation between throughput and efficiency, as higher efficiency serves more customers with the available labor supply. However, we note that the target occupancy is a consistent exception to this, as it achieves comparable efficiency to some other methods by getting people into vehicles as quickly as possible, but often suffers in throughput due to long detours. This result demonstrates the danger of comparing performance based on only one metric.

3.5.5 Customer Metrics

While throughput and efficiency are essential to the platform and regulators, they do not have a direct impact on customers. Thus it is also important to examine metrics that affect customers' experience. There is a clear trade-off between customer satisfaction and platform goals: longer matching time intervals for holding and optimizing requests produce larger throughput and more efficient matches but at the cost of customers' matching time. We also see the effects of different strategy choices. For example, the target occupancy method achieves low pickup times in general due to its matching radius requirement. However, it also experiences long delays because the trip assignments do not take destinations into account.

In the oversupply scenario (7% demand) in Table 3.4, the matching time is significantly increased for the bipartite matching and O/D grouping algorithms, as they employ a longer matching interval. This is disadvantageous to the consumers because they are forced to wait longer in uncertainty (i.e., they do not know when they will be matched), with often more extended pickup or trip times. We also see that the greedy restricted subgraph algorithm

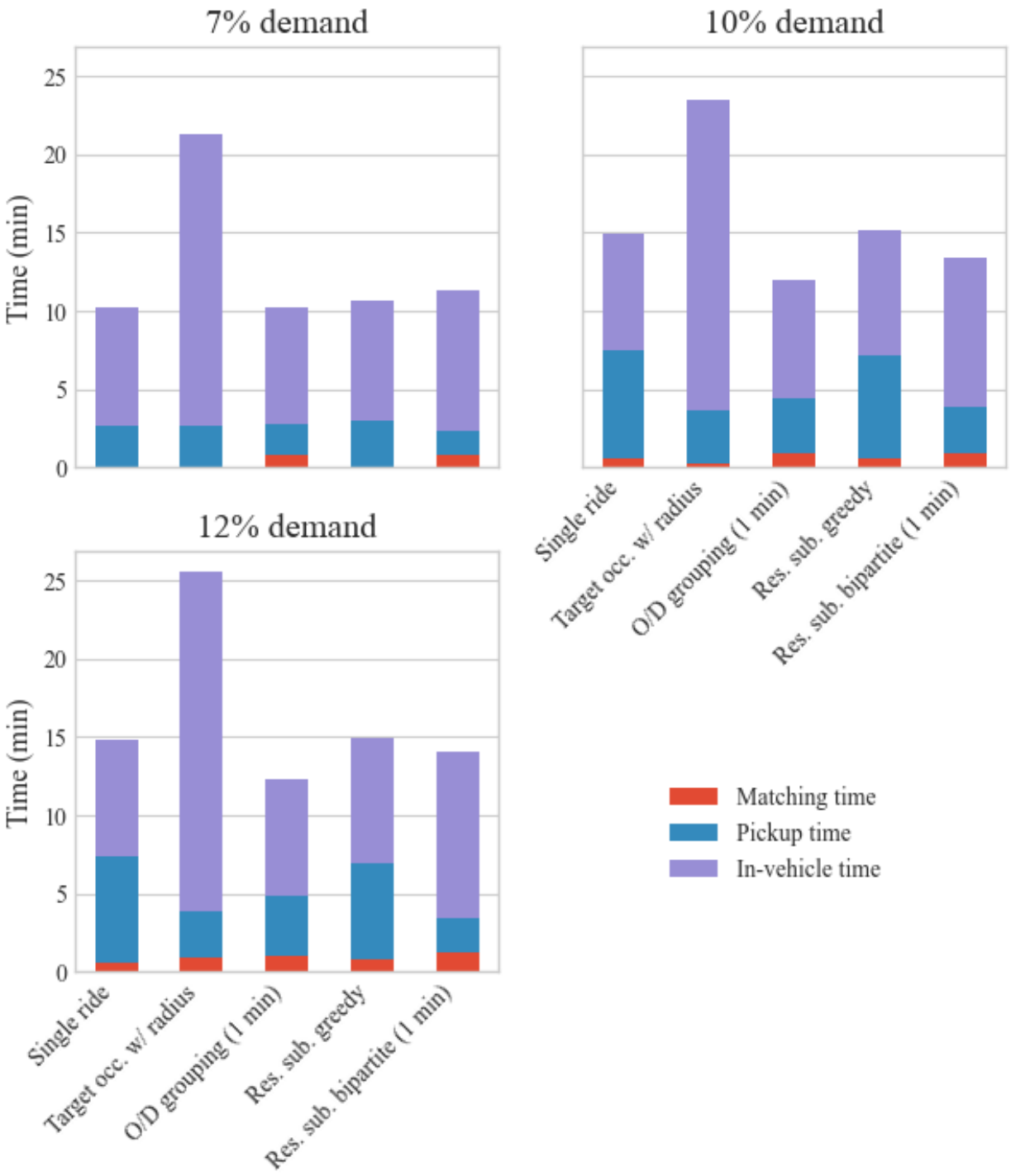


Figure 3.6: Summary of results for customer metrics

Table 3.4: Summary of customer metric results, in minutes, for varying demand levels

Over-supply case with 7% demand					
Platform	Average matching time	Average pickup time	Average trip time	Average total wait time	Average delay time
Single ride	0.07	2.75	7.55	2.82	0.74
Target occupancy with radius	0.07	2.78	18.60	2.85	11.64
Restricted subgraph greedy method	0.07	2.69	7.61	2.76	0.80
Res. sub bipartite matching (1 min)	0.75	2.57	9.13	3.32	2.35
O/D grouping (1 min)	0.75	3.73	7.66	4.48	0.82
Nearly-balanced case with 10% demand					
Single ride	0.52	8.05	7.51	8.56	0.74
Target occupancy with radius	0.25	3.59	19.70	3.84	12.77
Restricted subgraph greedy method	0.58	7.51	8.01	8.09	1.34
Res. sub bipartite matching (1 min)	0.90	3.74	9.88	4.64	3.16
O/D grouping (1 min)	0.86	6.09	7.64	6.95	0.83
Under-supply case with 12% demand					
Single ride	0.52	7.96	7.49	8.48	0.74
Target occupancy with radius	0.82	3.79	21.64	4.60	14.72
Restricted subgraph greedy method	0.76	7.40	8.18	8.17	1.53
Res. sub bipartite matching (1 min)	1.24	3.43	10.92	4.68	4.30
O/D grouping (1 min)	1.01	8.06	7.66	9.07	0.83

reduces the pickup time as intended in this case.

In the nearly-balanced scenario (10% demand) in Table 3.4, we see a significant reduction in pickup time using either the target occupancy and the bipartite matching method compared to the other platforms. This reduction comes at the cost of increased matching time, which customers might place a great value on, given the uncertainty of whether they will be assigned a vehicle at all.

Finally, the waiting and delay times increase further as supply is more constrained and more requests are served in the 12% demand scenario. In Table 3.4, we see the same trends in average matching time and pickup time, as well as increased trip delays in the target occupancy and bipartite methods. As more demand is trying to be served by the same amount of vehicles, the increased throughput comes with forcing longer delays. We present the restricted subgraph methods as a balance in these trade-offs between the platform's goals and customers' satisfaction, though TNCs and planners can determine what is best for specific scenarios by weighing more important metrics.

3.5.6 Statistical Test Results for Heuristics

We perform a pairwise t-test to compare throughput performance based on ten simulation runs at each demand level. The null hypothesis, h_0, is that the methods do not significantly differ in the system throughput. The throughput and p-values for the 12% demand scenario comparisons are reported in Figure 3.7. We only present the 12% demand statistical comparison here for brevity. The tests show statistical significance in most demand levels in all pairs except between the restricted subgraph method and the greedy restricted subgraph, which only have a statistically significant difference in the 7% demand scenario. It is important to note that, while these differences are often statistically significant, they are not always practically significant. As mentioned earlier, in some cases, the difference between O/D grouping and restricted subgraph methods is less than one percent.

3.5.7 Results Takeaways

In general, our results show that:

- The restricted subgraph method performs significantly better than the target occupancy method, a method practiced in industry, in all cases due to increased throughput and limited detour time.
- The performance of the origin-destination (O/D) grouping method, another one practiced in industry, depends on tuned parameters (e.g., the size of matching zones and the

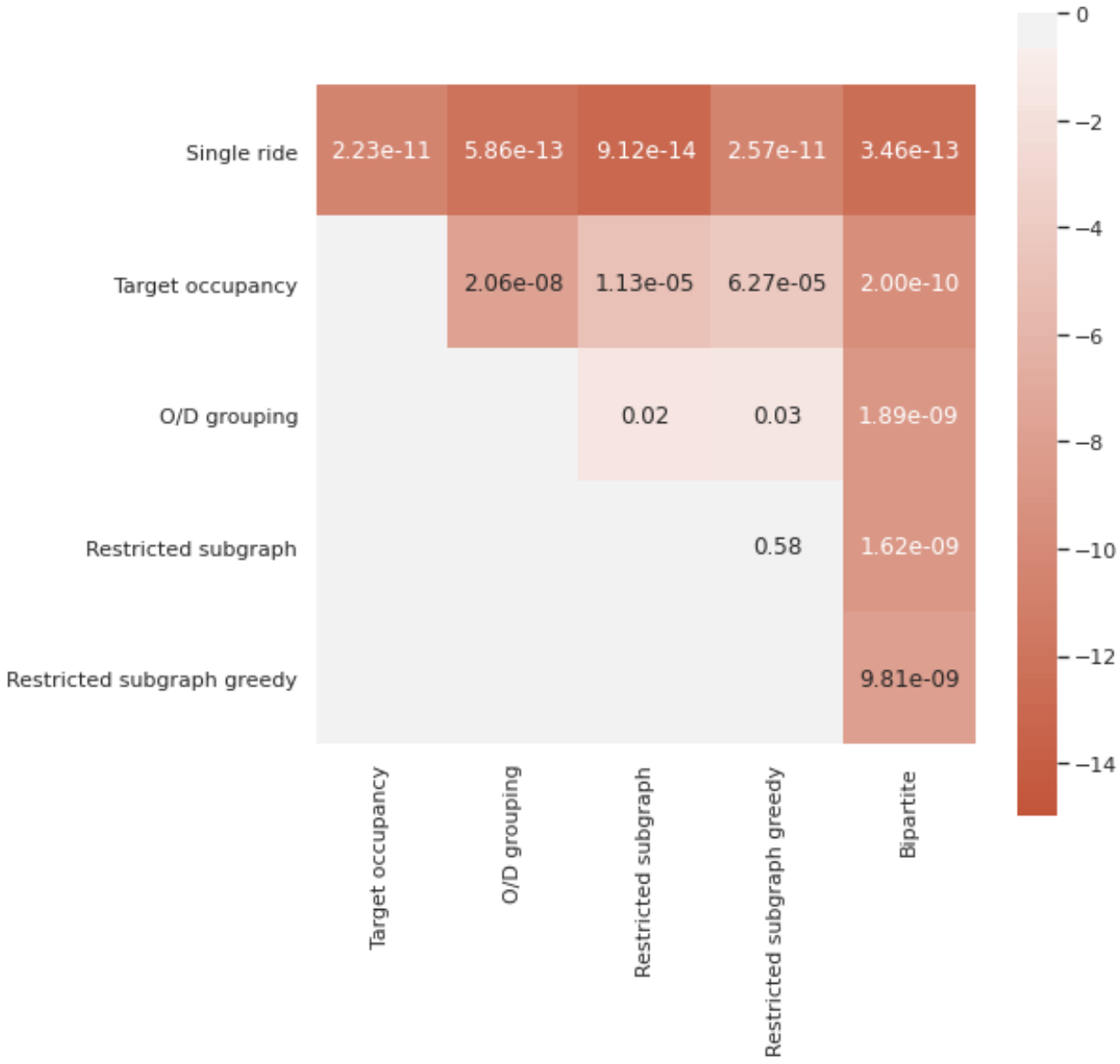


Figure 3.7: P-value of pairwise t-test. The smaller value indicates that the difference of system throughput using two algorithms is significant. The colorbar is in logarithm scale.

length of matching window). In under-supply scenarios with a long matching window, the restricted subgraph is on par with the throughput produced by the O/D grouping method while having shorter average matching times. In under-supply cases but with shorter matching windows, the restricted subgraph method achieves about 3% better throughput without needing adjustment. The only parameter in the subgraph method is the delay function dictated by perceived customer preference.

- The restricted subgraph method performs typically within 10% of the throughput of the bipartite matching method in under-supply, with lower matching times and lower trip times. Bipartite matching aiming to improve the matching reward inversely stretches people to the limits of their delays. Besides, the computing challenge for the near-optimal bipartite optimization grows exponentially in dense areas.

- Greediness in trip assignments improves the fleet utilization in oversupply scenarios by reducing average pickup times.

In summary, the simulation results demonstrate that the proposed restricted subgraph heuristic can well balance the interests of platforms and passengers. It limits their trip delay and improves their experience without compromising the system's productivity and efficiency.

3.6 Conclusion

In order for the ridesourcing industry to continue to grow, it must acknowledge and work to limit the impact it has on traffic congestion in dense urban areas. An important aspect of this will be the use and encouragement of ride-pooling to serve the same number of trips with fewer vehicles and fewer miles travelled. The restrict subgraph heuristic for ride-pooling assignment proposed in this work is a robust and easily implementable method as a substitute for the centralized optimization methods. It can achieve modest performance when facing a severe scalability issue in dense urban areas.

Future investigations are necessary to address the theoretical foundation of the ride-pooling heuristic methods. In addition, further development on a ride-pooling metaheuristic, i.e., a procedure to select suitable heuristics, is a promising direction. As seen in the numerical results, conditioning on the supply-demand relationship in ridesourcing platforms, the discussed heuristics and baseline single-ride algorithms all have their advantages and disadvantages. Developing a data-driven method that automatically chooses the appropriate methods is a meaningful supplement to the literature.

CHAPTER 4

Efficient Algorithms for Stochastic Ride-pooling Assignment with Mixed Fleets

4.1 Introduction

Ride-pooling assignment aims to dynamically determine the efficient dispatching of vehicles to handle multiple ride requests in a single ride in mobility-on-demand (MoD) systems. It generalizes various fleet management problems in applications ranging from ride-hailing (Santi et al. 2014) to microtransit (Li et al. 2021) and shared autonomous vehicles (Lokhandwala and Cai 2018). Efficient ride-pooling assignment algorithms can enhance the profitability of MoD services and increase the system throughput, i.e., the number of completed customer trips per unit of time (Ke et al. 2021). While consumers may experience trip delays due to detours, they are compensated by splitting the fare with co-riders. More importantly, ride-pooling can decrease dead-heading trips that contribute to excessive energy use and greenhouse gas emissions of MoD platforms (Markov et al. 2021).

One of the MoD platform's central tasks is to achieve a dynamic balance between supply (available vehicles) and demand (pending ride requests). However, this balance is often unattainable due to supply shortages, such as a lack of freelance drivers during peak hours (Guda and Subramanian 2019), the inefficacy of empty-car cruising and searching for customers (Braverman et al. 2019), and drivers' perception errors regarding the supply-demand imbalance (Dong et al. 2021). Contrariwise, the heterogeneity of travel demand and driver types, as well as advancements in vehicle automation, have introduced the notion of "mixed fleet" into MoD platforms, which is illustrated by the following examples:

Example 1: transportation network companies (TNCs) such as Uber and Didi Chuxing cater to diverse market segments by offering various service options. UberX is a standard service operated by freelancers, while Uber Black and Didi Chauffeur are premium services driven by professional drivers. Typically, the platform pairs users with their requested service class. However, when the standard class is in short supply, it may be advantageous for the

platform to reposition premium vehicles to high-demand areas in order to fulfill standard-class ride requests and minimize cancellations.

Example 2: A mixed-autonomy platform operates both fully automated vehicles (AVs) owned by the platform and conventional (human-driven) vehicles (CVs) driven by human freelancers to provide on-demand transit services (see Figure 4.1). When selecting the type of vehicle to dispatch, the operator must consider that (1) customers may prefer to be served by an AV or a CV, depending on their level of trust in automation (Lavieri and Bhat 2019), and (2) the accessible areas and operational costs of AVs and CVs for transporting customers may differ (Shladover 2018, Chen et al. 2017a).

While these examples have distinct contexts, they can be generalized as the following *Stochastic Ride-pooling Assignment with Mixed Fleets (SRAMF)* (SRAMF) problem. The mixed fleets consist of "basis supply" vehicles and "augmented supply" vehicles. Basis supply refers to vehicles operated by freelance drivers who employ self-interested strategies when searching for customers, serve most customers in a decentralized manner, and presumably produce friction in balancing supply and demand (Dong et al. 2023). Augmented supply refers to vehicles (such as AVs) that follow the platform's centralized repositioning policies. Due to the different characteristics of supply sources, the platform faces a tradeoff between cost and control when matching ride requests with available vehicles. For a given level of demand, assigning nearby basis supply vehicles will incur lower operational costs than assigning augmented set vehicles. For example, the platform must pay salaries to full-time drivers in the augmented supply in Example 1 and costly maintenance costs for AVs in Example 2, which will be incorporated into the cost of serving each ride request. On the other hand, the platform may only have the authority to proactively reposition and reassign augmented supply vehicles to complement unsatisfied demand. As such, the platform's decision involves whether and where to reposition augmented supply vehicles, which primarily depends on the consequent assignment between available basis and augmented supply vehicles with realized ride requests.

Two unique operational challenges arise due to the diversification of vehicle fleets on MoD platforms. First, operating MoD with mixed fleets face inherent *uncertainties* in the sequential vehicle repositioning and ride-pooling assignment processes as follows. In the first-stage vehicle repositioning decisions, the platform forecasts future demand and repositions selected premium service vehicles (Example 1) or AVs (Example 2) to specific locations in order to accommodate unmet demands for the basis supply. In the second-stage ride-pooling assignment decisions, the platform assigns realized ride requests to available vehicles, including basis and augmented supply, to maximize the total value of assignments. The uncertainties between the vehicle repositioning and assignment stages can be categorized as supply-based

or demand-based factors. Supply-based uncertainty concerns whether or not basis-supply drivers stay active in future periods. Demand-based uncertainty includes the origins and destinations of upcoming ride requests, the number of passengers per order, customers' unknown preferred vehicle type, and their value of time. Since falsely repositioned vehicles will result in a supply-demand mismatch in the future, joint repositioning and assignment will cause complicated tradeoffs in MoD operations.

Second, previous aggregate vehicle repositioning models are not implementable for vehicle-level operation in MoD systems. The SRAMF problem differs from the large body of mixed-fleet planning literature (Karamanis et al. 2021, Guo et al. 2021) that used an aggregate matching model in region-to-region repositioning flow computations. Focusing on vehicle-level operations under uncertainty will cause significantly more computational burden than the aggregate setting. This *scalability* issue intensifies as the platform uses high-capacity augmented supply vehicles to compensate for their high operational costs, such as on-demand transit services (Hasan and Van Hentenryck 2021). With expanded vehicle capacity, the number of candidate pickup and dropoff routes can grow exponentially. These unique technical challenges of joint repositioning and assignment decisions motivate the development of effective and efficient SRAMF algorithms in this study.

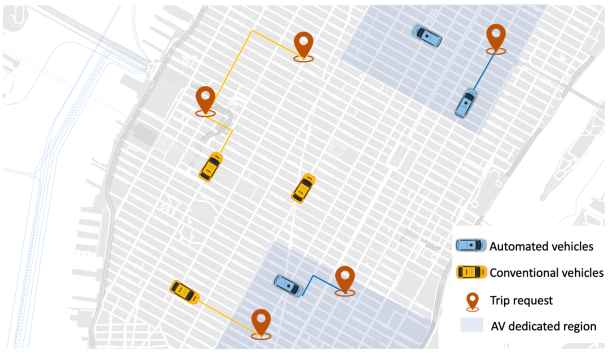


Figure 4.1: Example of ride-pooling with AVs and CVs. The first-stage decision involves repositioning AVs in dedicated regions; the second-stage decision is to solve a Generalized Assignment Problem (GAP).

This study expands the deterministic ride-pooling assignment of homogeneous vehicle fleets in Santi et al. (2014) and Alonso-Mora et al. (2017b) to a stochastic setting in a non-trivial way. The scalable framework addresses the computational challenge of the second-stage problem in SRAMF by separating the vehicle routing and trip-to-vehicle assignment into two sequential steps based on the notion of "shareability graphs". Specifically, given ride requests and available vehicles in each time interval, Alonso-Mora et al. (2017b) proposed a procedure that guaranteed anytime optimality, i.e., the resulting ride-pooling assignments

attain the same solutions as the integrated vehicle routing and trip assignment formulation (see Appendix B.2.1). The procedure is summarized as follows:

1. First, the algorithm constructs a shareability graph that represents the matchable relationship between all ride requests (demand) and available vehicles (supply) (see Figure 4.2) and computes the value associated with each matching.

2. Next, the algorithm maximizes the total matching value by solving a Generalized Assignment Problem (GAP) on the shareability graph.

3. Finally, the shareability graph is updated by deleting occupied vehicles and assigned demand and substituting them with incoming requests and available vehicles.

The SRAMF problem can be formulated as a two-stage stochastic integer program. Incorporating repositioning decisions into the deterministic ride-pooling assignment is difficult due to the relationship between these consecutive steps. Since the vehicle repositioning decision must select augmented supply vehicles by repositioning them from the augmented supply set (a set of candidate locations) before the realization of demand, convoluted tradeoffs must be made between the first- and second-stage decisions. If the platform underestimates demand and selects fewer vehicles, it cannot meet all future requests. If the platform overestimates demand and selects more vehicles than needed, it must pay extra operational costs. In addition, due to various sources of uncertainties stated above, the size of the shareability graph grows rapidly with the number of scenarios sampled. As a result, solving GAP in SRAMF using exact methods becomes inefficient or even infeasible.

4.1.1 Main Results and Contribution

The primary objective of this study is to develop approximation algorithms for solving large-scale SRAMF problems. We focus on the expected value maximization setting for several reasons. First, real-world concerns emphasize the need to enhance MoD systems' throughput and profitability (Ashlagi et al. 2018, Simonetto et al. 2019). Second, devising approximation algorithms for maximizing Generalized Assignment Problems (GAPs) tends to be more challenging than minimizing GAPs (Fleischer et al. 2006). The objective function of SRAMF can incorporate various attributes, such as trip fares, pickup times, and ride-pooling preferences. The primary performance metric for the proposed algorithms is the *tightness* of approximation ratios, offering a provable performance guarantee in worst-case scenarios.

To summarize our work, let p denote the mixed fleets' largest vehicular capacity plus one. Our main results are as follows:

1. The SRAMF problem is proved to be NP-hard for any finite number of scenarios, and its objective lacks attractive *submodular* properties. These characteristics necessitate the development of new approximation algorithms to exploit the computational advantages of shareability graphs.

2. Our analysis provides provable worst-case performance guarantees as follows:

   (a) For mid-capacity vehicles, we develop a local-search linear-program-relaxation (LSLPR) algorithm, with an approximation ratio of $\frac{1}{p^2}$. Mid-capacity vehicles carrying up to four passengers simultaneously are suitable for applications in Example 1.

   (b) For high-capacity vehicles, we develop a max-min online (MMO) algorithm, with an approximation ratio of $\frac{e-1}{(2e+o(1))p\ln p}$. High-capacity vehicles carrying more than four requests are suitable for automated transit services in Example 2.

   (c) These approximation ratios are close to the best possible bounds: no polynomial-time algorithm can achieve a ratio better than $O(\frac{\ln p}{p})$ under standard complexity assumptions.

Our methods rely on a linear relaxation of the second-stage GAP and carefully bound the integrality gap of the relaxation in each scenario. Additionally, this analysis explains the sources of computational intractability of SRAMF and recognizes the significance of considering uncertainties per assignment.

This study contributes to the literature on MoD system operations as follows:

1. Propose a two-stage stochastic integer program for SRAMF and propose approximation algorithms with satisfactory performance guarantees. These easy-to-implement algorithms can facilitate fleet operations on MoD platforms and guarantee their performance in the face of uncertainties with provable bounds.

2. Derive a general estimator for marginal values of trip-to-vehicle matchings. The primary analytical barrier for the design of approximation algorithms for SRAMF is to evaluate the expected value of repositioning additional vehicles to serve future demand in a specific area. Our proof bounds this value and is of independent interest to relevant literature, e.g., fleet sizing in MoD systems (Benjaafar et al. 2021).

3. Provide analytical solutions for fractional hypergraph matchings. Our analysis for the MMO algorithm derives a closed-form solution for the dual problem of fractional hypergraph matchings to accelerate enumerations. This closed-form solution can be transferred to other decomposition-based ride-pooling assignment methods.

We conducted comparative studies to illustrate the computational efficiency and optimality gaps of our developed algorithms using real-world taxicab trip data (TLC 2021). Numerical results showed our algorithms to be almost as competitive as exact methods (mixed-integer program (MIP)), indicating that the derived worst-case approximation ratios are conservative. This framework can incorporate various demand forecasts (Yang et al. 2020) and use state-dependent matching intervals (Qin et al. 2021a). Moreover, our results extend to mixed fleets of more than two vehicle types.

4.1.2 Organization and General Notation

The remainder of the chapter is organized as follows. We first review the related literature in Section 4.2. Section 4.3 formulates the SRAMF problem and shows its hardness. Section 4.4 proposes two approximation algorithms that achieve nearly tight approximation ratios. We test the effectiveness of these approximation algorithms using real-world and simulated data in Section 4.5 and draw conclusions in Section 4.6.

The following notation is used throughout this work. The notation := stands for "defined as". For any integer n, we let $[n] := \{1, 2, \ldots, n\}$. We use $v(\cdot)$ as the actual value function and $\hat{v}(\cdot)$ as the approximate or estimated value function. P stands for the class of questions for which some algorithm can provide an answer in polynomial time, and NP stands for those with nondeterministic polynomial time algorithms. For any set S, $|S|$ is its cardinality. Given two sets A and B, $A + B$ or $A \cup B$ represents the union of A and B; $A - B$ or $A \backslash B$ represents modifying A by removing the elements belonging to B. $A \sim B$ represents that set A intersects with B, i.e., $A \cap B \neq \varnothing$. i.i.d. stands for "independent and identically distributed". Other notation and acronyms used in this chapter are summarized in Table B.1 in Appendix B.1.

4.2 Literature Review

We refer to ride-pooling (also called ride-splitting/carsharing rides) in the broad context and focus on operations-level decisions. Solving the optimal ride-pooling assignment is challenging because the number of possible shared trips grows exponentially in the vehicle capacity and matching intervals. The following review covers the recent development of computational methods for ride-pooling applications with different objectives of maximizing the utilization of vehicles or reducing the negative externalities related to deadhead miles.

Decomposition and approximate dynamic programming approaches. Compared to the substantial body of literature for matching supply and demand without the

ride-pooling option (Wang and Yang 2019), there are only a few attempts to solve the ride-pooling assignment problem at the vehicle level by combining heuristic and decomposition methods (Yu and Shen 2019, Herminghaus 2019, Sundt et al. 2021). Although these heuristics achieved satisfying performance in numerical experiments, they cannot balance computational efficiency and accuracy with theoretical guarantees. The trip planning for ride-pooling is more tractable with fixed travel patterns, such as providing services for daily commuting. Hasan et al. (2020) proposed a commute trip-sharing algorithm that maximized total shared rides for a set of commute trips satisfying various time-window, capacity, pairing, ride duration, and driver constraints.

Another stream of papers emphasized the importance of non-myopic policies in MoD systems, as supply and demand dynamics are influenced by prior decisions. Unfortunately, due to the computational complexity, most nonmyopic ride-pooling assignment policies are restricted to aggregate models and compute optimal flows between regions. Shah et al. (2020) developed an approximate dynamic programming method to learn from the IP-based assignment and approximate the value function by neural networks. We refer readers to a comprehensive review Qin et al. (2021b) of reinforcement learning (RL) methods for ride-sharing assignments and other sequential decisions.

Deterministic ride-pooling assignment for shareability graphs. To tackle those unprecedented computational challenges in MoD systems, Santi et al. (2014) quantified the tradeoff between social benefits and passenger discomfort from ride-pooling by introducing the concept of "shareability networks". They found that the total empty-car travel time was reduced by 40% in the offline setting (i.e., with ex-post demand profiles) or 32% when demand is revealed en route. This work suffers a limitation in vehicle capacity as the matching-based algorithm can only handle up to three-passenger shared rides. Alonso-Mora et al. (2017b) expanded the framework to up to ten riders per vehicle. The high-capacity ride-pooling trip assignment is solved by decomposing the shareability graph into trip sets and vehicle sets and then solving the optimal assignments by a large-scale integer program (IP). As the vehicle capacities increase, the moderate size of the shared vehicle fleet ($2,000$ vehicles with capacities of four rides in their case studies) can serve most travel demands with short waiting times and trip delays. Simonetto et al. (2019) improved this approach's computational efficiency by formulating the master problem as a linear assignment problem. The resulting large-scale assignment on shareability networks is calculated in a distributed manner. However, despite the easy implementation of these methods, they lack theoretical performance guarantees.

Approximation algorithms for maximization GAP. Approximation algorithms can find near-optimal assignments with provable guarantees on the quality of returned solutions.

Since the ride-pooling assignment problem is a variant of GAP (Öncan 2007), we list the significant results here. Shmoys and Tardos (1993) and Chekuri and Khanna (2005) obtained polynomial-time $\frac{1}{2}$-approximation algorithms. Fleischer et al. (2006) obtained an linear program (LP)-rounding based $(1 - \frac{1}{e})$-approximation algorithm and a local-search based $\frac{1}{2}$-approximation algorithm. Previous studies have explored GAP algorithms for both instant and batched dispatching settings. Instant dispatching assigns requests to available vehicles upon arrival. Lowalekar et al. (2020) developed approximation algorithms for online vehicle dispatch systems. Their setting with i.i.d. demand assumptions are markedly different from the current work. Batched dispatching utilizes GAP on a hypergraph to search for locally optimal assignments. Mori and Samaranayake (2021) developed $\frac{1}{e}$-approximation LP-rounding algorithms for the deterministic request-trip-vehicle assignment problem. In contrast, the current work considers a stochastic setting in which sequential repositioning and assignment decisions jointly determined the objective in SRAMF. As a batched dispatching algorithm, this stochastic formulation can be applied to arbitrary demand distributions.

Shared mobility with mixed fleets. Mixed-fleet ridesharing systems are emerging research topics in literature. The first stream of research is motivated by MoD platforms' transition to a blended workforce of permanent employees and freelance workers. Dong and Ibrahim (2020) investigated the staffing problem in which a ride-hailing platform determined the number of fore-hire drivers considering its impact on other flexible workers. Dong et al. (2021) justified the dual-source strategy for mitigating the demand uncertainty in ride-hailing systems and designed optimal contracts to coordinate the mixed workforce. Castro et al. (2020) modeled the ridesharing market as matching queues where drivers had different flexibility levels. They proposed a robust throughput-maximizing capacity reservation policy against the unknown driver engagement function.

The introduction of automation in MoD systems in the foreseeable future motivates a second stream of mixed-fleet research. Lokhandwala and Cai (2018) used agent-based simulations to evaluate the impact of heterogeneous preferences and revealed that the transition to a mixed fleet would reduce the total number of vehicles, focus on areas of dense demands, and lower the overall service levels in the suburban regions. Wei et al. (2020) studied the equilibrium of a mixed autonomy network in which AVs are fully controlled by the platform and CVs are operated by individual drivers. The optimal pricing for the mixed service is formulated as a convex program. Li et al. (2022) proposed a traffic network equilibrium model with mixed autonomy based on two-player games and proved the existence of a speed policy that guarantees Pareto-efficient equilibria. Xie et al. (2023) developed an actor-critic learning approach for mixed-autonomy fleet management considering bounded rational drivers. In contrast, this work is one of the first attempts to develop algorithms for mixed-autonomy

operations at the vehicle level.

4.3 Problem Description

4.3.1 Basic Setting

This section introduces the formulation of the SRAMF problem as a two-stage stochastic integer program and shows its NP-hardness. These technical challenges motivate the design of new approximation algorithms in the remainder of this work.

4.3.1.1 Preliminaries: constructing a shareability graph of mixed fleets.

Ride-pooling assignment is conducted on a shareability graph, represented by a *hypergraph* $G = \{S, D, E\}$. The vertices of the hypergraph are $S \cup D$, where S denotes supply (available vehicles) and D denotes demand (ride requests). Each *hyperedge/clique* $e \in E$ consists of one vehicle and a subset of ride requests. In conventional assignments, each vehicle can serve only one ride request at a time, so G reduces to a bipartite graph. In the ride-pooling setting, each hyperedge $e \in E$ can contain any number of ride requests within the vehicle's capacity. Other constraints, such as the upper bounds for detour times, are considered when constructing the shareability graph (we refer readers to the discussion of shareability graphs in Appendix B.2.). The platform continuously updates such a hypergraph following the procedure outlined in Section 4.1.1. Appendix B.2 also describes a sequence of matching rules that can construct a hierarchical tree of matchable requests, significantly reducing the computational burden of dial-a-ride problems.

This generic model covers most MoD applications described in Section 4.1. The mixed-fleet supply contains a set S_A of locations to reposition augmented vehicles and a set of basis vehicles S_B. We assume that each augmented vehicle can reposition to any of the locations S_A and serve nearby ride requests covered by their incident hyperedges. Let $S = S_A \cup S_B$, $|S_A| = n_A$, and $|S_B| = n_B$. In order to keep notation simple, we will refer to the "locations to reposition augmented vehicles" S_A simply as the augmented supply/vehicles. We denote $p = 1 + \max_{i \in S_A \cup S_B}\{C_i\}$ where C_i is the capacity of vehicle i. Without loss of generality, we let the cost of using vehicles in S_B be 0 and the cost of each vehicle in S_A be normalized to 1. This will be extended to a more general setting of partition constraints in Section 4.4.3. The varying setup costs of S_A and S_B can be justified by the additional operations expenditure of repositioning centralized-controlled vehicles in the augmented set S_A, such as the annualized extra salary paid to full-time drivers in Example 1. Each hyperedge $e = \{i, J\}_{i \in S, J \subseteq D}$ corresponds to a potential trip where vehicle i serves ride requests in J.

The MoD platform will implement SRAMF algorithms using the online procedure outlined below. The platform first predicts available vehicles in S_B and ride requests D per batch, then constructs a shareability graph according to the procedure outlined in Appendix B.2.1. After calculating the value of each hyperedge v_e, the platform solves a two-stage stochastic integer program to determine the optimal centralized repositioning policy for vehicles in S_A. The platform then observes actual demand and vehicle locations and updates the shareability graph. The remainder of this section formally defines the SRAMF problem and highlights its unique technical challenges.

4.3.1.2 Formulation of SRAMF.

Before actual ride requests are sent, the platform chooses a subset $S_R \subset S_A$ of (at most) K locations to reposition vehicles from the augmented supply. After requests are revealed, the platform can assign ride requests only to vehicles in $S_R \cup S_B$ and collect instantaneous rewards (profits) from completing these trips, i.e., reassignment is not allowed.

The sequential decisions for the SRAMF problem are as follows:

1. In the first stage, for each augmented vehicle $i \in S_A$, $y_i = 1$ denotes that an augmented vehicle is allocated to location i for future assignment and $y_i = 0$ denotes not selected. Let $S_R := \{i \in [n_A] : y_i = 1\} \subseteq S_A$ denote a set of selected augmented vehicles. All basis supply vehicles are included, as they impose no additional setup cost, and the available supply in the second stage is $S_R \cup S_B$. The first-stage decision space is $Y \in \{0,1\}^{n_A+n_B}$.

2. In the second stage, a scenario ξ reveals a set of actual ride requests $D(\xi)$ and their associated hyperedges $E(\xi)$. The scenario ξ is assumed to follow a random distribution $F(\xi)$ with support on Ξ, which incorporates a demand forecast model. Each hyperedge $e \in E(\xi)$ includes a vehicle i from either basis or augmented supply and a subset of requests $J \subset D(\xi)$. $\{w_j\}_{j\in J}$ denotes the numbers of passengers in each ride request j. The total number of passengers in a set of ride requests J must satisfy $\sum_{j\in J} w_j \leq C_i$ where C_i is the capacity of vehicle i. The hyperedge value of e may include the following elements:

   (a) The profit u_j gained from serving the ride request j.

   (b) A trip $t = \{j_1, j_2, \cdots : j_k \in J\}$ represents a sequence of picking up ride requests in J. The associated travel cost $c(i,t)$ assumes that the vehicle i follows the shortest pick-up trip to minimize customers' waiting times.

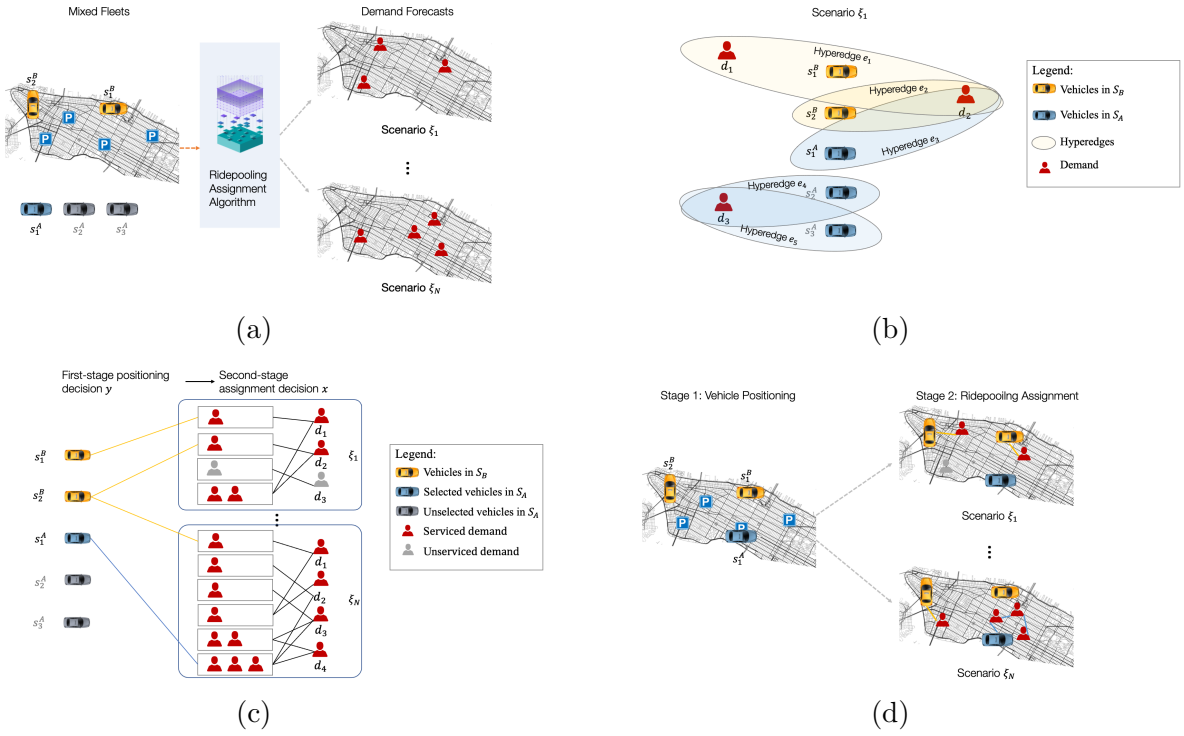


Figure 4.2: The illustration of SRAMF procedure per step. $S_B = \{s_1^B, s_2^B\}$ is the basis set (e.g., CVs) and $S_A = \{s_1^A, s_2^A\}$ is the augmented set (e.g., AVs). Figure 4.2a represents the algorithm's input, including the current locations of S_A and S_B, and obtains demand forecast. Figure 4.2b constructs a shareability graph for each scenario, where each trip is a clique containing one vehicle and multiple matchable requests. Figure 4.2c solves the SRAMF problem by approximation algorithms, in which one or more ride requests are assigned to a selected vehicle in each scenario ξ. Figure 4.2d implements the computed decisions and updates the system state.

(c) Each request j gains additional utility $\tilde{u}_{ij}$ if matched with their preferred vehicle type.

The hyperedge value for $e \in E(\xi)$ collected from a potential assignment is given by

$$v_e = \sum_{j \in J} u_j + \sum_{j \in J} \tilde{u}_{ij} - c(i,t) \geq 0. \tag{1}$$

The hyperedge value captures various sources of uncertainties between vehicle repositioning and trip assignment stages. u_j considers the uncertain number of ride requests and their origin and destination; w_j and the set J considers the unknown number of passengers in each ride request; $\tilde{u}_{ij}$ considers the customers' uncertain preference for vehicle types. Finally, due to fluctuating traffic conditions and different vehicle technol-

ogy (e.g., CVs and AVs), $c(i,t)$ represents that pickup times are uncertain. However, in the second stage (after the scenario ξ is observed), all hyperedge values are known precisely. It is worth mentioning that the calculation of hyperedge values can be integrated with advanced value function approximation techniques. For example, Tang et al. (2019) calculated the associated hyperedge value as a reward signal derived from a reinforcement-learning-based estimator.

3. The platform assigns ride requests to each available vehicle by determining $x_e \in \{0,1\}$ for all $e \in E(\xi)$. The second-stage assignment decision is equivalent to choosing a set of hyperedges in which every pair of hyperedges is disjoint. This condition guarantees that each vehicle and each ride request can be included no more than once in the final assignment per scenario. An assignment is only feasible between the chosen supply $S_R \cup S_B$ (denoted as $e \sim S_R \cup S_B$) and realized demand $D(\xi)$ in each scenario.

The optimal value of assignments in scenario ξ is calculated by $Q : Y \times \Xi \to \mathbb{R}$. Given a scenario, the second-stage decisions are trip assignments denoted by $x = \{x_e\}_{e \in E(\xi)}$. Our objective is to maximize the *expected total value*.

The SRAMF problem can be formulated as a two-stage stochastic integer program:

$$\underset{y}{\text{maximize}}\ \mathbf{E}[Q(y,\xi)] \tag{2}$$

$$s.t. \sum_{i \in S_A} y_i \le K \qquad \textit{(budget)} \tag{2a}$$

$$y_i \in \{0,1\} \qquad \forall i \in S_A \tag{2b}$$

$$y_i = 1 \qquad \forall i \in S_B, \tag{2c}$$

and the second-stage problem is given by

$$Q(y,\xi) = \underset{x}{\text{maximize}} \sum_{e \in E(\xi)} v_e x_e \tag{3}$$

$$s.t. \sum_{e \in E(\xi): j \in e} x_e \le 1 \qquad \forall j \in D(\xi) \quad \textit{(assignment I)} \tag{3a}$$

$$\sum_{e \in E(\xi): i \in e} x_e \le y_i \qquad \forall i \in S_B \cup S_A \quad \textit{(assignment II)} \tag{3b}$$

$$x_e \in \{0,1\} \qquad \forall e \in E(\xi). \tag{3c}$$

In the first-stage problem (2), K is the maximum number of locations for repositioning augmented supply vehicles. In the second-stage problem (3), the constraints (3a) and (3b) guarantee that each supply and demand is matched at most once and the vehicles selected

in $S_R \cup S_B$ are matchable. In other words, unassigned vehicles and ride requests in the hypermatching x will either renege or postpone to the next batch. The second-stage GAP is a p-set packing problem with p representing the maximum size of hyperedges, which is known to be NP-hard (Füredi et al. 1993) and Chan and Lau (2012).

4.3.1.3 Road-map for proving SRAMF approximation algorithms.

Figure 4.3 provides an overview of the performance analysis of two proposed approximation algorithms and their approximation ratios, respectively. We start with reducing the objective of (2) to the sample-average estimate in Section 4.3.2. We then show the hardness of the SRAMF problem in Section 4.3.3. Since the GAP problem (3) is NP-hard, our approximation algorithms rely heavily on the "fractional assignment" technique that relaxes the integrality constraints in (3) as a polynomial-time solvable linear program. Two different approximation algorithms, Local-Search LP-Relaxation (LSLPR) and max-min online (MMO), are discussed in detail in Section 4.4.1 and Section 4.4.2, respectively.

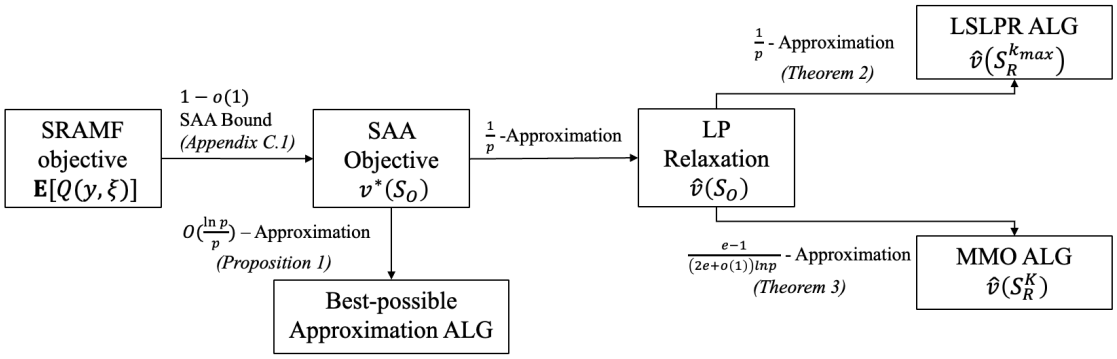


Figure 4.3: Road-map for the performance analysis on SRAMF algorithms; the approximation ratios on arrows refer to the results in this chapter; S_O is the optimal selection of vehicles and S_R is the section of vehicles generated by approximation algorithms.

4.3.2 Reduction to Sample-Average Estimate

The sample-average approximation (SAA) method is commonly used to solve two-stage stochastic integer programs. It draws N scenarios $\{\xi_\ell\}_{\ell=1}^N$ from a scenario-generating oracle (e.g., demand forecasting and vehicle simulation models) and approximates the expected objective function by a sample-average estimate $\mathbf{E}[Q(y,\xi)] \approx \frac{1}{N}\sum_{\ell=1}^N Q(y,\xi_\ell)$.

To simplify the analysis of Problem (2), we reduce the objective function $\mathbf{E}(Q(y,\xi))$ to finite-sample proximity. The main analysis is conditional on N mutually disjoint sets of ride requests $D(\xi)$ and hyperedges $E(\xi)$. Since the second-stage assignment ensures unique

matchings per scenario, we can make multiple disjoint copies when an identical ride request appears in multiple scenarios. The consistency and shrinking bias of the sample-average estimate are well-studied in literature, hence the proof of SAA is detailed in Appendix B.3.1 for completeness. Altogether, the optimal value of any approximation algorithm converges to $\mathbf{E}[Q(y,\xi)]$ as the number of scenarios $N \to \infty$.

This study's focus is therefore developing algorithms to solve the SRAMF problem in (2) with the sample-average estimate. As mentioned earlier, we will work with an LP relaxation of (3) as the original p-set packing problem is NP-hard. For any subset $S_R \subseteq S_A$ and scenario ξ, define $\hat{v}(S_R,\xi)$ to be the optimal value of the following LP:

$$\underset{x}{\text{maximize}} \sum_{e\in E(\xi)} v_e x_e \tag{4}$$

$$s.t. \sum_{e\in E(\xi):j\in e} x_e \le 1 \qquad \forall j \in D(\xi) \tag{4a}$$

$$\sum_{e\in E(\xi):i\in e} x_e \le 1 \qquad \forall i \in S_A \cup S_B \tag{4b}$$

$$x_e = 0 \qquad \forall e \sim S_A \setminus S_R \tag{4c}$$

$$x_e \ge 0 \qquad \forall e \in E(\xi). \tag{4d}$$

Solutions to the LP relaxation of (3) are called *fractional assignments*. $v(S_R,\xi)$ denotes the optimal value of exact solutions to (3), given a set of selected augmented supply S_R. Furthermore, we define two objective functions related to the sample average estimate:

- The objective value using the exact GAP in (3) for each scenario is given by

$$v^*(S_R) = \frac{1}{N}\sum_{\ell\in[N]} v(S_R,\xi_\ell). \tag{5}$$

- The objective value using the LP-relaxation of (4) is given by

$$\hat{v}(S_R) = \frac{1}{N}\sum_{\ell\in[N]} \hat{v}(S_R,\xi_\ell). \tag{6}$$

Fractional assignments of the p-set packing problem enjoy the following properties: (1) The integrality gap between the exact solution and LP relaxation is at most p times (Arkin and Hassin 1998). (2) a greedy algorithm selecting hyperedges e in decreasing order of their values v_e while maintaining feasibility achieves a $\frac{1}{p}$-approximation to the LP value. We restate them in the following theorem:

Theorem 1. *For any $S_R \subseteq S_A$, we have $v^*(S_R) \leq \hat{v}(S_R) \leq p \cdot v^*(S_R)$; furthermore, the greedy algorithm obtains a solution of value at least $\frac{1}{p} \cdot \hat{v}(S_R)$.*

These reductions narrow down the main task of bounding the approximation ratio of $\hat{v}(S_R)$. In particular, we will focus on the SRAMF problem with fractional assignments:

$$\max_{S_R \subseteq S_A : |S_R| \leq K} \quad \hat{v}(S_R). \tag{7}$$

If we obtain an α-approximation algorithm for (7), then combine it with Theorem 1, we would obtain an $\frac{\alpha}{p}$-approximation algorithm for SRAMF (with integral assignments). Also, observe that the objective in (7) is monotone non-decreasing in the selected vehicle set S_R. So, any maximal solution (including the optimal solution) selects exactly K vehicles in the repositioning decision. Before jumping into the design of approximation algorithms, the following section elaborates on some technical challenges.

4.3.3 Hardness and Properties of SRAMF

We show that solving SRAMF is computationally challenging due to the following reasons: (1) Proposition 1 shows that the second-stage assignment problem is NP-hard. Hence, computing the exact assignment for any S_R is costly. (2) Proposition 2 shows that $\hat{v}(S_R)$ is *not* submodular, preventing the use of efficient submodular maximization algorithms. These facts motivate the development of new approximation algorithms in Section 4.4 to exploit the specific structure of the SRAMF problem.

Proposition 1. *There is no algorithm for SRAMF (even with $N = 1$ scenario) with an approximation ratio better than $O(\frac{\ln p}{p})$, unless $P = NP$.*

Proof. Proof for the hardness of SRAMF : We reduce from the *p-dimensional matching* problem, defined as follows. There is a hypergraph H with vertices V partitioned into p parts $\{V_r\}_{r=1}^{p}$, and hyperedges E. Each hyperedge contains exactly one vertex from each part (so each hyperedge's size is p). The goal is to find a collection F of disjoint hyperedges that have maximum cardinality $|F|$.

Given any p-dimensional matching as above, we generate the following SRAMF instance. The augmented vehicles are $S_A = V_1$ and the basis vehicles are $S_B = \varnothing$. There is $N = 1$ scenario with ride requests $V_2 \cup \ldots V_p$ and hyperedges E (each of value 1). Each vehicle has a capacity of $p-1$ and each ride request has one or more passengers. Note that each hyperedge contains precisely one vehicle, as required in SRAMF. The bound $K = |S_A|$ so the optimal first stage solution is clearly $S_R = S_A$ (select *all* locations for augmented vehicles). Now, the

SRAMF problem instance reduces to its second stage problem (3), which involves selecting a maximum cardinality subset of disjoint hyperedges. This is precisely the p-dimensional matching problem.

It follows that if there is any α-approximation algorithm for SRAMF with $N = 1$ scenario then there is an α-approximation algorithm for p-dimensional matching. Finally, Hazan et al. (2006) proved that it is NP-hard to approximate p-dimensional matching better than an $O(\frac{\ln p}{p})$ factor (unless $P = NP$). The proposition now follows. □ □

This intractability is the reason that we work with the *fractional* assignment problem (7). A natural approach for budgeted maximization problems such as (7) is to prove that the objective function is *submodular*, in which case one can directly use the $(1 - \frac{1}{e})$-approximation algorithm by (Nemhauser et al. 1978). However, we show a negative result about the submodularity of $v^*(S_R)$ as well as $\hat{v}(S_R)$, which precludes the use of such an approach. Recall that a set function $f : 2^\Omega \to \mathbb{R}_+$ on groundset Ω is submodular if $f(U \cup \{i\}) - f(U) \geq f(W \cup \{i\}) - f(W)$ for all $U \subseteq W \subseteq \Omega$ and $i \in \Omega \setminus W$.

Proposition 2. *$v^*(S_R)$ and $\hat{v}(S_R)$ are* **not** *submodular functions.*

Proof. Proof: Recall that the ground set for both functions v^* and $\hat{v}$ is $\Omega := S_A$ the set of augmented vehicles. We provide an SRAMF instance with $N = 1$ scenario where these functions are not submodular. Consider a shareability graph with $|S_A| = 3$, $S_B = \varnothing$ and three ride requests $\{d_1, d_2, d_3\}$. Let $p = 3$, i.e., each vehicle can carry at most two requests. The set of hyperedges is

$$\{(s_1^A, d_1), (s_1^A, d_2, d_3), (s_2^A, d_2), (s_3^A, d_3)\}.$$

See also Figure 4.4. The value of each hyperedge reduces to the number of ride requests it covers.

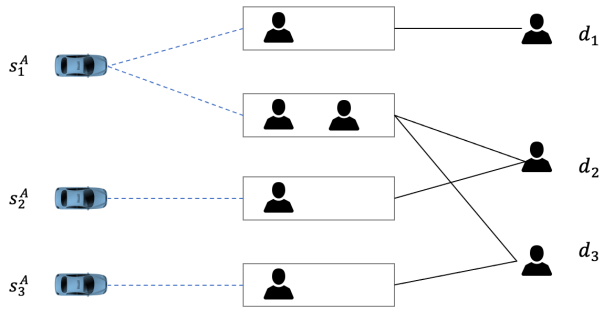


Figure 4.4: An example for non-submodularity of function $v^*(S_R)$.

Let subsets $U = \{s_1^A\}$ and $W = \{s_1^A, s_2^A\}$. Also, let $i = s_3^A$. Clearly, $v^*(U) = 2$ (serving d_2, d_3), $v^*(W) = 2$ (serving d_1, d_2 or d_2, d_3), $v^*(U \cup \{i\}) = 2$ (serving d_1, d_3 or d_2, d_3), and $v^*(W \cup \{i\}) = 3$. Therefore, we have:

$$v^*(W \cup \{i\}) - v^*(W) = 1 > 0 = v^*(U \cup \{i\}) - v^*(U),$$

which implies the set function v^* is not submodular. It is easy to check that the LP value function $\hat{v} = v^*$ for this instance, so function $\hat{v}$ is also not submodular. □ □

4.4 Approximation Algorithms for SRAMF

This section provides two different approximation algorithms for SRAMF. Both the algorithms focus on solving the fractional assignment problem (7), and achieve approximation ratios $\frac{1}{p}$ and $\approx \frac{e-1}{2e \cdot \ln p}$, respectively. Combined with Theorem 1, these imply approximation algorithms for SRAMF with an additional factor of $\frac{1}{p}$.

4.4.1 Local Search Algorithm for Mid-Capacity SRAMF

The Mid-Capacity SRAMF models the current ride-hailing market, in which each vehicle can deliver two to four ride requests simultaneously. In this section, we propose an Local-Search LP-Relaxation (LSLPR) algorithm that obtains $\frac{1}{p}$-approximation for the fractional assignment problem (7).

4.4.1.1 Overview of the LSLPR algorithm.

Let $\epsilon > 0$ be an arbitrarily small parameter to serve as the algorithm's stopping criterion. The outline of the LSLPR algorithm is as follows:

1. Start from any solution $S_R \subseteq S_A$ with $|S_R| = K$.
2. Consider all alternative solutions $S_{R'} = S_R - \{i\} + \{i'\}$ where $i \in S_R$ and $i' \notin S_R$ after swapping one vehicle and evaluate the corresponding LP value $\hat{v}(S_{R'})$.
3. Change the current solution S_R to $S_{R'}$ if the objective value improves significantly, i.e., $\hat{v}(S_{R'}) > (1 + \epsilon) \cdot \hat{v}(S_R)$.
4. Stop if such a significant local swap does not exist.

Formally, let k index the iterations. Let S_R^k denote the current solution in iteration k. The following subroutine implements a single iteration.

Algorithm 3: Local swap subroutine.

for $i \in S_R^k$ and $i' \in S_A \backslash S_R^k$ **do**
  obtain $\hat{v}(S_R^k - i + i')$ by solving the fractional assignment problem;
end
let (c, c') be the pair that maximizes $\hat{v}(S_R^k - i + i')$ over $i \in S_R^k$ and $i' \in S_A \backslash S_R^k$;
if $\hat{v}^*(S_R^k - c + c') > (1+\epsilon) \cdot \hat{v}(S_R^k)$ **then**
  set $S_R^{k+1} \leftarrow S_R^k - c + c'$ and continue with $k \leftarrow k+1$
;
else
  halt local search and output S_R^k;
end

In a broad sense, the local swap subroutine does not necessarily enumerate all pairs (i, i') to search for the optimal (c, c'). A more efficient alternative is terminating each iteration at the first pair of $i \in S_R$ and $i' \in S_A \backslash S_R$ that increases the objective by more than $\epsilon \cdot \hat{v}(S_R^k)$.

The complete LSLPR algorithm is as follows:

Algorithm 4: The LSLPR algorithm for mid-capacity SRAMF

Data: Augmented supply S_A, basis supply S_B, scenarios $\{\xi_\ell\}_{\ell=1}^N$ and $\epsilon > 0$.
Result: Near-optimal $S_R \subset S_A$ and the corresponding trip assignment.
Initialization: Set $k = 1$ and randomly select K vehicles from S_A as S_R^1;
while $k \leq k_{\max}$ **do**
  Run the local swap subroutine in Algorithm 1;
  Obtain the final trip assignment with $S_R = S_R^{k_{\max}}$ using the greedy algorithm (Theorem 1).
end

Algorithm 4 obtains the final selection of vehicles $S_R^{k_{\max}}$ where the maximal number of iterations k_{max} will be derived below. In the final step, the algorithm obtains an *integral* assignment for each scenario instead of the fractional assignments. To this end, we can use the greedy algorithm (see Theorem 1) to select the assignment for each scenario, which is guaranteed to have an objective value at least $\frac{1}{p}$ times the fractional assignment. In Section 4.4.1.2, we first analyze the approximation ratio and then the computational complexity of LSLPR.

4.4.1.2 Analysis of the LSLPR algorithm.

Recall that S_R is the solution obtained by our algorithm and $|S_R| = K$. Let S_O denote the optimal solution: we assume (without loss of generality) $|S_O| = K$. Note that S_O is a fixed subset only used in the analysis. Also, let $\boldsymbol{x} = \langle \boldsymbol{x}^\xi \rangle$ and $\boldsymbol{z} = \langle \boldsymbol{z}^\xi \rangle$ denote the optimal LP solutions to $\hat{v}(S_R)$ and $\hat{v}(S_O)$, respectively.

It will be convenient to consider the overall hypergraph on vertices $S_A \cup S_B \cup (\cup_\xi D(\xi))$ and hyperedges $\cup_\xi E(\xi)$. As the objective $\hat{v}(\cdot)$ is additive over the scenarios ξ, we may assume, by duplicating demands and hyperedges (if necessary), that demands $D(\xi)$ and hyperedges $E(\xi)$ are disjoint across scenarios ξ. Recall that $\boldsymbol{x}^\xi$ (and $\boldsymbol{z}^\xi$) has a decision variable corresponding to each hyperedge in $E(\xi)$. For each demand $d \in \cup_\xi D(\xi)$, let H_d denote the hyperedges incident to it. For each vehicle $i \in S_A \cup S_B$ and scenario ξ, let $E_{i,\xi}$ denote the hyperedges in $E(\xi)$ containing i. So, $F_i := \cup_\xi E_{i,\xi}$ is the set of hyperedges incident to vehicle i.

For any demand d, the following lemma sets up a mapping between the hyperedges (incident to d) used in the solutions $\boldsymbol{x}$ and $\boldsymbol{z}$. For the analysis, we add a dummy hyperedge $\perp$ incident to d so that the assignment constraints in the LP solutions $\boldsymbol{x}$ and $\boldsymbol{z}$ are binding at d. So, $\sum_{e \in H_d} x_e + x_\perp = 1$ and $\sum_{f \in H_d} z_f + z_\perp = 1$. Let $H'_d := H_d \cup \{\perp\}$ denote the hyperedges incident to d.

Lemma 1. *For any demand d, there exists a decomposition mapping $\Delta_d : H'_d \times H'_d \to \mathbb{R}$ satisfying the following conditions:*

1. $\Delta_d(e, f) \geq 0$ *for all* $e, f \in H'_d$;
2. $\sum_{e \in H'_d} \Delta_d(e, f) = z_f$ *for all* $f \in H'_d$;
3. $\sum_{f \in H'_d} \Delta_d(e, f) = x_e$ *for all* $e \in H'_d$.

Figure 4.5 illustrates this mapping. Appendix B.3 includes the definition of $\Delta_d(e, f)$ and the proof of Lemma 1. Note that $\sum_{e \in H'_d} \sum_{f \in H'_d} \Delta_d(e, f) = 1$ for any demand d. For any subset $F \in H'_d$, we use the shorthand $\Delta_d(e, F) := \sum_{f \in F} \Delta_d(e, f)$ and $\Delta_d(F, e) := \sum_{f \in F} \Delta_d(f, e)$.

Here is an outline of the remaining analysis. Let $\mathcal{L}$ denote a bijection between S_R (LSLPR algorithm's solution) and S_O (optimal solution), consisting of pairs (i_1, i_2) where $i_1 \in S_R$ and $i_2 \in S_O$. We also ensure that $\mathcal{L}$ contains the pairs (i, i) for all vehicles $i \in S_R \cap S_O$. Note that there is such a bijection because $|S_R| = K = |S_O|$. We first consider a swap $S_R - \{i_1\} + \{i_2\}$ where $(i_1, i_2) \in \mathcal{L}$, and lower bound the objective increase. Note that the approximate local optimality of S_R implies that the objective increase is at most $\epsilon \cdot \hat{v}(S_R)$. Then, we add the inequalities corresponding to the objective increase for the swaps in $\mathcal{L}$ and obtain the approximation ratio.

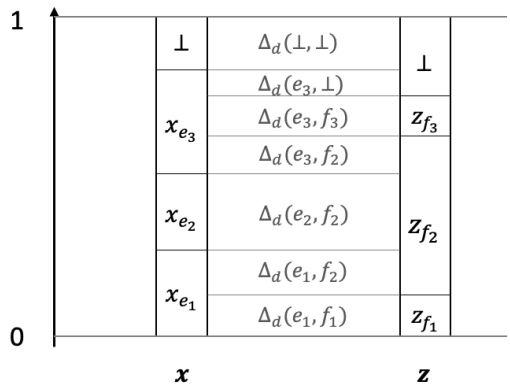


Figure 4.5: Illustration of mapping $\Delta_d(e, f)$ with $E(\xi) = \{e_1, e_2, e_3\}$.

Analysis of a single swap (i_1, i_2). Consider any $i_1 \in S_R$ and $i_2 \in S_O$. We now lower bound $\hat{v}(S_R - \{i_1\} + \{i_2\}) - \hat{v}(S_R)$. Recall that for any subset S, $\hat{v}(S) = \frac{1}{N}\sum_\xi \hat{v}(S, \xi)$ where $\hat{v}(S, \xi)$ is the LP value for scenario ξ. So, we have

$$\hat{v}(S_R - \{i_1\} + \{i_2\}) - \hat{v}(S_R) = \frac{1}{N}\sum_\xi \left(\hat{v}(S_R - \{i_1\} + \{i_2\}, \xi) - \hat{v}(S_R, \xi)\right).$$

We now focus on a single scenario ξ and lower bound $\hat{v}(S_R - \{i_1\} + \{i_2\}, \xi) - \hat{v}(S_R, \xi)$. $\bar{\boldsymbol{x}}^\xi$ represents a feasible solution for fractional assignment $\hat{v}(S_R - \{i_1\} + \{i_2\}, \xi)$. Recall that $\boldsymbol{x}^\xi$ denotes the optimal solution for LP $\hat{v}(S_R, \xi)$. So, we can then bound:

$$\hat{v}(S_R - \{i_1\} + \{i_2\}, \xi) - \hat{v}(S_R, \xi) \quad \geq \quad \boldsymbol{v}^\intercal \bar{\boldsymbol{x}}^\xi - \boldsymbol{v}^\intercal \boldsymbol{x}^\xi, \tag{8}$$

where $\boldsymbol{v}$ is the vector of hyperedge values for $E(\xi)$. As we focus on a single scenario ξ, we drop ξ from the notation whenever it is clear.

We are now ready to construct the new fractional assignment $\bar{\boldsymbol{x}}$. Define:

1. $\bar{x}_e = 0$ for all $e \in F_{i_1}$. This corresponds to dropping vehicle i_1 from S_R.
2. $\bar{x}_e = z_e$ for all $e \in F_{i_2}$. This corresponds to adding vehicle i_2 to S_R.
3. $\bar{x}_e = x_e - \max_{d \in e} \Delta_d(e, F_{i_2} \cap H_d)$ for all $e \in E(\xi) \setminus F_{i_1} \setminus F_{i_2}$.

If $i_1 = i_2$, then we drop case 1 above. The third case above is needed to make space for the hyperedges incident to the new vehicle i_2 (which is increased in case 2). The following two lemmas prove the feasibility of this solution $\bar{\boldsymbol{x}}$ and bound its objective value. Below, we assume that $i_1 \neq i_2$ (the proof for $i_1 = i_2$ is nearly the same, in fact even simpler). So, $i_1 \in S_R \setminus S_O$ and $i_2 \in S_O \setminus S_R$.

Lemma 2. *$\bar{\boldsymbol{x}}$ is a feasible solution for $\hat{v}(S_R - \{i_1\} + \{i_2\})$.*

Proof. Proof for Lemma 2:

We show the feasibility by checking all constraints in (4). Note that $\bar{x}_e = 0$ for all hyperedges e incident to a vehicle in $S_A \setminus (S_R - \{i_1\} + \{i_2\})$.

Constraint $\bar{\boldsymbol{x}} \geq 0$. It suffices to check this for hyperedges $e \in E \setminus F_{i_1} \setminus F_{i_2}$. Note that

$$\bar{x}_e = x_e - \max_{d \in e} \Delta_d(e, F_{i_2} \cap H_d) = \min_{d \in e} \left(x_e - \Delta_d(e, F_{i_2} \cap H_d)\right) \geq 0,$$

where the inequality uses Lemma 1 (condition 3), i.e., $x_e = \Delta_d(e, H'_d) \geq \Delta_d(e, F_{i_2} \cap H_d)$.

Constraint (4a): By definition of $\bar{\boldsymbol{x}}$, for any demand d, we have:

$$\begin{aligned}
\sum_{e \in H_d} \bar{x}_e &\leq \sum_{e \in H_d \cap F_{i_2}} z_e + \sum_{e \in H_d \setminus F_{i_1} \setminus F_{i_2}} [x_e - \Delta_d(e, F_{i_2} \cap H_d)] \\
&\leq \sum_{e \in H_d \cap F_{i_2}} z_e + \sum_{e \in H'_d} [x_e - \Delta_d(e, F_{i_2} \cap H_d)] \qquad (9) \\
&= \sum_{e \in H_d \cap F_{i_2}} z_e + \sum_{e \in H'_d} x_e - \sum_{e \in H'_d} \Delta_d(e, F_{i_2} \cap H_d) \\
&= \sum_{e \in H_d \cap F_{i_2}} z_e + \sum_{e \in H'_d} x_e - \sum_{f \in F_{i_2} \cap H_d} \Delta_d(H'_d, f) \\
&= \sum_{e \in H'_d} x_e \;=\; 1. \qquad (10)
\end{aligned}$$

Above, (9) uses $x_e \geq \Delta_d(e, F_{i_2} \cap H_d)$ by Lemma 1, and the first equality in (10) uses $z_f = \Delta_d(H'_d, f)$ by Lemma 1 (condition 2).

Constraint (4b): The augmented vehicle set can be divided into three groups.

1. Vehicle i_1: $\sum_{e \in F_{i_1}} \bar{x}_e = 0$.
2. Vehicle i_2: $\sum_{e \in F_{i_2}} \bar{x}_e = \sum_{e \in F_{i_2}} z_e \leq 1$ by definition.
3. Vehicles $j \neq i_1, i_2$: $\sum_{e \in F_j} \bar{x}_e \leq \sum_{e \in F_j} x_e \leq 1$. Here, we used the definition of $\bar{x}_e$ and $\Delta_d(\cdot, \cdot) \geq 0$ by Lemma 1 (condition 1).

Therefore, $\bar{\boldsymbol{x}}$ is a feasible fractional assignment solution. □ □

Lemma 3. *The increase in the objective is:*

$$\sum_{e \in E(\xi)} v_e(\bar{x}^\xi_e - x^\xi_e) \geq \sum_{e \in F_{i_2} \cap E(\xi)} v_e z^\xi_e - \sum_{f \in F_{i_1} \cap E(\xi)} v_f x^\xi_f - \sum_{e \in E(\xi)} v_e \sum_{d \in e} \Delta_d(e, F_{i_2} \cap H_d).$$

Proof. Proof for Lemma 3: By definition of $\bar{\boldsymbol{x}}$,

$$\bar{x}_e - x_e = \begin{cases} z_e & \text{if } e \in F_{i_2} \\ -x_e & \text{if } e \in F_{i_1} \\ -\max_{d \in e} \Delta_d(e, F_{i_2} \cap H_d) & \text{otherwise} \end{cases}.$$

Above, we used that $x_e = 0$ for all $e \in F_{i_2}$ as $i_2 \in S_A \backslash S_R$. So, we have

$$\begin{aligned} \sum_{e \in E(\xi)} v_e(\bar{x}_e - x_e) &\geq \sum_{e \in F_{i_2} \cap E(\xi)} v_e z_e - \sum_{f \in F_{i_1} \cap E(\xi)} v_f x_f - \sum_{e \in E(\xi)} v_e \max_{d \in e} \Delta_d(e, F_{i_2} \cap H_d) \\ &\geq \sum_{e \in F_{i_2} \cap E(\xi)} v_e z_e - \sum_{f \in F_{i_1} \cap E(\xi)} v_f x_f - \sum_{e \in E(\xi)} v_e \sum_{d \in e} \Delta_d(e, F_{i_2} \cap H_d). \end{aligned}$$

□ □

Combining Lemmas 2 and 3, and adding over scenarios ξ, we obtain:

Lemma 4. *For any pair $(i_1, i_2) \in \mathcal{L}$, we have*

$$\hat{v}(S_R - \{i_1\} + \{i_2\}) - \hat{v}(S_R) \geq \sum_{e \in F_{i_2}} v_e z_e - \sum_{f \in F_{i_1}} v_f x_f - \sum_{e \in E} v_e \sum_{d \in e} \Delta_d(e, F_{i_2} \cap H_d).$$

Combining all the swaps. Recall that $\mathcal{L}$ is a bijection between S_R and S_O, so $|\mathcal{L}| = K$. Moreover, using the local-search termination condition, there is no swap that improves the objective of the final solution S_R by more than $\epsilon \cdot \hat{v}(S_R)$. Hence,

$$\begin{aligned} K\epsilon \cdot \hat{v}(S_R) &\geq \sum_{(i_1,i_2) \in \mathcal{L}} [\hat{v}(S_R - \{i_1\} + \{i_2\}) - \hat{v}(S_R)] \\ &\geq \sum_{(i_1,i_2) \in \mathcal{L}} \left[\sum_{e \in F_{i_2}} v_e z_e - \sum_{f \in F_{i_1}} v_f x_f - \sum_{e \in E} v_e \sum_{d \in e} \Delta_d(e, F_{i_2} \cap H_d) \right] \qquad (11) \\ &= \sum_{i_2 \in S_O} \sum_{e \in F_{i_2}} v_e z_e - \sum_{i_1 \in S_R} \sum_{f \in F_{i_1}} v_f x_f - \sum_{i_2 \in S_O} \sum_{e \in E} v_e \sum_{d \in e} \Delta_d(e, F_{i_2} \cap H_d) \\ &\geq \sum_{i_2 \in S_O} \sum_{e \in F_{i_2}} v_e z_e - \sum_{i_1 \in S_R} \sum_{f \in F_{i_1}} v_f x_f - \sum_{e \in E} v_e \sum_{d \in e} \Delta_d(e, H_d) \qquad (12) \\ &\geq \sum_{i_2 \in S_O} \sum_{e \in F_{i_2}} v_e z_e - \sum_{i_1 \in S_R} \sum_{f \in F_{i_1}} v_f x_f - \sum_{e \in E} v_e \sum_{d \in e} x_e \qquad (13) \\ &= v^T z - v^T x - \sum_{e \in E} |\{d \in e\}| v_e x_e \\ &\geq v^T z - v^T x - (p-1) v^T x \;=\; v^T z - p \cdot v^T x \;=\; \hat{v}(S_O) - p \cdot \hat{v}(S_R). \qquad (14) \end{aligned}$$

Above, (11) is by Lemma 4, (12) uses that $\{F_{i_2}\}_{i_2 \in S_O}$ are disjoint, (13) uses Lemma 1, and the inequality in (14) uses that each hyperedge has at most $p-1$ demands.

Setting $\epsilon = \frac{1}{pK^2}$, it follows that $\hat{v}(S_R) \geq \frac{1}{p+o(1)} \cdot \hat{v}(S_O)$. Combined with Theorem 1, we obtain $v^*(S_R) \geq \frac{1}{p} \cdot \hat{v}(S_R) \geq \frac{1}{p^2+o(p)} \cdot \hat{v}(S_O)$. Hence,

Theorem 2. *The LSLPR algorithm for SRAMF is a $\frac{1}{p^2}$-approximation algorithm.*

Time complexity of the LSLPR algorithm. Note that each iteration of Algorithm 3 involves considering $K(n_A - K)$ potential swaps and recall that $n_A = |S_A|$. For each swap, we need to evaluate $\hat{v}$, which can be done using any polynomial time LP algorithm such as the ellipsoid method (Bertsimas and Tsitsiklis 1997). So, the time taken in each iteration is polynomial.

We now bound the number of local search iterations. In each iteration, the objective value increases by a factor of at least $1+\epsilon$. So, after k iterations,

$$\hat{v}(S_R^{k+1}) \geq (1+\epsilon)^k \hat{v}(S_R^1).$$

Clearly, the assignment associated with the initial solution S_R^0 has a lower bound $\hat{v}(S_R^0) \geq \frac{1}{N} \cdot v_{\min}$ where $v_{\min} = \min_{e:v_e>0} v_e$ is the minimum value over all hyperedges. Recall that hyperedges with nonpositive values are not considered in any assignment. The maximum objective of any solution is at most $(n_A + n_B) \cdot v_{\max}$ where $|S_A| = n_A$, $|S_B| = n_B$ and $v_{\max} = \max_e v_e$ is the maximum value over all hyperedges. Hence,

$$(n_A + n_B) \cdot v_{\max} \geq \hat{v}(S_R^{k+1}) \geq (1+\epsilon)^k \cdot \frac{1}{N} v_{\min},$$

which implies that the maximum number of iterations

$$k_{\max} \leq \log_{1+\epsilon} \left(\frac{N(n_A+n_B)v_{\max}}{v_{\min}} \right) = O\left(\frac{1}{\epsilon} \log \frac{N(n_A+n_B)v_{\max}}{v_{\min}} \right).$$

Using $\epsilon = \frac{1}{pK^2}$, it follows that the number of iterations is polynomial.

The last step of Algorithm 4 implements the greedy p-set packing algorithm for each scenario, which also takes polynomial time. It follows that LSLPR solves the SRAMF problem in polynomial time regarding parameters p, K, N, $|E|$, n_A, and n_B.

4.4.2 Max-Min Online Algorithm for High-Capacity SRAMF

The LSLPR algorithm is capable of assigning rides in shared mobility applications using mid-capacity vehicles. When the maximal capacity of vehicles in MoD is large (e.g., the maximum

capacity of MoD transit service is ten in (Alonso-Mora et al. 2017b)), the $\frac{1}{p^2}$-approximation ratio is disadvantageous. We propose an alternative method for high-capacity SRAMF. The main idea of the max-min online (MMO) algorithm is to use LP-duality to reformulate $\hat{v}$ as a *covering* linear program. Then, the max-min optimization in Feige et al. (2007) can further improve the approximation ratio. This framework requires two technical properties (monotonicity and online competitiveness), which are satisfied in the SRAMF problem. We will prove that the MMO algorithm obtains an approximation ratio of $(1-\frac{1}{e})\frac{1}{2p\ln p}$.

Using LP-duality and the definition of $\hat{v}(S_R)$ (see the derivation in Appendix B.3.3), we can reformulate:

$$\begin{aligned} \hat{v}(S_R) \;=\; & \underset{u}{\text{minimize}} \sum_{\xi}\sum_{g\in G} u_{g,\xi} \\ & s.t. \sum_{g\in e} u_{g,\xi} \geq \frac{v_e}{N}, \qquad\qquad \forall e\in F_{i,\xi}, \quad \forall \xi, \quad \forall i \in S_R \cup S_B \\ & \boldsymbol{u} \geq 0. \end{aligned} \tag{15}$$

Here, $G = S_A \cup S_B \cup (\cup_\xi D(\xi))$ is a combined groundset consisting of all vehicles and demands from all scenarios. For any vehicle i and scenario ξ, set $F_{i,\xi} \subseteq E(\xi)$ denotes all the hyperedges incident to i in scenario ξ.

We can scale the covering constraints to normalize the right-hand-side to 1 and rewrite the constraints as $\sum_{g\in e} \frac{N}{v_e} u_{g,\xi} \geq 1$. Note that the *row sparsity* of this constraint matrix (i.e., the maximum number of non-zero entries in any constraint) is $\max_{e\in E}|e| = p$ and $v_e > 0$ for all hyperedges. Let $\boldsymbol{c}_e$ be the row of constraint coefficients for any hyperedge $e \in E = \cup_\xi E(\xi)$, i.e.,

$$c_e(g,\xi) = \begin{cases} \frac{N}{v_e} & \text{if } g \in e \text{ and } e \in E(\xi) \\ 0 & \text{otherwise} \end{cases}.$$

Then, the SRAMF problem with fractional assignments $\max_{S_R\subseteq S_A:|S_R|\leq K} \hat{v}(S_R)$ can be treated as the following max-min problem:

$$\max_{S_R\subseteq S_A:|S_R|\leq K} \quad \min_{u}\{\mathbf{1}^\intercal \boldsymbol{u} \mid \boldsymbol{c}_e^\intercal \boldsymbol{u} \geq 1,\ \forall e\in F_i,\ \forall i \in S_R\cup S_B;\ \boldsymbol{u}\geq 0\}, \tag{16}$$

where $F_i = \cup_\xi F_{i,\xi}$ for each vehicle i. For the remainder, $t = 1, 2, \ldots$ indexes steps of the online algorithm.

The main result is:

Theorem 3. *There is a* $\frac{e-1}{(2e+o(1))\ln p}$*-approximation algorithm for* (16).

Before proving this result, we introduce two important properties.

Definition 1. *(Competitive online property) An α-competitive online algorithm for the covering problem* (15) *takes as input any sequence $(i_1, i_2, \ldots, i_t, \ldots)$ of vehicles from S_A and maintains a non-decreasing solution $\boldsymbol{u}$ such that the following hold for all steps t.*

- *$\boldsymbol{u}$ satisfies constraints $\boldsymbol{c}_e^\intercal \boldsymbol{u} \geq 1$ for $e \in F_i$, for all vehicles $i \in \{i_1, i_2, \ldots, i_t\}$, and*
- *$\boldsymbol{u}$ is an α-approximate solution, i.e., the objective $\mathbf{1}^\intercal \boldsymbol{u} \leq \alpha \cdot \hat{v}(\{i_1, i_2, \ldots, i_t\})$.*

Note that the online algorithm may only increase variables $\boldsymbol{u}$ in each step t.

Definition 2. *(Monotone property) For any $\boldsymbol{u} \geq 0$ and $S \subseteq S_A$, let*

$$Aug^*(S|\boldsymbol{u}) := \{\min_{\boldsymbol{w} \geq 0} \mathbf{1}^\intercal \boldsymbol{w} : \boldsymbol{c}_e^\intercal(\boldsymbol{u} + \boldsymbol{w}) \geq 1, \forall e \in F_i, \forall i \in S \cup S_B\}.$$

The covering problem (15) *is said to be monotone if for any $\boldsymbol{u} \geq \boldsymbol{u}' \geq 0$ (coordinate wise) and any $S \subseteq S_A$, $Aug^*(S|\boldsymbol{u}) \leq Aug^*(S|\boldsymbol{u}')$.*

These properties were used by Feige et al. (2007) to show the following result.

Theorem 4. *(Feige et al. 2007) If the covering problem* (15) *satisfies the monotone and α-competitive online properties, there is a $\frac{e-1}{e \cdot \alpha}$-approximation for the max-min problem in* (16).

Our max-min problem indeed satisfies both these properties.

Lemma 5. *The covering problem* (15) *has an $\alpha = O(\ln p)$ competitive online algorithm. Moreover, when p is large, the factor $\alpha = (2 + o(1)) \ln p$.*

Proof. Proof: Recall that (15) is a covering LP with row-sparsity p. Moreover, in the online setting, constraints to (15) arrive over time. So, this is an instance of online covering LPs, for which an $O(\ln p)$-competitive algorithm is known (Gupta and Nagarajan 2014). See also Buchbinder et al. (2014) for simpler proof. Moreover, one can optimize the constant factor in Buchbinder et al. (2014) to get $\alpha = (2 + o(1)) \ln p$.

We note that these previous papers work with the online model when only one covering constraint arrives in each step. Although Lemma 5 involves multiple covering constraints F_i arriving in each step, this complexity can easily be reduced to the prior setting as follows. We introduce the constraints in F_i one-by-one in any order. The algorithms in (Gupta and Nagarajan 2014, Buchbinder et al. 2014) can therefore be used directly. □ □

Lemma 6. *The covering problem* (15) *is monotone.*

Proof. Proof: Consider any $\boldsymbol{u} \geq \boldsymbol{u}' \geq 0$ and any $S \subset S_A$. Let $\boldsymbol{w}' \geq 0$ denote an optimal solution to $Aug^*(S_R|\boldsymbol{u}')$. As all constraint-coefficients $\boldsymbol{c}_e \geq 0$, it follows that $\boldsymbol{c}_e^\intercal(\boldsymbol{u}+\boldsymbol{w}') \geq \boldsymbol{c}_e^\intercal(\boldsymbol{u}'+\boldsymbol{w}') \geq 1$ for all $e \in F_i$ and $i \in S \cup S_B$. Hence, $\boldsymbol{w}'$ is also a feasible for the constraints in $Aug^*(S|\boldsymbol{u})$. Therefore, $Aug^*(S|\boldsymbol{u}) \leq \mathbf{1}^\intercal \boldsymbol{w}' = Aug^*(S|\boldsymbol{u}')$, which proves the monotonicity. □ □

Combining Lemmas 5 and 6 with Theorem 4, we obtain Theorem 3. We note that our $\Omega(\frac{1}{\ln p})$ approximation ratio is nearly the best possible for the max-min problem (16), as the problem is hard to approximate to a factor better than $O(\frac{\ln \ln p}{\ln p})$; see Feige et al. (2007).

We now describe the complete algorithm for SRAMF below. This is a combination of the online LP algorithm from Buchbinder et al. (2014) and the max-min algorithm from Feige et al. (2007). For any ordered subset S of vehicles, let $\hat{v}_{ON}(S)$ denote the objective value of the online algorithm for (15) after adding constraints corresponding to the vehicles in S (in that order). Algorithm 5 describes the updates performed by the online algorithm when a vehicle i is added.

Algorithm 5: Updating subroutine in Max-Min Online algorithm

For a given $i \in S_A \cup S_B$, perform the following updates;

for $e \in F_i = \bigcup_\xi F_{i,\xi}$ **do**

let $\{u^-_{g,\xi}\}_{g \in e}$ be the values of variables in hyperedge e and $\Gamma^-_e = \sum_{g \in e} u^-_{g,\xi}$;

if $\Gamma^-_e < \frac{v_e}{N}$ **then**

$$\text{update } u_{g,\xi} \leftarrow \left(u^-_{g,\xi} + \frac{v_e}{N}\delta\right) \cdot \frac{1+|e|\cdot\delta}{\frac{N}{v_e}\Gamma^-_e + |e|\cdot\delta} - \frac{v_e}{N}\delta, \quad \text{for all } g \in e.$$

end

end

Proof. Proof for the updating subroutine in MMO algorithm: Consider the updates when vehicle i is added. Consider any scenario ξ and hyperedge $e \in F_{i,\xi}$: the corresponding covering constraint is $c_e^T u = \frac{N}{v_e}\sum_{g \in e} u_{g,\xi} \geq 1$. Let τ be a continuous variable denoting time and $\delta > 0$ be a constant. The online LP algorithm in (Buchbinder et al. 2014) raises variables $u_{g,\xi}$ in a continuous manner as follows:

$$\frac{\partial u_{g,\xi}}{\partial \tau} = \frac{N}{v_e} u_{g,\xi} + \delta, \qquad \forall g \in e, \tag{17}$$

until the constraint is satisfied. Letting $\Gamma_e = \sum_{g \in e} u_{g,\xi}$, we have

$$\frac{\partial \Gamma_e}{\partial \tau} = \frac{N}{v_e} \sum_{g \in e} u_{g,\xi} + |e| \cdot \delta = \frac{N}{v_e} \Gamma_e + |e| \cdot \delta.$$

By integrating, it follows that the duration of this update is

$$T = \int_{\Gamma = \Gamma_e^-}^{\Gamma_e^+} \frac{\partial \Gamma_e}{\frac{N}{v_e}\Gamma_e + |e| \cdot \delta} = \frac{v_e}{N} \cdot \ln \left(\frac{\frac{N}{v_e}\Gamma_e^+ + |e| \cdot \delta}{\frac{N}{v_e}\Gamma_e^- + |e| \cdot \delta} \right) = \frac{v_e}{N} \cdot \ln \left(\frac{1 + |e| \cdot \delta}{\frac{N}{v_e}\Gamma_e^- + |e| \cdot \delta} \right).$$

Above Γ_e^- and Γ_e^+ denote the values of Γ_e at the start and end of this update step; note that $\Gamma_e^+ = v_e/N$ as the updates stop as soon as the constraint is satisfied. For each $g \in e$, using (17),

$$T = \int_{\tau=0}^{T} \frac{\partial u_{g,\xi}}{\frac{N}{v_e} u_{g,\xi} + \delta} = \frac{v_e}{N} \cdot \ln \left(\frac{\frac{N}{v_e} u_{g,\xi}^+ + \delta}{\frac{N}{v_e} u_{g,\xi}^- + \delta} \right).$$

Again, $u_{g,\xi}^-$ and $u_{g,\xi}^+$ denote the values of $u_{g,\xi}$ at the start and end of this update step. Combined with the above value for T, we get a closed-form expression for the new variable values:

$$\frac{N}{v_e} u_{g,\xi}^+ + \delta = \left(\frac{N}{v_e} u_{g,\xi}^- + \delta \right) \cdot \frac{1 + |e| \cdot \delta}{\frac{N}{v_e}\Gamma_e^- + |e| \cdot \delta}, \qquad \forall g \in e.$$

□ □

The complete MMO algorithm is described in Algorithm 6:

Algorithm 6: Max-Min online algorithm for SRAMF

Data: Augmented supply S_A, basis supply S_B, hypergraph G with $E(\xi)$, and $\epsilon > 0$.

Result: Near-optimal $S_R \subset S_A$ and the corresponding trip assignment.

Initialization: $S_R \leftarrow \varnothing$ and dual variables $\mathbf{u} \leftarrow 0$;

For each vehicle in S_B (in any order), run Algorithm 5 to obtain $\hat{v}_{ON}(S_B)$

for $k = 1, \ldots, K$ **do**

  for $i \in S_A \backslash S_R$ **do**

    Run the updating subroutine in Algorithm 5 and obtain $\hat{v}_{ON}(S_B + S_R + \{i\})$.

  end

  $i^* = \arg\max_{i \in S_A \backslash S_R} \hat{v}_{ON}(S_B + S_R + \{i\})$;

  $S_R \leftarrow S_R + \{i^*\}$;

end

4.4.3 Extensions to SRAMF under Partition Constraints

We now consider a more general setting where the augmented set S_A is partitioned into M subsets $S_A(1), \cdots S_A(m), \cdots, S_A(M)$ and the platform requires K_m vehicles from each subset.

Example 3: In the market segmentation of Example 1 described in Section 4.1, there are M types of vehicles, so the cardinality constraint is further specified for each vehicle type as partition constraints $\sum_{i \in S_A(m)} y_i \leq K_m$ for all $m \in [M]$.

Example 4: In the mixed autonomy application of Example 2, there are M separate AV zones and the repositioning capacity requirement is proportional to the demand density in each zone. $\sum_{i \in S_A(m)} y_i \leq K_m$ is now a constraint for each AV zone $m \in [M]$.

The original SRAMF problem (2) is now expanded to solve:

$$\underset{y}{\text{maximize}}\, \mathbf{E}[Q(y, \xi)] \tag{18}$$

$$s.t. \sum_{i \in S_A(m)} y_i \leq K_m \qquad \forall m \in [M] \tag{18a}$$

$$y_i \in \{0, 1\} \qquad \forall i \in S_A. \tag{18b}$$

We can extend our result to obtain:

Theorem 5. *The MMO algorithm is a* $\frac{1}{(4+o(1))p \log p}$*-approximation algorithm for SRAMF with partition constraints.*

The proof is identical to that of Theorem 3. The only difference is the use of the following result for max-min covering under a partition constraint (instead of Theorem 4, which only holds for a cardinality constraint).

Theorem 6. *(Gupta et al. 2015) If the covering problem* (15) *satisfies the monotone and* α*-competitive online properties, there is a* $\frac{1}{2\alpha}$*-approximation for the max-min problem with a partition (or matroid) constraint.*

4.5 Numerical Experiments

4.5.1 Data Description and Experiment Setup

We evaluate the effectiveness of the proposed approximation algorithms in two hypothetical mixed-fleet scenarios:

1. **Setting 1** simulates mixed fleets of standard and premium vehicles in **Example 1** and represents the mid-capacity SRAMF scenario. The MoD platform periodically repositions premium vehicles of S_A to serve ride requests when the future demand exceeds the capacity of standard vehicles in S_B. We consider the stochastic nature of system dynamics as the probability of demand surges in some zones across the city. The value of hyperedges in these zones increases when demand surges, rewarding algorithms that successfully reposition in locations with high surge probability. The main task is to reposition premium vehicles to accommodate predicted surge demand.

2. **Setting 2** simulates the early deployment of AVs in **Example 2** and represents the high-capacity SRAMF scenario. Due to regulatory or technological restrictions, we assume that automated MoD buses operate only within certain AV zones (Chen et al. 2017b) and deliver up to ten passengers per trip (Alonso-Mora et al. 2017b). The main task is to periodically reposition K automated MoD buses in these AV zones to accommodate future demand.

To demonstrate the value of employing a stochastic assignment framework, we consider two benchmark models:

1. **Benchmark 1:** *Stochastic assignment using IP solver* solves SRAMF exactly using the SAA approach in Section 4.3.2. This benchmark method and approximation algorithms use the same set of samples to assess on a fair basis. The SAA approach is implemented in a state-of-the-art IP solver (Gurobi 9.1).

2. **Benchmark 2:** *Assignment with mean demand forecasts* solves a deterministic ride-pooling assignment problem based on the mean demand forecasts. This method solves the joint vehicle repositioning and trip assignment problem using a one-shot approach based on the mean hyperedge value and demand distribution $F(\xi)$. The goal is to address the significance of considering demand and supply uncertainties in SRAMF, albeit at the expense of increased computing complexity.

Note that the objective values of Benchmark 2 and SRAMF are not comparable. The following reconstruction procedure is therefore used in evaluations: (1) Select a set of augmented supply S_R based on the average scenario. (2) Generate a new set of test samples as outlined in the SAA method. (3) Recompute the objective values for all algorithms using identical test samples. This procedure can prevent the fallacy of cherry-picking in numerical experiments and is described further in section 4.5.1.2.

4.5.1.1 Data description and preprocess.

We test the performance of these approximation algorithms in a simulated MoD system with mixed fleets. The ride-pooling simulation follows a batch-to-batch procedure similar to that employed in Alonso-Mora et al. (2017b), with a demand forecast module to maximize the expected total value realized by serving travel demand.

Table 4.1 summarizes our experiment settings. The primary data inputs include:

1. Road networks: The road network in Manhattan, New York City (NYC) is obtained from the OpenStreetMap (OSM) data. The average traveling time on each road segment is computed using the historical speed data in (Sundt et al. 2021). In Setting 1, both vehicle types can serve any nearby ride requests made in Manhattan. To demonstrate AVs' early deployment in Setting 2, two AV zones are selected in the planning phase (see Figure 4.9a). These zones are highly congested areas proposed for pedestrianization and could potentially be closed off to most vehicles besides MoD transit services. Due to regulations for safety concerns, automated MoD buses only operate within these AV zones (Chen et al. 2017b).

2. Supply: The basis set S_B represents ride-hailing vehicles with a fixed capacity of two that provides the standard service.

    - In Setting 1, the augmented set S_A represents a set of locations to which premium vehicles can be repositioned (see Figure 4.6). Full-time drivers provide reservation-based service with these premium vehicles, which have a capacity of three passengers (Ma et al. 2017). Thus, the MoD platform can allocate at most K idling premium vehicles to facilitate freelancing drivers who accommodate ad-hoc demand surges for standard services (Dong et al. 2023).
    - In Setting 2, the augmented set S_A represents the initial parking locations from which automated MoD buses are repositioned (see Figure 4.9). Each shuttle bus in S_A has a capacity of up to ten passengers and can operate only within AV zones at the beginning of intervals. If each MoD bus can be repositioned only to a particular subset of locations, we use the generalized setting in Section 4.4.3, where K_m represents the set of approachable locations for vehicle m. The same approximation ratio holds for this extension.

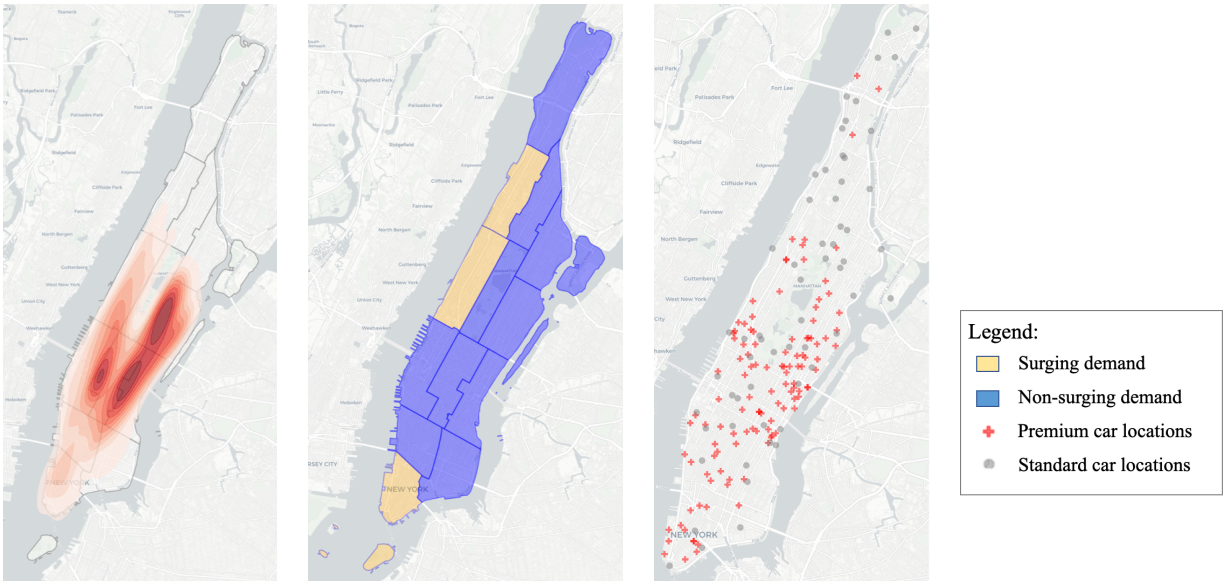
3. Demand: We create a demand forecast model to sample ride requests from the NYC Taxi and Limousine Commission trip data (TLC 2021). The forecast model utilizes this dataset's origin-destination, number of passengers, trip time, and fare information to predict hyperedge values as accurately as possible (See Figure 4.9b).

4. Hyperedge values: The value of each hyperedge e is computed by (1). Each trip's pickup time follows the shortest path connecting all ride requests in the hyperedge e, and customers' preference over mixed fleets is randomly generated such that $v_e > 0$.

5. Time intervals: We choose different matching intervals in Setting 1 and Setting 2 in the batch-to-batch implementation. Setting 1 uses a one-minute interval to provide convenient and responsive MoD services; Setting 2 uses a ten-minute interval to permit repositioning between AV zones. Choosing the relatively large matching interval (ten-minute) also illustrates the scalability of approximation algorithms, whereas Qin et al. (2021a) showed that the optimal interval might depend on the supply-demand relationship.

Table 4.1: Parameters in numerical experiments

Setting	Augmented Set S_A			Basis Set S_B		Problem Statement	Demand-to-Supply Ratio	Matching Interval (minute)	Number of Sample Scenarios N
	Capacity	Location $\#\lvert S_A\rvert$	K	Capacity	Vehicle $\#\ \lvert S_B\rvert$				
Setting 1	2	115	60	2	60	Mid-Capacity SRAMF for Example 1 (standard-premium cars)	$1.7 - 2.0$	1	10
Setting 2	10	30	5	2	115	High-Capacity SRAMF for Example 2 (mixed autonomy)	$35 - 45$	10	50



(a) Demand density (b) Surging demand areas (c) S_A and S_B for Setting 1 in a given scenario

Figure 4.6: Mid-capacity mixed autonomy traffic experiment in Manhattan, NYC.

4.5.1.2 Assessment of algorithms in mixed-fleet simulations.

The objective of both settings established in Section 4.5.1 is to choose a subset of locations for repositioning vehicles S_A to maximize the total expected assignment value. Consequently, the assessment of different benchmarks and proposed approximation algorithms consider three metrics: total computational time (runtime in seconds), the expected assignment value based on predicted demand samples (the objective value of the SRAMF problem defined in (2)), and the average realized assignment value of the proposed locations given new demand samples. In this section we describe these assessment methods and how they are calculated. The numerical experiments conducted in this study generate a fixed number of scenarios, each of which constructs a shareability graph using the process outlined in Appendix B.2 and creates all hyperedges with only positive values.

We measure runtime as the duration in seconds it takes an algorithm to produce the desired output, a subset of augmented supply locations, given the inputs of a hyperedge graph (including associated hyperedge values and the samples of predicted demand). In Benchmark 1, the SRAMF problem is solved to optimality using Gurobi 9.1, a highly optimized MIP solver. The number of variables in Equation (2) equals the product of the number of hyperedges and the sample size, which may increase exponentially in real-world applications. Hence, we set a six-hour computation time limit for solving the SRAMF problem with any method, mainly for the exact solver. Unlike in Benchmark 1, our proposed approximation algorithms focus on solving the SRAMF problem by swapping vehicle locations or adding them sequentially from the set S_A, rather than solving the entire two-stage integer program at once. As a result, a parallel-computing scheme can considerably reduce the total runtime of approximation algorithms by evaluating multiple scenarios concurrently. We report two computation times for these parallelized versions, one for the observed runtime (programmed by Python 3.8 on our server) and another for the hypothetical runtime based on a maximum number of threads. The maximal computing resource (max-thread) runtime limit means that the algorithm can simultaneously evaluate all pairs of candidates in the active set of LSLPR or the dual variables for all hyperedges in MMO. This limit includes additional time for computations in series but excludes the overhead cost of creating processes. The number of used threads for each experiment setting is provided in the footnotes of Tables 2 and 3. Computation times are reported from performance on a server with an 18-core 3.1 GHz processor and 192GB RAM. In some experiments, the overhead cost of parallelizing the MMO algorithm in Python actually increases the overall computation time, so we report a simply vectorized version. Given the inefficiency of the parallelizing process in Python, we believe times closer to the parallel limit can still be achieved.

The computation times of generating demand forecasts and the corresponding shareability

graph are not reported, because this study focuses on reducing the computation time for a given shareability graph (hypergraph) in the SRAMF problem. In practice, the construction processes of these hypergraphs may include more sophisticated demand forecast models (Geng et al. 2019), so their preprocessing times are not considered in the assessment. Note that approximation algorithms can handle more extensive shareability graphs or require significantly shorter computation time limits. However, measuring their optimality gaps requires comparing objective values of approximation algorithms with that of benchmark methods and downsampling from the original taxicab data. The end of this section appends a separate set of experiments to demonstrate the scalability of approximation algorithms with larger instances.

Additionally, we evaluate the performance of our algorithms by comparing the objective value they achieve to that of the optimal IP solution, commonly known as the optimality gap. Let the objective of the IP solver be OPT and the approximation algorithm's solution be ALG. The optimality gap is measured by $(OPT - ALG)/OPT$, and is reported as a percentage.

Finally, to assess the value of incorporating stochasticity in ride-pooling matching, we compare the average performances of these algorithms by dividing the dataset into training and test samples. They are evaluated based on 10 new test samples drawn from the same distribution $F(\xi)$ as the training samples used for initializing the algorithms. Each algorithm and benchmark returns a subset of augmented vehicle locations from the training samples, which are optimally assigned to incoming demand in the next interval. For each test sample, we calculate the optimal matching that can be achieved given the selected augmented supply vehicles. The performance is calculated as a multiplier of the assignment value achieved by Benchmark 1 within each realized sample, which are then averaged across all 10 test samples to give the reported *average test performance*. Note that this multiplier can be greater than one, as the IP solver often produces locations that are optimal for the training samples but may not generalize well to the overall distribution. The IP solver may also reach the computation time limit and find a suboptimal solution. Benchmark 2 was calculated by solving a deterministic second-stage assignment problem (3) with average demand forecasts, reducing it to a much easier-to-solve integer assignment program.

4.5.2 Numerical Results for Mid-Capacity SRAMF

Setting 1 uses large fleets of premium and standard vehicles to provide ride-pooling services in tandem. The demand density, surging demand areas, and two sets S_A, S_B are shown in Figure 4.6. We generate data for surging demand as follows: 1) Divide Manhattan, NYC,

into 12 regions (using NYC Community Districts (Data 2022)); 2) Create a probability matrix of all regions to chart the occurrence of surging demand. For concision, we choose three regions with high probabilities of surges (see Figure 4.6b); the remaining regions have a low probability of surging demand. These regions were chosen to be outside of areas with high demand at the time of prediction to test the quality of service with highly fluctuating demand distributions. The surging magnitude is defined by the multiplier of sampling rates of surging versus the non-surging cases from the actual trips from the NYC taxi dataset. 3) Generate i.i.d. demand profile and build shareability graphs for each scenario. 4) Run all algorithms on the same sample set, using benchmark models, and evaluate numerical results.

Since the shareability graph is constructed for each scenario, the computed optimal routes in a sample scenario are shown in Figure 4.7. As can be seen, premium vehicles selected from the augmented set S_A pick up mainly customers in the surging demand areas. Since the hyperedge values of these rides are likely to have a surge multiplier, the platform tends to reposition more idling premium vehicles and switch them to serve standard requests. The computational results for Setting 1 are summarized in Table 4.2.

Table 4.2: Summary of Numerical Results for Mid-Capacity SRAMF

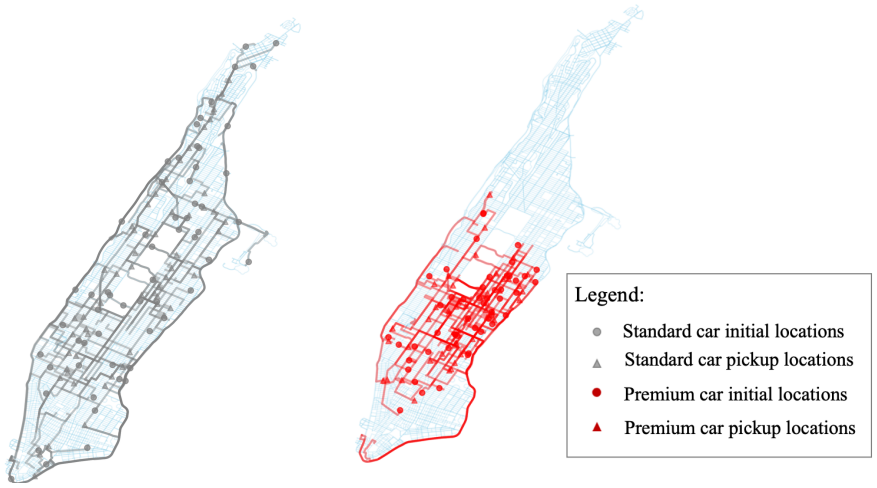
(Base, surge) demand sample rate	Surge prob.	# samples	Benchmark 1	Benchmark 2	LSLPR				MMO			
			IP runtime (s)	Avg test perf.	Runtime (s) (36-thread)	Max-thread runtime (s)	Opt gap (%)	Avg test perf.	Runtime (s) (vectorized)	Max-thread runtime (s)	Opt gap (%)	Avg test perf.
(0.4, 0.6)	0.3	10	1109	1.00	1818	19	**0.1**	1.15	41.1	**0.26**	3.1	1.14
		20	4161	0.98	2682	28	**0.6**	1.14	52.8	**0.35**	3.1	1.13
	0.5	10	1117	1.00	1178	12	**0.2**	1.19	43.8	**0.26**	2.7	1.14
		20	4213	0.99	–	273	**0.0**	1.18	53.5	**0.34**	4.5	1.12
(0.5, 0.75)	0.3	10	1160	1.00	3742	39	**0.1**	1.21	48.7	**0.28**	2.6	1.18
		20	4202	0.98	3893	41	**0.2**	1.19	56.1	**0.41**	5.2	1.14
	0.5	10	1177	1.01	3090	33	**0.1**	1.25	43.0	**0.29**	8.5	1.16
		20	4214	0.98	6365	67	**0.0**	1.22	58.3	**0.42**	10.4	1.12

- The maximum computation time is 3 hours; – denotes hitting the maximum runtime.

4.5.2.1 Computation times.

At small sample sizes, the IP's runtime is fairly comparable to that of the LSLPR approximation algorithm. Gurobi is a powerful and heavily optimized MIP solver, so this result is unsurprising. MMO and LSLPR have a clear advantage at larger sample sizes because they can utilize parallel computation resources. MMO is also the only algorithm to complete within the time limit for this setting, although many LSLPR runs can also reach this threshold with enough parallel threads.

The runtime of LSLPR varies widely as the swap order greatly affects the runtime. When the value ϵ in the stopping criterion is small in cases analyzed by large-scale shareability graph, long runtimes for approximation algorithms are occasionally observed. This is primarily because the algorithm must evaluate many swaps to find one that improves the overall



(a) Standard vehicle routes

(b) Premium vehicle routes

Figure 4.7: Optimal trip assignment and routes in the mid-capacity scenario.

objective value. If the algorithm randomly initializes with a competitive solution of S_R, finding another swap to improve the objective value becomes increasingly difficult. This can lead to the algorithm's evaluating a combinatorial number of swaps per iteration, but the quality of such an approximation attains near optimality. One advantage of LSLPR is that it can be stopped early and still return valid assignments without searching all swaps, which other algorithms and benchmarks cannot do. We verify this hypothesis by the following observation: With $\epsilon = 0.1$, Table 4.2 shows the very small optimality gaps this algorithm can achieve. LSLPR becomes time-competitive in runtime for most instances (see Figure 4.8) when ϵ increases to 0.2.

4.5.2.2 Optimality gaps.

Table 4.2 shows that the actual optimality gaps throughout the numerical experiments are smaller than the theoretical bounds. The optimality gaps of LSLPR ($<= 1\%$) are significantly smaller than those of MMO ($3\% - 10\%$), which matches the theoretical analysis.

Regarding performance on the test samples, LSLPR and MMO vehicle selections consistently achieve 12%-25% higher objective values than those of Benchmark 1 (IP solver). This advantage is always slightly worse when more samples are used initially, demonstrating that the benchmark is optimizing heavily to the specific training samples. This result does not generalize to demand distributions as well as LSLPR or MMO. Although Benchmark 1 might outperform our algorithms on some test samples and obtain a larger objective value

with enough initial samples, the IP solver scales poorly in regard to the average runtime (see Table 4.5). Additionally, the deterministic benchmark shows that disregarding demand uncertainty entirely can cause a $15\% - 22\%$ loss of objective value in the worst case when compared to values that LSLPR and MMO achieved. This is a significant difference considering that the future MoD market is a billion-dollar industry. Therefore, it is valuable to implement SRAMF algorithms to proactively reposition vehicles as in ride-hailing literature (Qin et al. 2021b) instead of solving the deterministic problem.

4.5.3 Sensitivity analysis.

To evaluate the impacts of this parameter on the computational efficiency, we test the computation times and optimality gaps with varying $K \in [10, 80]$. The results in Figure 4.8 show that the computation times of approximation algorithms increase significantly with K, which is a shortcoming of any local-search-based algorithm. The IP solver is relatively unaffected by the change of parameters. However, the optimality gap of these approximation algorithms drops to nearly zero with the increasing budget, as the increased budget has diminishing marginal value to the ride-pooling assignment (i.e., the market is saturated already). Therefore, the platform can select a reasonably small budget for approximation algorithms to have clear advantages over the exact solver.

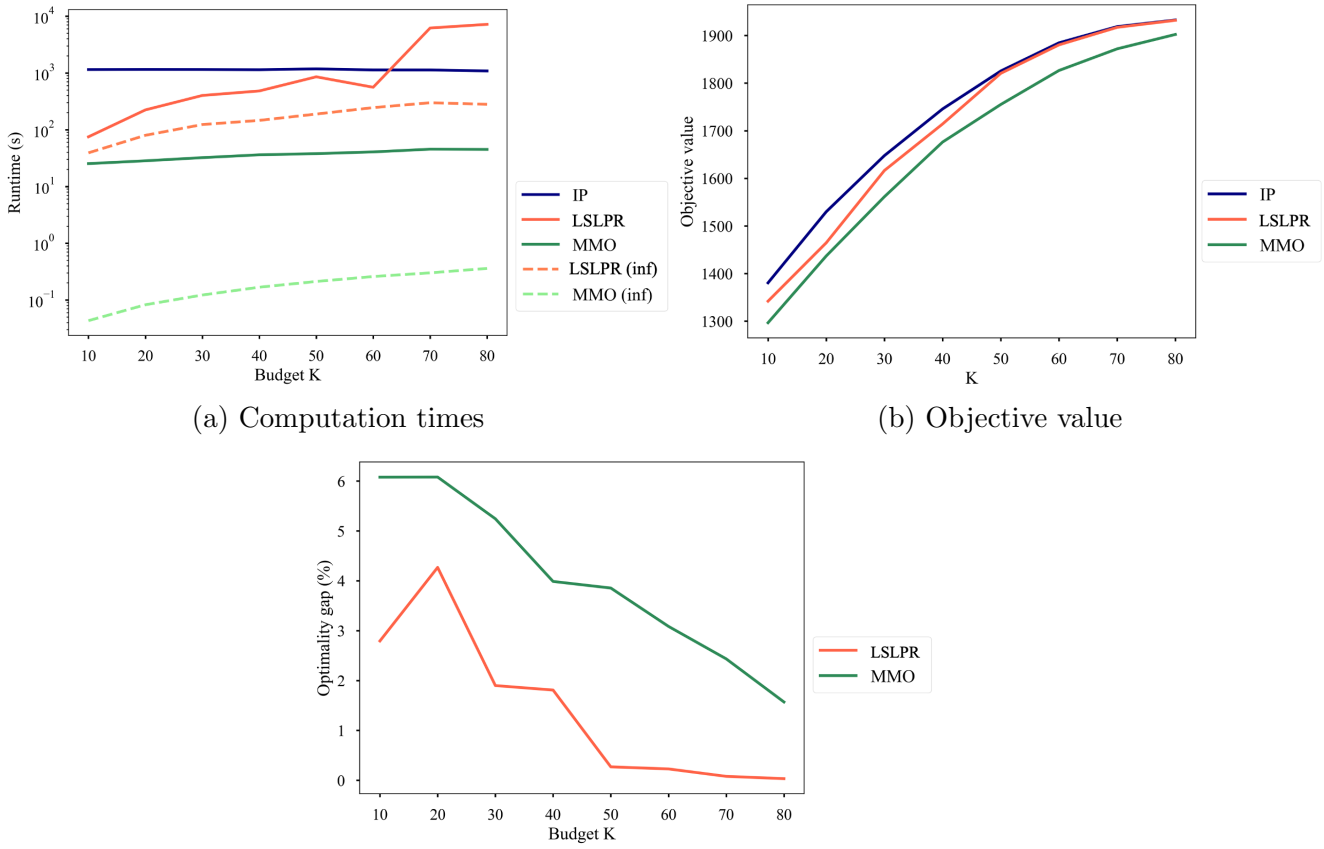
4.5.4 Numerical Results for High-Capacity SRAMF

In Setting 2, the proposed algorithm computes near-optimal solutions for the mixed-autonomy fleet, including repositioning automated MoD buses (AVs) among locations S_A (see Figure 4.9) and determining their pickup routes for demand samples.

The performance of the proposed approximation algorithm is evaluated under different supply and demand distributions. The system is tested in both a relatively balanced demand scenario as well as a massive under-supply scenario, with mean numbers of demand across scenarios ranging from 250 to 4000, respectively, which are considerably larger in a *stochastic* setting. Figure 4.10 shows that algorithms allocate four AVs to the high-demand-for-AV zone and one AV to the low-demand-for-AV zone for such demand forecasts. Notice that these decisions are complementary to CVs' trip assignment decisions, as the latter fleets are still the primary MoD service providers.

4.5.4.1 Computation times.

The reported computation time includes solving for the near-optimal vehicle selection and exact assignment in each scenario. This comparison excludes the runtime required to gener-



(a) Computation times

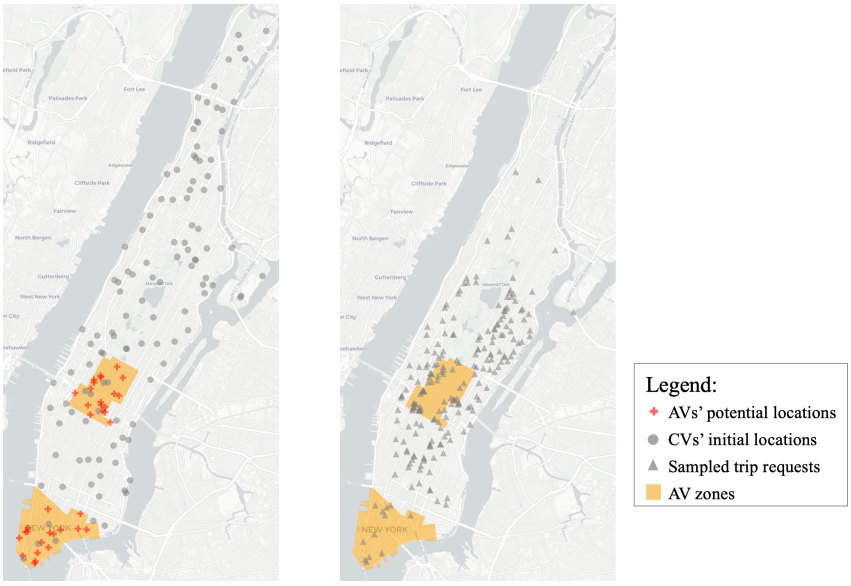
(b) Objective value

(c) Optimality gaps

Figure 4.8: Impact of K on computation time and optimality gap in the mid-capacity scenario.

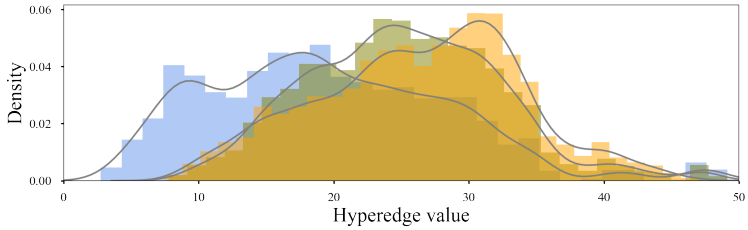
ate hypergraphs in order to emphasize the performance of algorithms in solving the SRAMF problem. Recall that the size of the augmented set $|S_A|$ and the number of hyperedges (the number of decision variables in each scenario) determine the size of the shareability graph. Table 4.3 shows how the total runtime grows with the increasing size of the hypergraph. The computation time of LSLPR and MMO algorithms are shown in Table 4.3.

The largest runtime per iteration is reported in Table 4.3, where the number of hyperedges is the maximal number across all scenarios. Our results show that (a) The max-thread MMO setting obtains near-optimal solutions to SRAMF with the smallest runtimes because it evaluates all potential vehicles in $S_A \setminus S_R$ in parallel. (b) The performance of LSLPR is worse than MMO for high-capacity SRAMF, which matches our approximation ratio analyses where the number of potential swaps grows exponentially. Nevertheless, its computation time will always improve when additional resources are available.



(a) CVs' and AVs' initial locations in Setting 2

(b) Request pickup locations in a sampled experiment

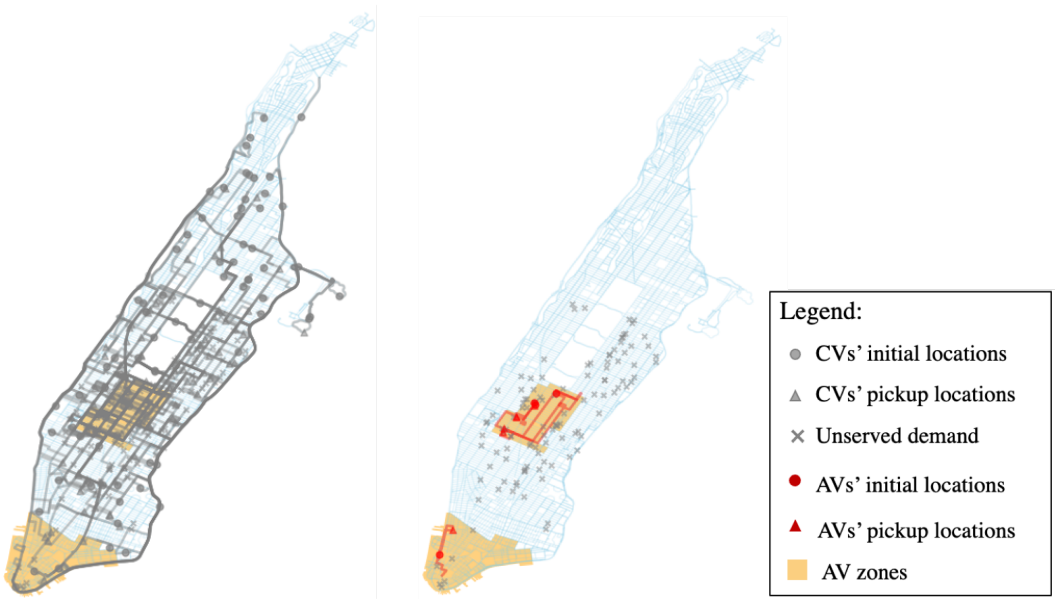


(c) Sampled hyperedge value distributions in shareability graphs

Figure 4.9: High-capacity mixed autonomy traffic experiment in Manhattan, NYC.

4.5.4.2 Optimality gaps.

Table 4.3 shows that optimality gaps of both algorithms grow slightly with the size of shareability graphs, but the overall performance of the proposed approximation algorithms is satisfactory for various supply-demand ratios. The optimality gaps are below 2% throughout tested instances, confirming that approximation ratios derived for the worst-case scenario, $\frac{1}{p^2}$ or $\frac{e-1}{(2e+o(1))p\ln p}$, are loose with the real-world trip data. In other words, the performance degradation of these approximation algorithms is negligible when implementing them in shared mobility systems.



(a) CV routes (b) AV routes

Figure 4.10: Optimal trip assignment and routes in mixed autonomy, high-capacity SRAMF.

4.5.4.3 Sensitivity analysis.

Three sensitivity analyses for the high-capacity SRAMF problem involve (a) distribution of hyperedge values, (b) vehicle number and capacity, and (c) sample size. They test how the performance of these approximation algorithms is affected based on changes in input data and model assumptions. We discuss them individually in this section.

Hyperedge value distribution. The first set of sensitivity analyses aims to check algorithms' performance degrades with different supply and demand distributions. By replacing the empirical hyperedge values with randomly generated hyperedge values, this analysis examines the robustness of these algorithms. Figure 4.11 shows the runtime and optimality gaps with uniformly generated hyperedge values. Figure 4.9c represents that customers' level of trust in the AV technology dominates the hyperedge value such that $v_e = \sum_{j \in t} u_j + \sum_{j \in t} \tilde{u}_{ij} - c(i,t) \approx \sum_{j \in t} \tilde{u}_{ij}$ for each $e \in E(\xi)$ and $\tilde{u}_{ij}$ follows a uniform distribution.

The runtime of random hyperedge values is smaller than those of real-world data, and the optimality gaps stay low across most instances. This is mainly because the empirical hyperedge values are more concentrated around specific values (i.e., the average trip length). Hence it is more difficult to reposition vehicles from the augmented set. In this case, the local-search-based algorithms outperforms other algorithms with uniformly distributed values.

Vehicle capacity. In Stetting 2 numerical experiments, automated MoD buses (AVs)

Table 4.3: Summary of Numerical Results for High-Capacity SRAMF

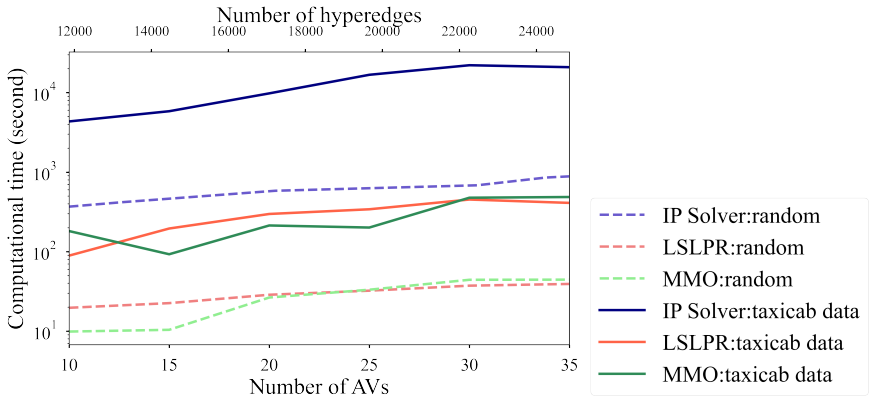
Demand-supply ratio	S_A, S_B	Benchmark 1	LSLPR			MMO		
		IP runtime (second)	LSLPR runtime (8-thread) (second)	LSLPR runtime (max-thread) (second)	Opt. Gap	MMO (8-thread) (second)	MMO (max-thread) (second)	Opt. Gap
1.78	10, 115	1177	384	20	**0.2%**	**38**	**16**	0.4%
1.78	15, 115	1332	383	**19**	0.9%	**51**	**19**	**0.8%**
1.78	20, 115	1470	337	23	**0.7%**	**58**	**15**	0.8%
1.78	25, 115	1354	357	**24**	0.9%	**72**	26	**0.8%**
1.78	30, 115	1464	548	31	**0.5%**	**82**	**15**	0.9%
1.78	35, 115	1448	599	27	**0.4%**	**97**	**17**	1.0%
1.83	10, 115	1425	460	29	0.3%	**48**	**15**	**0.1%**
1.83	15, 115	1516	509	63	0.1%	**60**	**17**	**0.0%**
1.83	20, 115	1730	449	33	0.5%	**77**	**19**	**0.3%**
1.83	25, 115	1817	483	35	**0.3%**	**91**	**22**	0.7%
1.83	30, 115	2027	566	31	**0.7%**	**104**	**21**	0.8%
1.83	35, 115	2373	565	44	**0.5%**	**137**	**31**	1.2%
1.87	10, 115	1180	353	**26**	0.7%	**87**	65	**0.3%**
1.87	15, 115	1350	398	**12**	0.5%	**66**	34	**0.1%**
1.87	20, 115	1509	455	55	1.0%	**78**	**38**	**0.1%**
1.87	25, 115	1650	488	**18**	1.0%	**128**	70	**0.3%**
1.87	30, 115	1690	536	35	1.0%	**105**	**32**	**0.3%**
1.87	35, 115	1795	535	77	1.3%	**141**	**42**	**0.3%**
1.93	10, 115	4468	1011	196	0.4%	**121**	**63**	**0.2%**
1.93	15, 115	6411	1036	**58**	0.5%	**250**	140	**0.4%**
1.93	20, 115	11243	1237	204	**0.2%**	**224**	**72**	0.3%
1.93	25, 115	11820	1318	**28**	0.4%	**390**	167	**0.2%**
1.93	30, 115	5432	1453	99	0.8%	**397**	**86**	**0.6%**
1.93	35, 115	7038	1830	**64**	0.6%	**516**	133	**0.4%**
2.02	10, 115	4348	998	**50**	**0.6%**	**226**	182	1.0%
2.02	15, 115	5843	1792	157	**0.2%**	**175**	**93**	1.0%
2.02	20, 115	9802	2177	**210**	**0.2%**	**442**	214	1.0%
2.02	25, 115	16787	3508	300	**0.5%**	**609**	**201**	0.9%
2.02	30, 115	22141	4143	**198**	0.6%	**1234**	477	**0.4%**
2.02	35, 115	20920	6327	**161**	**0.4%**	**1362**	487	0.5%

- The demand-supply ratio is the average ride requests over the total number of vehicles ($K + |S_B|$); $K = 5$.

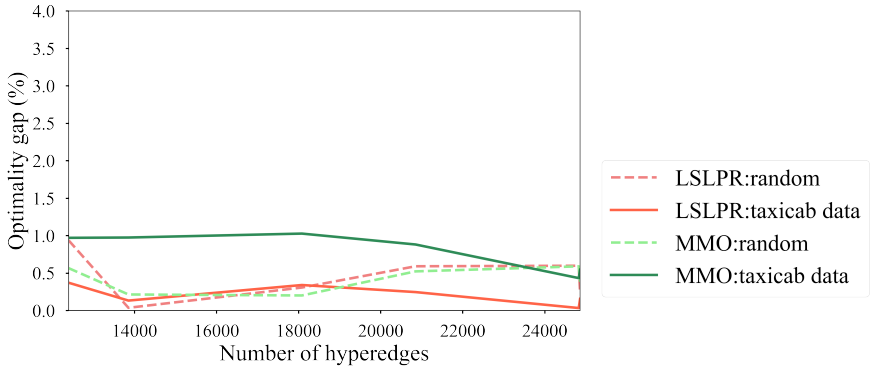
- The max-thread runtimes assume enough threads to evaluate all potential swaps or drivers at once:

  - The total number of variables in Benchmark 1 (IP) ranges from 3×10^6 to 1.2×10^7.

provide mixed autonomy mass transport whose vehicle capacity is up to ten passengers. CVs have a fixed capacity of three passengers. Recall that p bounds the vehicle capacity. Table 4.4 shows how the vehicle capacity affects the approximation ratios. A surprising observation is that the vehicle capacity is not the bottleneck of approximation algorithms' performance throughout the experiments. In contrast, the IP benchmark's computation time increases significantly vehicle capacities. This is because the number of hyperedges plateaus above a certain capacity due to the process of constructing shareability graphs outlined in Section 4.5.1. In order for a high-capacity trip to exist, i.e., the corresponding hyperedge value is positive, all subset trips must also exist. This requirement leads to a combinatorially decreasing number of hyperedges with large trip sizes unless an even larger set of compatible trips exist. Since the density of ride requests in the AV zones does not satisfy the existence



(a) Computation time comparison



(b) Optimality gap comparison

Figure 4.11: Impact of input distribution on computation time and optimality gap.

conditions for compatible trips, the optimality gaps of both approximation algorithms are not evidently affected by the AV capacity.

Table 4.4: Impact of vehicle capacity on computation time and optimality gap

Number of AV locations	AV capacity C_{AV}	Number of hyperedges	Runtime of IP (second)	Optimality gap of LSLPR (%)	Optimality gap of MMO (%)
35	2	8121	220	**0.62**	1.12
35	4	11396	288	**0.80**	0.88
35	6	12048	299	0.99	**0.43**
35	8	12068	298	1.00	**0.03**
35	10	12070	301	1.00	**0.02**

Sample size. The SAA method guaranteeing a uniform convergence to the optimal value does not directly reveal how the sample size affects the total computation time of solving the SRAMF problem. Table 4.5 summarizes the computational results of algorithms compared

with the same instances, which reports runtimes with finite and maximal computational resources (i.e., number of threads for parallel computing).

Table 4.5: Impact of sample size on computation time and optimality gap

Number of samples	Number of AV locations	AV capacity	Avg number of hyperedges	Runtime of IP (second)	Runtime of LSLPR (second)	Optimality gap (%)
10	20	5	3100	14	2	3.23
25	20	5	3100	104	5	1.63
50	20	5	3100	445	12	2.03
75	20	5	3100	907	15	2.01
100	20	5	3100	2088	25	0.91
150	20	5	3100	4076	38	3.15
200	20	5	3100	7485	109	1.01

The approximation algorithms address demand uncertainties in MoD platforms while controlling the computational time to increase linearly with the sample size. Table 4.5 demonstrates that optimality gaps are small for all instances investigated. In light of the important nature of stochastic demand and the movement of basis vehicles, our numerical results suggest that MoD platforms should employ more computational resources to facilitate approximation algorithms with large sample sizes in order to improve the platform's average profit and quality of service.

4.6 Conclusion

SRAMF utilizes a two-stage stochastic integer program to calculate joint vehicle repositioning and assignment for large-scale MoD systems with mixed fleets. This research introduces two approximation algorithms, LSLPR and MMO, which leverage the structure of shareability graphs to assess potential matchings with increased supply. These algorithms efficiently generate near-optimal solutions for maximizing the expected total value of ride-pooling assignments in a relatively short time. The main theoretical results provide provable guarantees for their worst-case performance, as validated by extensive numerical experiments involving mid-capacity and high-capacity vehicles. Our results illustrate the significant benefits of integrating stochastic programming modules into the MoD vehicle dispatching processes to improve system profitability and throughput.

To close this chapter, we point out several promising future research avenues to address the following limitations. First, alternative pickups and dropoffs in trip planning are not permitted, i.e., the total number of ride requests per hyperedge is less or equal to vehicle

capacity. Second, as the SRAMF problem only considers a two-stage uncertainty structure, it is meaningful to extend this framework to a multi-stage setting with time-varying demand forecasts and vehicle repositioning decisions that adapt to revealing scenarios. Constructing hypergraphs with multi-stage demand forecasts will be a major computational bottleneck. Hence, new representations for potential matchings must be developed to address computational issues. Third, penalties related to balking trips or carryover supply are not directly considered in the current setting, whereas they can be incorporated into the hyperedge value in Section 4.3.1. Finally, the current trip assignment does not consider cancellation and re-assignment after dispatching vehicles to passengers. Considering these factors in practice may improve the stability of ride-pooling algorithms.

CHAPTER 5

Conclusions and Future Research

5.1 Research Summary and Findings

The introduction of ride-hailing as a new option for mobility has led to rapid adoption and sector growth in cities across the country. This new mode has brought a flexible, on-demand transportation option at a more affordable price and met the needs of many people. Other benefits arise too, including better mobility options for elderly or children and reduced car dependency. We are only just beginning to understand the effects of these systems though, as studies have recently shown them to also be responsible for increased congestion, vehicle miles, and emissions. In order to mitigate these negative externalities, this dissertation focuses on ride-pooling as a reasonable middle ground. But ride-pooling faces a number of challenges to widespread adoption, which this dissertation aims to address. These challenges roughly fall into three areas, which we discuss along with our contributions below.

5.1.1 Customer Preferences and Comfort (Chapters 2 and 3)

Ride-pooling, more than most other mobility options, requires enough demand density in order to actually be able to find compatible. This means we must In Chapter 2, we present the idea of a customer's mobility profile. Our hypothesis is that such a profile would capture underlying preferences about mobility choices such as origins and destinations, activities, departure time, and route choice. By matching users with similar mobility profiles, they would be more comfortable giving up single occupancy trips and more likely to accept pooling options with those matches. We discuss and present elements of a customers mobility profile based on their historical trajectories. Finally, as one example for how these could be used, we propose a Robust Data Driven Optimization (RDDO) framework for matching users based on this historical profile. Other recommendation engines are also possible using mobility profiles.

We continue to prioritize passenger comfort in Chapter 3 by discussing a family of customer-focused performance metrics that measure important factors such as waiting time and delay. We also design and test some heuristics that explicitly limit delay so as to ensure user satisfaction.

5.1.2 Performance Metrics and Optimality (Chapters 3 and 4)

We determined that many analyses of ride-pooling strategies lack a consistent performance metric, and that some commonly used ones, like occupancy, do not capture the full story alone. To this end, we developed a new metric for measuring system efficiency of ride-pooling platforms and apply it to our heuristic testing. By factoring in passenger trip hours served and total labor hours, we account for delay and penalize inefficient pooling matches. We also demonstrate the importance of including multiple metrics alongside this, including the customer-focused metrics mentioned above, as often they change in different ways and can lead to additional insight.

While these metrics are important for comparing results in specific scenarios between proposed methods, they do not help quantify the maximum or optimal theoretical performance for the system. In Chapter 4 we address this by investigating approximation algorithms in an online matching setting with short term fleet relocation. This setting also expands the driver pool to multi-source, meaning the platform may use different classes or fleets of drivers to serve different demand. We develop two approximation algorithms for mid- and high-capacity vehicles and provide provable worst case guarantees on their performance. These algorithms achieve approximation ratios close to the best possible bounds achievable under these complexity assumptions.

5.1.3 Scalability to Large-Scale Real-Time Operations (Chapters 3 and 4)

A large focus of this dissertation has also been on developing methods that scale to large cities and high demand. This is an especially tricky problem with ride-pooling as potential matching trips often need to be evaluated by solving a time consuming Vehicle Routing Problems (VRPs). We address this in chapter 3 by proposing simple rule-based heuristics for evaluating matches. These can be evaluated quickly without the need for a lengthy VRP or bipartite matching process, and achieve almost similar performance in some scenarios.

Similarly, the SRAMF problem setting presented in chapter 4 is proven to be NP-hard and thus we have no way of solving it in polynomial time. The proposed approximation

algorithms provide polynomial time methods that outperform industrial grade Gurobi optimization while achieving very small (often sub-1%) optimality gaps.

5.2 Directions for Future Work

We have provided some directions for future research at the end of each chapter but there remain a number of more generally applicable and interesting areas to further explore. We detail a few of them here.

5.2.1 Variance Across Cities

Cities across the world vary widely in size, street layout, and demand profile. Each of these elements plays an important role in both the feasibility of ride-pooling and the specific strategies that make a platform work.

For example the calculation of DTW distances in chapter 2 can be tuned based on a threshold for what constitutes similar trajectories. This is affected by the noise of the GPS readings (amplified if in an area of tall buildings or poor signal), as well as road geometry. For example, if two trajectories travel on parallel roads in Manhattan, the distance in that section will be at a minimum the block length times the number of observations.

Additionally, road geometry affects the size and ease of calculation of the restricted subgraph, among other parameters. A platform operating in a suburban area might need to expand the pick-up radius or the allowable delay in order to find a suitable number of available drivers and compatible sharing trips. This however is a tradeoff with computation time and customer satisfaction, and the optimal balance of these parameters will change with the city layout.

5.2.2 Time Period Length and Rolling Horizon

In chapters 3 and 4, we batch requests and make assignments within a short time period. In some scenarios, we observe that a longer time period yields better results (as is the case with bipartite matching) at the cost of longer waiting times for customers and longer computation times due to handling a larger batch of customers.

Especially in large-scale implementations, where computation power is precious, it is important to understand where this balance lies. Additionally, it remains an open question how to account for longer term demand forecasts when relocating and assigning vehicles. Our work in Chapter 4 answers this question for short term demand forecasts, but this does not take into account longer relocation times such as moving from suburbs to downtown.

5.2.3 Integration with Pricing Models

There has been much research separately into pricing models in ride-hailing systems as a method of encouraging or suppressing demand. However at an operational level, this can be integrated with platform matching and relocation as we have control of where drivers will be in future time periods and can encourage demand based on O/D so as to better facilitate ride-pooling and fleet utilization.

5.2.4 Transit System Coordination

It is also important to cooperate and work alongside public transit systems as they run scheduled, higher-capacity trips that are important for relieving customer demand and offer sometimes faster travel times due to dedicated infrastructure. It is critical to facilitate transfers to and from this system and ride-hailing, as otherwise fleet capacity issues would potentially lead to increased negative externalities in congestion and emissions. However incorporating these systems into the models presented in this dissertation, especially with the intent of determining performance guarantees, remains an open question. Coordinating with local public transportation as well as other smaller micro-mobility solutions is an important aspect of near-future mobility systems in cities.

APPENDIX A

Appendix for Chapter 3

A.1 Simulation Environment

The complex nature of real-time ride-hailing operations, with individual-level driver and customer queuing and location decisions, is difficult to accurately capture in simplified models. Small differences in policies of route choice, assignment, or relocation can lead to large differences in performance over longer periods of time. Tuning parameters such as pick-up radius, delay, and batch time can also differ significantly for different cities due to properties of demand distribution and street layout. Thus, in order to capture these behaviors and effects, it is important to have a microscopic simulation platform that accurately represents these real-time decisions and policy implementations.

To this end we have developed **RideSim**, a modular, object-oriented python environment that encompasses all aspects of the ride-hailing and ride-pooling process. This environment allowed us the ability to not only granularly define driver, customer, and platform behaviors but also easily swap between these different policies on the same dataset and random seed. We are also able to measure important statistics on miles traveled, driver status and occupancy, and waiting times at each timestep of the simulation environment.

A.1.1 Environment and Agent Classes

The simulation environment was coded based on three main agent classes: Driver Manager, Customer Manager, and Platform. The environment interacts with these classes through specific functions that are detailed below. Thanks to properties of object-oriented programming, Python makes it easy to "extend" these classes. Essentially, they serve as a blueprint, allowing users to build more complex functionality into their own implementations, so long as they retain the structure and main functions of the original base class. We provide examples below of extensions that can be constructed with only a few lines of code but can define radically different behaviors. The modular nature of this environment makes it both

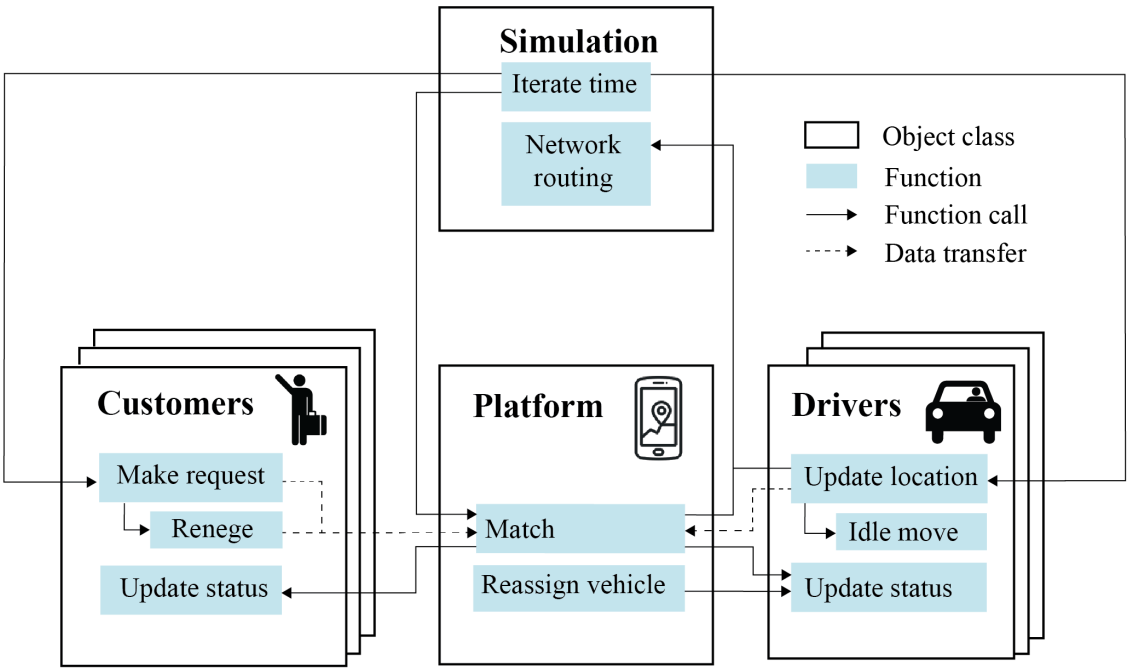


Figure A.1: Diagram of relationships between classes in ride-pooling simulator

extremely flexible and easy to test on, as well as potentially useful to a wide variety of researchers in this field.

These classes are initialized by the user and then input to the simulation environment. The simulation then updates them in order at each timestep as the simulation progresses through time, and results are passed between classes. As the cycle progresses, metrics recorded by the simulation environment are updated. This cycle can be seen in figure A.1.

A.1.1.1 Driver Manager Class

The Driver Manager class is created to handle the vehicle fleet. At a base level, this class is mainly responsible for maintaining driver statuses, routing vehicles, triggering pickup and dropoff of customers upon reaching destinations, and recording VMT and occupancy metrics. The Driver Manager class updates important attributes of each driver that are often called on by the platform or simulation. Chief among these are driver locations (both in latitude/longitude coordinates and representations of road network locations), driver status (offline, idle, en-route to pickup, or en-route to destination), occupancy, and specifically which customers are in the vehicle. These attributes are updated both as the driver moves

along the road network, as well as with data from other classes (i.e. assignment of a customer to a vehicle by the platform). The interaction from the simulation environment and Driver Manager class are mainly handled via the following functions:

- *initialize_drivers*: Responsible for initializing driver locations, statuses, parameters, etc.
  - Currently, this function creates a fleet of idle drivers and randomizes their locations within the service area. This could instead be rewritten to mimic a certain starting distribution of vehicles and statuses.
- *update_drivers*: Handles updating drivers for the next t second timestep, including calling the functions for general and idle move
  - In our code, this function only serves to call the general and idle move functions, but can be extended to also consider worker shifts and determining whether a driver will go offline or come online. We assume a fixed fleet size in our scenarios so we have not demonstrated this implementation, but it would not be difficult to achieve.
- *general_move*: Moves en-route vehicles towards their destination by the given timestep amount, handling route choice and network dynamics
  - This function is fairly straightforward as the shortest path is chosen and the vehicle is moved according to travel times on the links on that path. This is made very easy by the useful features of OSMNX, a python library that helpfully converts real-world OpenStreetMap (OSM) data into a NetworkX data structure. The resulting data structure contains speed limits, distances, and road geometry, and we have further augmented it with average speed data from Uber for our specific network.
- *idle_move*: Provides behavior for idle vehicles over the length of the update timestep
  - At its most simple this function leaves vehicles where they are, essentially parked while waiting for the next trip assignment. However that is not entirely accurate to how real ride-hailing drivers operate, so we have also defined an extended class where idle drivers roam, randomly choosing the next link to drive on at each intersection.

As a demonstration of the ease-of-use and modularity of this code, swapping the idle driver behavior from no movement to random link choice takes less than 20 lines of code total. Modifying this to reflect likely turning probabilites at an intersection instead of a uniform random distribution would be simply be a single line change to that already defined class. Additionally, even smarter drivers can be created, though not used in the results of this paper, that move toward areas of higher demand in order to have a higher likelihood of being matched. This would require additional data from the platform or customer classes, but is absolutely feasible in this framework.

The Driver Manager class can further disperse the behavior to instances of a Driver class, but that is not required and often leads to detrimental computational performance of the simulation. The updates are easiest handled in batches where they can be threaded and parallelized. However the ability to further individualize the behavior is useful in scenarios with many different classes of drivers or where drivers may be randomized with different parameters.

A.1.1.2 Customer Manager Class

The Customer Manager class is created to generate and handle customer trip requests and their waiting queue. This agent class is mainly responsible for keeping track of waiting time, reneging, and trip time metrics. Based on the waiting time, a customer may choose to renege (leave the queue) if they are not matched within a certain time window. In the simulation results, a time window of five minutes was used.

A.1.1.3 Platform Class

The only class modified in each of the simulation runs is the Platform class. The Platform class is responsible for matching the current waiting customers and drivers given their states. The different heuristics given in chapter 3 were coded as different matching functions, which were plugged in to create different platform classes.

APPENDIX B

Appendices for Chapter 4

B.1 Summary of Notation

Table B.1: Summary of notation and acronyms

Notation	Description
S_A, S_B	Augmented set and basis set of vehicles
ξ	Randomly generated scenario
$\ell \in [N]$	Index for sampled scenarios and the total number of samples
$D(\xi)$	Set of demand in scenario ξ
$E(\xi)$	Set of hyperedges in scenario ξ
G	Shareability graph, a hypergraph consisting of supply and demand vertices and hyperedges
e	Each hyperedge $e = \{i, J\}_{i \in S, J \subseteq D}$ is a potential trip where vehicle i serves requests J
u_j	The expected profit of request j
$\tilde{u}_{ij}$	Utility gained from matching request j with preferred vehicle type i
$c(i,t)$	Travel cost for vehicle i to serve trip t
α	Approximation ratio
n	Total number of vehicles such that $\lvert S_A \rvert = n_A$ and $\lvert S_B \rvert = n_B$
K	Maximum number of vehicles allowed from the augmented set
C_i	Capacity of vehicle $i \in S_A \cup S_B$
w_i	Number of passengers in request j
p	Maximum capacity of hyperedge, $p = \max_{i \in S_A \cup S_B}\{1 + C_i\}$
j	Index for travel demand $j \in D(\xi)$
w_j	Size of travel demand $j \in D(\xi)$

$E_{i,\xi}$	Set of hyperedges contains vertex $i \in S_B \cup S_R$ in scenario ξ
t	Trip is a set of demand following the shortest pickup-and-then-dropoff order
v_e	Value of hyperedge $e \in E(\xi)$
$nb(e)$	Neighboring hyperedges $e' \in E(\xi)$ intersecting with e
x_e	Decision variable for hyperedge e, $x_e \in \{0, 1\}$
$\bar{x}_e$	Decision variable for fractional assignment, $x_e \in [0, 1]$
y_i	Decision variable for vehicle $i \in [S_A]$, $y_i \in \{0, 1\}$
$v^*(\cdot)$	Optimal value of the exact GAP
$Q(y, \xi)$	Optimal value of the assignment in scenario ξ
$v_{\max}$	Maximal hyperedge value for all $e \in E$
$v_{\min}$	Minimal hyperedge value for all $e \in E$ such that $v_e > 0$
$\mathcal{I}$	Independent set as a union of hyperedges satisfying the set-packing constraint
S_O	Optimal choice of vehicles for SRAMF $S_O \subset S_A$
S_R	Choice of vehicles from the algorithm $S_R \subset S_A$
$\mathcal{L}$	The bijection between S_R and S_O
$\hat{v}(\cdot)$	The objective value of fractional assignment
$\boldsymbol{z}$	Optimal LP solutions to $\hat{v}(S_O)$
F_i	Hyperedges intersect with vehicle i
H_d	Hyperedges intersect with demand d
$\perp$	Dummy hyperedge in LSLPR
$\Delta_d(e, f)$	Decomposition mapping between hyperedge e and f
$U_{i_1 i_2}$	Marginal value function with $i_1 \in S_R$ and $i_2 \in S_O$
$\hat{v}_{ON}$	Objective value of the online algorithm
$u_{g,\xi}$	Dual variable in the MMO algorithm for $g \in e$ and scenario ξ
Γ_e	$\Gamma_e = \sum_g u_{g,\xi}$ as the left side of dual constraints
$\boldsymbol{c}_e$	Row of cost coefficient in the dual covering problem with entry $c_e(g, \xi)$
M	The augmented set is partitioned into M subsets
ϵ	Error tolerance (for stopping criteria)
δ	Error tolerance (for sample average approximation) or constant in MMO update subroutine
OPT	Optimal value of the SRAMF problem
ALG	Objective value of solving SRAMF by approximation algorithms

B.2 Performance Analysis of Construction of Shareability Graphs

The main idea of recent ride-pooling assignment papers (Santi et al. 2014, Alonso-Mora et al. 2017b, Simonetto et al. 2019) is to separate the problem into two parts: 1) constructing the shareability graph and compatible requests and vehicles, and 2) optimally assigning those trips to vehicles by solving GAP. This paper primarily focuses on algorithms and approximation bounds for the stochastic extension to the second part, but we acknowledge the importance and difficulty of the first task and describe them in detail below for completeness.

B.2.1 Procedure for Constructing Shareability Graphs

$D(\xi)$ is a set of all trip requests revealed in scenario ξ, and this section omits ξ when there is no confusion because the hyperedges for scenarios are generated separately. We consider a number of parameters to be given by the customer or externally dictated to the platform (based on desired service parameters). These include, for each customer j, the maximum waiting time, ω_j, and allowable delay, r_j.

- *(Constraint I)* Travel time from vehicle location to pick-up of customer j in order must be less than ω_j.
- *(Constraint II)* Travel time from origin to destination of customer j in order must be less than r_j.

Additionally, as defined in the setting, the hyperedge weight consists of three parts: value of trip requests $\sum_j u_j$, preference of the vehicle type $\tilde{u}_{sj}$, and travel cost of delivering all trip requests in a single trip. We take a three-request clique (j_1, j_2, j_3) as an example. Let $t_k = \{O_{j_1}, O_{j_2}, \ldots, D_{j_2}, D_{j_3}\}$ be a specific ordered sequence of origins and destinations and $SP(t_k)$ be the shortest path route connecting them. Let $T_e = \cup_k t_k$. Note that this is slightly less demanding than finding all feasible Hamiltonian paths if we enforce that in all trips, the origins must be picked up before any destination is visited. Let $c(s,e) = \min_{t_k \in T_e} v(t_k)$ where $c(t_k)$ is the cost of serving all requests following the shortest-path $SP(t_k)$. We define a set function $f(e)$ that takes a hyperedge consisting of a vehicle s and a potential combination of trips, T_e, as below:

$$f(e) = \begin{cases} 0 & \text{if } \forall t_k \in T_e,\ SP(t_k) \text{ violates constraints I and II} \\ c(s,e) & \text{otherwise} \end{cases}.$$

The bottleneck of computation time is still finding vehicle routes that satisfy the given constraints by solving a constrained Vehicle Routing Problem (VRP) problem, which is NP-hard. Therefore, all heuristic methods can only minimize this bottleneck as much as possible by reducing the number of combinations to check at each step. For example, Ke et al. (2021) suggested a reformulation for finding $c(s, e)$ to avoid enumerating all possible paths.

We combine multiple heuristic methods in literature to construct the shareability graph. First, we need to identify the valid single customer trips for a given vehicle s. Let D_s be the demand that can be served by vehicle s in a single trip within the allowable pickup time. We may further reduce the number of trips by planning on a spatiotemporal graph and examining compatible trips' cliques. By testing trips in order of increasing size and only considering a trip if all subsets of trips (where one request is removed from the trip) are feasible, we reduce the number of candidate trips by orders of magnitude. This heuristic generates the shareability graph in Figure 3.1 in which a set of requests is tested for trip compatibility only if every subset of that set of requests is also compatible.

Lemma 7. *(Alonso-Mora et al. (2017b)) A trip associated with the hyperedge e is feasible for vehicle s only if, for all $j \sim e, j \in D_s$, hyperedges (subtrips) $e' = e \backslash \{j\}$ are feasible.*

The heuristic reduces the candidate hyperedge sets by leveraging the topological relationship between matchable trips of size k and $k + 1$ (see Figure B.1), without eliminating potentially feasible trips. The hypergraph can then be constructed in order of increasing capacity to minimize the number of request sets tested. Additionally, we adopt the following rules to further reduce the number of candidate trips:

1. Since only hyperedges with nonnegative edge weights are of interest, we remove all the trips from the candidate set subject to $f(e) <= 0$.

2. If a vehicle v is not feasible for trip t_k at time τ, it will not be feasible for t_k at any time $\tau' > \tau$ (Liu and Samaranayake 2020).

Let $C_k(D)$ be the set of combinations of size k of the elements of the set D. This process is summarized as follows:

Algorithm 7: Construction of Shareability Graph

Data: Vehicle locations and requests (request time, pick-up, drop-off, preferred vehicle type, acceptable delay)

Result: Set of hyperedges, E, each containing a vehicle, s, and a set of compatible requests for that vehicle to serve in one trip. Hyperedge values are v_e for all $e \in E$.

Initialize $E = \varnothing$

for $s \in S_A \cup S_B$ **do**

  Identify candidate passengers

  $D_s^1 \leftarrow \{e \in D \mid f((s,e)) > 0\}$

  Add hyperedges of size one

  $E_k \leftarrow \bigcup_{e \in D_s^1}(s,e)$

  for $k = 2, \ldots, c$ **do**

    for Demand set $d \in C_k(D_s^{k-1})$ **do**

      Add trips of size k if all subsets exist and value greater than 0

      if $(s,e') \in E_{k-1} \forall i \in C_{k-1}(d)$ and $f((s,e')) > 0$ **then**

        $E_k \leftarrow E_k \cup (s,d)$

        $D_s^k \leftarrow D_s^k \cup d$

$E = \bigcup_{k=1}^{p-1} E_k$

Return hyperedges E and their values v_e

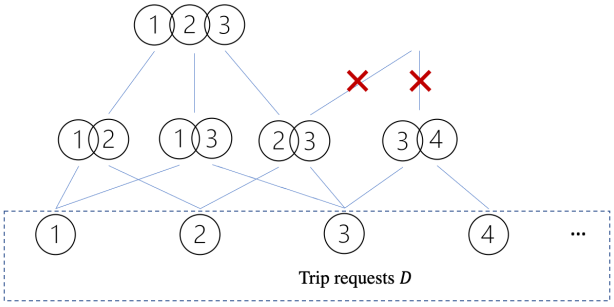


Figure B.1: Topological relationship between cliques of matchable requests. In this example, (2,3,4) is not a valid combination of requests because the (2,4) combination was not valid.

B.2.2 Performance and Complexity Analysis of the Hypergraph Construction Procedure

B.2.2.1 Optimality analysis.

The two-step ride-pooling assignment that first constructs the hypergraph and then solves GAP obtains the exact optimal solution of solving the joint VRP and enjoys the computational advantage for large fleets. Since this work focuses on stochastic assignment, the optimality analysis does not consider the errors of computing hyperedge values. The following results from Alonso-Mora et al. (2017b) provide positive guarantees for returning a feasible set of hyperedges in the shareability graph: without enumerating all trip combinations:

1. v^* from solving GAP on the shareability graph obtains the optimal value for ride-pooling for an arbitrary batch of supply and demand.

2. The construction of the shareability graph is anytime optimal, i.e., given additional computational resources, the set of hyperedges is only expanded to allow for improved matching.

The second property guarantees using a capacity bound, the threshold of which is derived below, as an early stopping criterion in generating the hypergraph will still provide satisfactory results. Solving GAP on this reduced shareability graph can guarantee anytime optimality such that the output is near-optimal for the original problem with high probability.

B.2.2.2 Computational complexity analysis.

We consider a fixed sample of demand D and vehicles S in this section as the hyperedges of each scenario can be generated in parallel. The realized demand has size $|D| = d$.

Lemma 8. *In the worst case, where all demand is compatible and can be served by all vehicles, the runtime is* $O(|S|d^{p-1})$.

While in the worst-case runtime is large, this scenario only arises when all trips are compatible for ride-pooling, which is unlikely in practice. Therefore, we consider the Erdos-Renyi model in which an arbitrary pair of demand is matchable (i.e., satisfy the conditions above) with probability q. Empirical studies showed that q was often a small number (< 0.1) over a large area (Ke et al. 2021).

Lemma 9 (Bollobás and Erdös (1976))**.** *The expected number of cliques of size k is* $\binom{d}{k} q^{\binom{k}{2}}$.

For example, with $d = 1000$ and $q = 0.1$, the expected number of cliques of size 3 (each vehicle deliver at most two requests in a single trip) is 500. Often we observe the size of complete cliques of compatible trips to be less than 10, our maximum tested capacity, and the total number of hyperedges is manageable.

Lemma 10 (Matula (1976))**.** *As $d \to \infty$, the maximal clique size ρ takes on one of at most two values around $\frac{2\log d}{log 1/q}$ with probability tending to one, i.e. with $b = 1/q$, $\lfloor 2\log_b d \rfloor < \rho < \lceil 2\log_b d \rceil$.*

Therefore, we only need to consider hyperedges with size less than $p^* = \min\{p, \rho + 1\}$ (i.e., the height of the cliques' graph in Figure B.1). We have the following theorem for the runtime of constructing shareability graphs.

Theorem 7. *In the average case that the demand and supply profiles satisfying the random geometric graph conditions, the runtime is $O(|S|d^{p^*-1})$.*

Proof. Proof: The expected number of hyperedges connected to vehicle s, $E_{s,\max}$, is bounded by

$$\begin{aligned} E_{s,\max} &= \binom{d}{1}1^2 + \binom{d}{2}2^2 q + \cdots + \binom{d}{(p^*-1)}(p^*-1)^2 q^{(p^*-2)} \\ &\leq ed + \left(\frac{ed}{2}\right)^2 2^2 q + \cdots + \left(\frac{ed}{p^*-1}\right)^{p^*-1}(p^*-1)^2 q^{(p^*-2)} \\ &= ed + \frac{1}{q}\left[\left(\frac{eqd}{2}\right)^2 2^2 + \cdots + \left(\frac{eqd}{p^*-1}\right)^{p^*-1}(p^*-1)^2\right] = O(d^{p^*-1}), \end{aligned}$$

where e is the Euler's number. □

B.3 Supplementary Results for Approximation Algorithms

B.3.1 Proof for Sample Average Approximation in SRAMF

Proof. Proof: We denote the optimal value of the SRAMF problem (2) as v^* and the optimal value of problem for the objective from Algorithm 4 as $\hat{v}(S_O)$. Let δ be the upper bound of the optimality gap $v^* - \hat{v}(S_O)$. We assume that $\mathbb{E}[Q(y, \xi)] = \Omega(m^{-2})$ for any ξ where m is a given constant. The main task is to show that:

1. $\mathbb{E}[\hat{v}(S_O)] = \Omega(m^2)$ with the sample size $N = \frac{m^4}{\delta^2}$.

2. $Pr(\hat{v}(S_O) \notin [(1-\delta)v^*, (1+\delta)v^*]) \leq \exp(-\frac{\delta^2}{2}\mathbb{E}[\hat{v}(S_O)])$.

Let $D(\xi) < D$ for all ξ. For any selection of vehicles in S_A denoted by $y \in Y$, $\mathbb{E}[Q(y,\xi)^2] < \infty$, because we can choose K vertices in S_A with maximum number of D edges. The upper bound of of hyperedge value v_{ij} is $v_{\max}$. Thus we have $\mathbb{E}[Q(y,\xi)^2] < K^2|v_{\max}|^2D^2 < \infty$. Without loss of generality, we draw the following observations from the standard stochastic programming literature (Pagnoncelli et al. 2009):

1. $\hat{v}(S_O) \to v^*$ as $N \to \infty$;

2. $\mathbb{E}[\hat{v}(S_O)] \geq v^*$.

Since N samples are i.i.d., we can use the Chernoff bound on the measure:

$$Pr(\hat{v}(S_O) \notin [(1-\delta)v^*, (1+\delta)v^*]) \leq \exp(-\frac{\delta^2}{2}\mathbb{E}[\hat{v}(S_O)]).$$

Setting $N = m^4/\delta^2$ and using the assumption that $\mathbb{E}[Q(y,\xi)] = \Omega(m^{-2})$, by Jensen's inequality, we have:

$$\delta^2\mathbb{E}[\hat{v}(S_O)] \geq \delta^2 N \cdot E[Q(y,\xi)],$$

i.e., $\delta^2 \cdot \mathbb{E}[\hat{v}(S_O)] = \Omega(m^2)$. We have

$$Pr\Big(\hat{v}(S_O) \notin [(1-\delta)v^*, (1+\delta)v^*]\Big) \leq \exp(-\Omega(m^2)),$$

which achieves the second task as

$$\begin{aligned}
&Pr\left((\frac{1}{2}-\epsilon)\hat{v}(S_O) < (\frac{1}{2}-\epsilon)(1-\delta)v^*\right) + Pr\left((\frac{1}{2}-\epsilon)\hat{v}(S_O) > (\frac{1}{2}-\epsilon)(1+\delta)v^*\right) \\
\leq &Pr\left(\hat{v}(S_R) < (\frac{1}{2}-\epsilon)(1-\delta)v^*\right) + Pr\left(\hat{v}(S_R) > (\frac{1}{2}-\epsilon)(1+\delta)v^*\right) \\
\leq &\exp(-\Omega(m^2)).
\end{aligned}$$

The first inequality is because $\frac{1}{p^2}\hat{v}(S_O) \leq \hat{v}(S_R) \leq \hat{v}(S_O)$. This concludes the approximation ratio for LSLPR algorithm for the stochastic counterpart of the ride-pooling problem. □

□

B.3.2 Proof for Lemma 1

We use a network flow formulation to prove the existence of the mapping $\Delta_d : H'_d \times H'_d \to \mathbb{R}_+$. Consider a bipartite graph with nodes $L = \{\ell_e : e \in H'_d\}$ and $R = \{r_f : f \in H'_d\}$, and arcs $L \times R$. There is an additional source node s, and arcs from s to each L-node and arcs from

each R-node to s. Every arc (i,j) in this network has a *lower bound* $\alpha(i,j)$ and an *upper bound* $\beta(i,j)$. The goal is to find a *circulation* z such that $\alpha(i,j) \le z(i,j) \le \beta(i,j)$ for all arcs (i,j). Recall that a circulation is an assignment of non-negative values to the arcs of the network so that the in-flow equals the out-flow at every node. The lower/upper bounds are set as follows.

1. For each arc $(i,j) \in L \times R$, we have $\alpha(i,j) = 0$ and $\beta(i,j) = \infty$.
2. For each arc (s, ℓ_e) where $e \in H'_d$, we have $\alpha(s,\ell_e) = \beta(s,\ell_e) = x_e$.
3. For each arc (r_f, s) where $f \in H'_d$, we have $\alpha(r_f,s) = \beta(r_f,s) = z_f$.

Recall that $\boldsymbol{x}$ and $\boldsymbol{z}$ are the LP solutions corresponding to $\hat{v}(S_R)$ and $\hat{v}(S_O)$.

Given *any* circulation z, we define $\Delta_d(e,f) = z(\ell_e, r_f)$ for all $e, f \in H'_d$. Then, it is easy to see that all 3 conditions in Lemma 4 are satisfied.

It just remains to prove the existence of some circulation. By Hoffman's circulation theorem (Hoffman 2003), there is a circulation if and only if

$$\alpha\left(\delta^-(T)\right) \le \beta\left(\delta^+(T)\right), \qquad \forall T \text{ subset of nodes.} \tag{1}$$

Above, $\delta^-(T)$ denotes all arcs from a node outside T to a node inside T; similarly, $\delta^+(T)$ denotes all arcs from a node inside T to a node outside T. This condition can be verified using the following cases:

- $T \cap L \neq \varnothing$ and $T \cap R \neq R$. In this case, there is some arc from $L \times R$ in $\delta^+(T)$, so the RHS in (1) is ∞, which is clearly satisfies the condition.
- $T \cap L = \varnothing$. If source $s \notin T$ then $\alpha(\delta^-(T)) = 0$; so (1) is clearly true. If source $s \in T$ then $\beta(\delta^+(T)) \ge \sum_{e \in H'_d} x_e = 1$ as all of L lies outside T, and clearly $\alpha(\delta^-(T)) \le 1$; so (1) holds.
- $T \cap R = R$. If $s \in T$ then $\alpha(\delta^-(T)) = 0$; so (1) is clearly true. If source $s \notin T$ then $\beta(\delta^+(T)) \ge \sum_{f \in H'_d} z_f = 1$ as all of R lies inside T, and clearly $\alpha(\delta^-(T)) \le 1$; so (1) holds.

B.3.3 Supplementary Results for MMO Algorithm

Recall that $\hat{v}(S_R) = \sum_\xi \hat{v}(S_R, \xi)$ where $\hat{v}(S_R, \xi)$ is defined as the LP in (4). So, we can write $\hat{v}(S_R)$ as the following LP:

$$\begin{aligned}
\hat{v}(S_R) = \underset{x}{\text{maximize}} \frac{1}{N} \sum_\xi \sum_{e \in E(\xi)} v_e x_e^\xi & \qquad (2) \\
s.t. \sum_{e \in E(\xi): j \in e} x_e^\xi \leq 1 & \qquad \forall j \in D(\xi) \quad \forall \xi \\
\sum_{e \in E(\xi): i \in e} x_e^\xi \leq 1 & \qquad \forall i \in S_A \cup S_B \quad \forall \xi \\
x_e^\xi = 0 & \qquad \forall e \sim S_A \setminus S_R \quad \forall \xi \\
x_e^\xi \geq 0 & \qquad \forall e \in E(\xi) \quad \forall \xi.
\end{aligned}$$

For any vehicle i and scenario ξ, set $F_{i,\xi} \subseteq E(\xi)$ denotes all the hyperedges incident to i in scenario ξ. Note that all variables x_e^ξ with $e \sim S_A \setminus S_R$ are set to zero. So, it suffices to consider the LP with variables x_e^ξ for $e \in F_{i,\xi}$ and $i \in S_B \cup S_R$.

We now consider the dual of the above LP (which has the same optimal value by strong duality). Let $G = S_A \cup S_B \cup (\cup_\xi D(\xi))$ denote a combined groundset consisting of *all* vehicles and demands from all scenarios. The dual variables are $u_{g,\xi}$ for all $g \in G$ and scenarios ξ. The dual LP is:

$$\begin{aligned}
\hat{v}(S_R) \;=\; & \underset{u}{\text{minimize}} \sum_\xi \sum_{g \in G} u_{g,\xi} \\
& s.t. \sum_{g \in e} u_{g,\xi} \geq \frac{v_e}{N}, \qquad \forall e \in F_{i,\xi}, \quad \forall \xi, \quad \forall i \in S_R \cup S_B \\
& \boldsymbol{u} \geq 0.
\end{aligned}$$

BIBLIOGRAPHY

Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proceedings of the National Academy of Sciences, 114(3):462–467, 2017a. ISSN 0027-8424. doi: 10.1073/pnas.1611675114. URL https://www.pnas.org/content/114/3/462.

Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proceedings of the National Academy of Sciences, 114(3):462–467, 2017b.

Esther M Arkin and Refael Hassin. On local search for weighted k-set packing. Mathematics of Operations Research, 23(3):640–648, 1998.

Arsam Aryandoust, Oscar van Vliet, and Anthony Patt. City-scale car traffic and parking density maps from uber movement travel time data. Scientific data, 6(1):1–18, 2019.

Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. Maximum weight online matching with deadlines. arXiv preprint arXiv:1808.03526, 2018.

Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pages 551–563, 2017.

Xiaohui Bei and Shengyu Zhang. Algorithms for trip-vehicle assignment in ride-sharing. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

Saif Benjaafar, Shining Wu, Hanlin Liu, and Einar Bjarki Gunnarsson. Dimensioning on-demand vehicle sharing systems. Management Science, 2021.

Dimitris Bertsimas and John N Tsitsiklis. Introduction to linear optimization, volume 6. Athena scientific Belmont, MA, 1997.

Béla Bollobás and Paul Erdös. Cliques in random graphs. In Mathematical Proceedings of the Cambridge Philosophical Society, volume 80(3), pages 419–427. Cambridge University Press, 1976.

Anton Braverman, Jim G Dai, Xin Liu, and Lei Ying. Empty-car routing in ridesharing systems. Operations Research, 67(5):1437–1452, 2019.

Niv Buchbinder, Shahar Chen, Anupam Gupta, Viswanath Nagarajan, and Joseph Naor. Online packing and covering framework with convex objectives. arXiv preprint arXiv:1412.8347, 2014.

Francisco Castro, Peter Frazier, Hongyao Ma, Hamid Nazerzadeh, and Chiwei Yan. Matching queues, flexibility and incentives. arXiv preprint arXiv:2006.08863, 2020.

Yuk Hei Chan and Lap Chi Lau. On linear and semidefinite programming relaxations for hypergraph matching. Mathematical programming, 135(1):123–148, 2012.

Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. SIAM Journal on Computing, 35(3):713–728, 2005.

Danjue Chen, Soyoung Ahn, Madhav Chitturi, and David A Noyce. Towards vehicle automation: Roadway capacity formulation for traffic mixed with regular and automated vehicles. Transportation research part B: methodological, 100:196–221, 2017a.

Zhibin Chen, Fang He, Yafeng Yin, and Yuchuan Du. Optimal design of autonomous vehicle zones in transportation networks. Transportation Research Part B: Methodological, 99:44–61, 2017b.

Judd Cramer and Alan B Krueger. Disruptive change in the taxi business: The case of uber. American Economic Review, 106(5):177–82, 2016.

NYC Open Data. Nyc community district data. https://data.cityofnewyork.us/City-Government/Community-Districts/yfnk-k7r4, 2022. Accessed: 2021-11-02.

Xuan Di, Henry X Liu, Xuegang Ban, and Hai Yang. Ridesharing user equilibrium and its implications for high-occupancy toll lane pricing. Transportation Research Record, 2667(1):39–50, 2017.

Felipe F Dias, Patrícia S Lavieri, Taehooie Kim, Chandra R Bhat, and Ram M Pendyala. Fusing multiple sources of data to understand ride-hailing use. Transportation Research Record, 2673 (6):214–224, 2019.

Jing Dong and Rouba Ibrahim. Managing supply in the on-demand economy: Flexible workers, full-time employees, or both? Operations Research, 68(4):1238–1264, 2020.

Tingting Dong, Zhengtian Xu, Qi Luo, Yafeng Yin, Jian Wang, and Jieping Ye. Optimal contract design for ride-sourcing services under dual sourcing. Transportation Research Part B: Methodological, 146:289–313, 2021.

Tingting Dong, Xiaotong Sun, Qi Luo, Jian Wang, and Yafeng Yin. The dual effects of team contest design on on-demand service work schedules. Service Science, 2023.

Gregory D Erhardt, Sneha Roy, Drew Cooper, Bhargava Sana, Mei Chen, and Joe Castiglione. Do transportation network companies decrease or increase congestion? Science advances, 5(5): eaau2670, 2019.

Uriel Feige, Kamal Jain, Mohammad Mahdian, and Vahab Mirrokni. Robust combinatorial optimization with exponential scenarios. In International Conference on Integer Programming and Combinatorial Optimization, pages 439–453. Springer, 2007.

Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pages 611–620, 2006.

Zoltán Füredi, Jeff Kahn, and Paul D. Seymour. On the fractional matching polytope of a hypergraph. Combinatorica, 13(2):167–180, 1993.

Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01):3656–3663, Jul. 2019. doi: 10.1609/aaai.v33i01.33013656. URL https://ojs.aaai.org/index.php/AAAI/article/view/4247.

Marta C. González, César A. Hidalgo, and Albert-László Barabási. Understanding individual human mobility patterns. Nature, 453(7196):779–782, 2008. doi: 10.1038/nature06958. URL https://doi.org/10.1038/nature06958.

Jeffery B. Greenblatt and Susan Shaheen. Automated vehicles, on-demand mobility, and environmental impacts. Current Sustainable/Renewable Energy Reports, 2(3):74–81, 2015. doi: 10.1007/s40518-015-0038-5. URL https://doi.org/10.1007/s40518-015-0038-5.

Harish Guda and Upender Subramanian. Your uber is arriving: Managing on-demand workers through surge pricing, forecast communication, and worker incentives. Management Science, 65(5):1995–2014, 2019.

Xiaotong Guo, Nicholas S Caros, and Jinhua Zhao. Robust matching-integrated vehicle rebalancing in ride-hailing system with uncertain demand. Transportation Research Part B: Methodological, 150:161–189, 2021.

Anupam Gupta and Viswanath Nagarajan. Approximating sparse covering integer programs online. Mathematics of Operations Research, 39(4):998–1011, 2014.

Anupam Gupta, Viswanath Nagarajan, and R Ravi. Robust and maxmin optimization under matroid and knapsack uncertainty sets. ACM Transactions on Algorithms (TALG), 12(1): 1–21, 2015.

Jonathan D Hall, Craig Palsson, and Joseph Price. Is uber a substitute or complement for public transit? Journal of Urban Economics, 108:36–50, 2018.

Mohd Hafiz Hasan and Pascal Van Hentenryck. The benefits of autonomous vehicles for community-based trip sharing. Transportation Research Part C: Emerging Technologies, 124:102929, 2021.

Mohd Hafiz Hasan, Pascal Van Hentenryck, and Antoine Legrain. The commute trip-sharing problem. Transportation Science, 54(6):1640–1675, 2020.

Andrew J. Hawkins. Uber and lyft are the 'biggest contributors' to san francisco's traffic congestion, study says. The Verge, May 2019. URL https://www.theverge.com/2019/5/8/18535627/uber-lyft-sf-traffic-congestion-increase-study.

Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. Computational Complexity, 15(1):20–39, 2006.

Alejandro Henao and Wesley E Marshall. The impact of ride-hailing on vehicle miles traveled. Transportation, 46(6):2173–2194, 2019.

Stephan Herminghaus. Mean field theory of demand responsive ride pooling systems. Transportation Research Part A: Policy and Practice, 119:15–28, 2019.

Alan J Hoffman. Inequalities to extremal combinatorial analysis. In Selected Papers of Alan Hoffman With Commentary, volume 10, page 244. World Scientific, 2003.

Zihan Hong, Ying Chen, Hani S Mahmassani, and Shuang Xu. Commuter ride-sharing using topology-based vehicle trajectory clustering: Methodology, application and impact evaluation. Transportation Research Part C: Emerging Technologies, 85:573–590, 2017.

Michael Hyland and Hani S. Mahmassani. Dynamic autonomous vehicle fleet operations: Optimization-based strategies to assign avs to immediate traveler demand requests. Transportation Research Part C: Emerging Technologies, 92:278 – 297, 2018. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2018.05.003. URL http://www.sciencedirect.com/science/article/pii/S0968090X18306028.

Ishan Jindal, Zhiwei Tony Qin, Xuewen Chen, Matthew Nokleby, and Jieping Ye. Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining. In 2018 IEEE International Conference on Big Data (Big Data), pages 1417–1426. IEEE, 2018.

Renos Karamanis, Eleftherios Anastasiadis, Marc Stettler, and Panagiotis Angeloudis. Vehicle redistribution in ride-sourcing markets using convex minimum cost flows. IEEE Transactions on Intelligent Transportation Systems, 2021.

Jintao Ke, Zhengfei Zheng, Hai Yang, and Jieping Ye. Data-driven analysis on matching probability, routing distance and detour distance in ride-pooling services. Transportation Research Part C: Emerging Technologies, 124:102922, 2021.

Pantelis Kopelias, Elissavet Demiridi, Konstantinos Vogiatzis, Alexandros Skabardonis, and Vassiliki Zafiropoulou. Connected & autonomous vehicles – environmental impacts – a review. Science of The Total Environment, 712:135237, 2020. ISSN 0048-9697. doi: https://doi.org/10.1016/j.scitotenv.2019.135237. URL http://www.sciencedirect.com/science/article/pii/S0048969719352295.

Patrícia S Lavieri and Chandra R Bhat. Modeling individuals' willingness to share trips with strangers in an autonomous vehicle future. Transportation research part A: policy and practice, 124:242–261, 2019.

DongWoo Lee and Steve H. L. Liang. Crowd-sourced carpool recommendation based on simple and efficient trajectory grouping. In Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science, CTS '11, page 12–17, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450310345. doi: 10.1145/2068984.2068987. URL https://doi.org/10.1145/2068984.2068987.

Jia Li, Di Chen, and Michael Zhang. Equilibrium modeling of mixed autonomy traffic flow based on game theory. Transportation research part B: methodological, 166:110–127, 2022.

Shukai Li, Qi Luo, and Robert Cornelius Hampshire. Optimizing large on-demand transportation systems through stochastic conic programming. European Journal of Operational Research, 295(2):427–442, 2021.

Yang Liu and Samitha Samaranayake. Proactive rebalancing and speed-up techniques for on-demand high capacity ridesourcing services. IEEE Transactions on Intelligent Transportation Systems, 2020.

Mustafa Lokhandwala and Hua Cai. Dynamic ride sharing using traditional taxis and shared autonomous taxis: A case study of nyc. Transportation Research Part C: Emerging Technologies, 97:45–60, 2018.

Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. Competitive ratios for online multi-capacity ridesharing. In Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, pages 771–779, 2020.

Qi Luo, Zhiyuan Huang, and Henry Lam. Dynamic congestion pricing for ridesourcing traffic: a simulation optimization approach. In Proceedings of Winter Simulation Conference, 2019.

Jiaqi Ma, Xiaopeng Li, Fang Zhou, and Wei Hao. Designing optimal autonomous vehicle sharing and reservation systems: A linear programming approach. Transportation Research Part C: Emerging Technologies, 84:124–141, 2017.

Iliya Markov, Rafael Guglielmetti, Marco Laumanns, Anna Fernández-Antolín, and Ravin de Souza. Simulation-based design and analysis of on-demand mobility services. Transportation Research Part A: Policy and Practice, 149:170–205, 2021.

Neda Masoud and R. Jayakrishnan. A real-time algorithm to solve the peer-to-peer ride-matching problem in a flexible ridesharing system. Transportation Research Part B: Methodological, 106:218 – 236, 2017. ISSN 0191-2615. doi: https://doi.org/10.1016/j.trb.2017.10.006. URL http://www.sciencedirect.com/science/article/pii/S0191261517301169.

David W Matula. The largest clique size in a random graph. Department of Computer Science, Southern Methodist University Dallas, Texas . . . , 1976.

J. Carlos Martínez Mori and Samitha Samaranayake. On the request-trip-vehicle assignment problem. In Proceedings of the 2021 SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21), pages 228–239, 2021.

Abdullah Mueen and Eamonn Keogh. Extracting optimal performance from dynamic time warping. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, page 2129–2130, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2945383. URL https://doi.org/10.1145/2939672.2945383.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. Mathematical programming, 14(1):265–294, 1978.

Temel Öncan. A survey of the generalized assignment problem and its applications. INFOR: Information Systems and Operational Research, 45(3):123–141, 2007.

Bernardo K Pagnoncelli, Shabbir Ahmed, and Alexander Shapiro. Sample average approximation method for chance constrained programming: theory and applications. Journal of Optimization Theory and Applications, 142(2):399–416, 2009.

Dominik Pelzer, Jiajian Xiao, Daniel Zehe, Michael H Lees, Alois C Knoll, and Heiko Aydt. A partition-based match making algorithm for dynamic ridesharing. IEEE Transactions on Intelligent Transportation Systems, 16(5):2587–2598, 2015.

Guoyang Qin, Qi Luo, Yafeng Yin, Jian Sun, and Jieping Ye. Optimizing matching time intervals for ride-hailing services using reinforcement learning. Transportation Research Part C: Emerging Technologies, 129:103239, 2021a.

Zhiwei Qin, Hongtu Zhu, and Jieping Ye. Reinforcement learning for ridesharing: A survey. arXiv preprint arXiv:2105.01099, 2021b.

Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. Proceedings of the National Academy of Sciences, 111(37):13290–13294, 2014.

Maximilian Schreieck, Hazem Safetli, Sajjad Ali Siddiqui, Christoph Pflügler, Manuel Wiesche, and Helmut Krcmar. A matching algorithm for dynamic ridesharing. Transportation Research Procedia, 19:272–285, 2016.

Pavel Senin. Dynamic time warping algorithm review. Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA, 855(1-23):40, 2008.

Sanket Shah, Meghna Lowalekar, and Pradeep Varakantham. Neural approximate dynamic programming for on-demand ride-pooling. Proceedings of the AAAI Conference on Artificial Intelligence, 34(01):507–515, Apr. 2020. doi: 10.1609/aaai.v34i01.5388. URL https://ojs.aaai.org/index.php/AAAI/article/view/5388.

Steven E Shladover. Connected and automated vehicle systems: Introduction and overview. Journal of Intelligent Transportation Systems, 22(3):190–200, 2018.

David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. Mathematical Programming, 62(1-3):461–474, 1993.

Andrea Simonetto, Julien Monteil, and Claudio Gambella. Real-time city-scale ridesharing via linear assignment problems. Transportation Research Part C: Emerging Technologies, 101: 208–232, 2019.

Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. Science, 327(5968):1018–1021, 2010. ISSN 0036-8075. doi: 10.1126/science.1177170. URL https://science.sciencemag.org/content/327/5968/1018.

Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. The benefits of meeting points in ride-sharing systems. Transportation Research Part B: Methodological, 82:36–53, 2015.

Alexander Sundt, Qi Luo, John Vincent, Mehrdad Shahabi, and Yafeng Yin. Heuristics for customer-focused ride-pooling assignment. arXiv preprint arXiv:2107.11318, 2021.

Amirmahdi Tafreshian, Neda Masoud, and Yafeng Yin. Frontiers in service science: Ride matching for peer-to-peer ride sharing: A review and future directions. Service Science, 12(2-3):44–60, 2020.

Xiaocheng Tang, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. A deep value-network based approach for multi-driver order dispatching. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1780–1790, 2019.

TLC. Nyc taxi and limousine commission trip record data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page, 2021. Accessed: 2021-05-02.

Yongxin Tong, Jieying She, Bolin Ding, Lei Chen, Tianyu Wo, and Ke Xu. Online minimum matching in real-time spatial data: experiments and analysis. Proceedings of the VLDB Endowment, 9(12):1053–1064, 2016.

Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, Jieping Ye, and Ke Xu. A unified approach to route planning for shared mobility. Proceedings of the VLDB Endowment, 11(11):1633–1646, 2018.

Hai Wang and Hai Yang. Ridesourcing systems: A framework and review. Transportation Research Part B: Methodological, 129:122–155, 2019.

Zizhuo Wang, Peter Glynn, and Yinyu Ye. Likelihood robust optimization for data-driven problems, 2014.

Zizhuo Wang, Peter W. Glynn, and Yinyu Ye. Likelihood robust optimization for data-driven problems. Computational Management Science, 13(2):241–261, 2016. doi: 10.1007/s10287-015-0240-3. URL https://doi.org/10.1007/s10287-015-0240-3.

Qinshuang Wei, Ramtin Pedarsani, and Samuel Coogan. Mixed autonomy in ride-sharing networks. IEEE Transactions on Control of Network Systems, 7(4):1940–1950, 2020.

David P Williamson and David B Shmoys. The design of approximation algorithms. Cambridge university press, 2011.

Jiaohong Xie, Yang Liu, and Nan Chen. Two-sided deep reinforcement learning for dynamic mobility-on-demand management with mixed autonomy. Transportation Science, 2023.

X. Xie, W. Ma, Y. Chen, and Y. Zheng. Geolife2.0: A location-based social networking service. In 2013 IEEE 14th International Conference on Mobile Data Management, pages 357–358, Los Alamitos, CA, USA, may 2009. IEEE Computer Society. doi: 10.1109/MDM.2009.50. URL https://doi.ieeecomputersociety.org/10.1109/MDM.2009.50.

Hai Yang, Xiaoran Qin, and Jintao Ke. Modelling and optimizing the real-time matching processes in a ride-sourcing market. In Transportation Systems in the Connected Era-Proceedings of the 23rd International Conference of Hong Kong Society for Transportation Studies, HKSTS 2018, page 589, 2018.

Hai Yang, Xiaoran Qin, Jintao Ke, and Jieping Ye. Optimizing matching time interval and matching radius in on-demand ride-sourcing markets. Transportation Research Part B: Methodological, 131:84–105, 2020.

Josh Jia-Ching Ying, Eric Hsueh-Chan Lu, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S. Tseng. Mining user similarity from semantic trajectories. In Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks, LBSN '10, page 19–26, New York, NY, USA, 2010. Association for Computing Machinery. ISBN

9781450304344. doi: 10.1145/1867699.1867703. URL https://doi.org/10.1145/1867699.1867703.

Xian Yu and Siqian Shen. An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling. IEEE Transactions on Intelligent Transportation Systems, 2019.

Liteng Zha, Yafeng Yin, and Zhengtian Xu. Geometric matching and spatial pricing in ride-sourcing markets. Transportation Research Part C: Emerging Technologies, 92:58–75, 2018.

Yu Zheng. Trajectory data mining: An overview. ACM Trans. Intell. Syst. Technol., 6(3), May 2015. ISSN 2157-6904. doi: 10.1145/2743025. URL https://doi.org/10.1145/2743025.