VEHICLE DISPATCH IN HIGH-CAPACITY SHARED AUTONOMOUS MOBILITY-ON-DEMAND SYSTEMS

By

CHENG LI

A thesis submitted to
the University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
January 2022

ABSTRACT

Ride-sharing is a promising solution for transportation issues such as traffic congestion and parking land use, which are brought about by the extensive usage of private vehicles. In the near future, large-scale Shared Autonomous Mobility-on-Demand (SAMoD) systems are expected to be deployed with the realization of self-driving vehicles. It has the potential to encourage a car-free lifestyle and create a new urban mobility mode where ride-sharing is widely adopted among people. This thesis addresses the problem of improving the efficiency and quality of vehicle dispatch in high-capacity SAMoD systems.

The first part of the thesis develops a dispatcher which can efficiently explore the complete candidate match space and produce the optimal assignment policy when only deterministic information is concerned. It uses an incremental search method that can quickly prune out infeasible candidates to reduce the search space. It also has an iterative re-optimization strategy to dynamically alter the assignment policy to take into account both previous and newly revealed requests. Case studies of New York City using real-world data shows that it outperforms the state-of-the-art in terms of service rate and system scalability. The dispatcher developed in this part can serve as a foundation for the next two parts, which consider two kinds of uncertain information, stochastic travel times and the dynamic distribution of requests in the long-term future, respectively.

The second part of the thesis describes a framework which makes use of stochastic travel time models to optimize the reliability of vehicle dispatch. It employs a candidate match search method to generate a candidate pool, uses a set of preprocessed shortest path tables to score the candidates and provides an assignment policy that maximizes the overall score. Two different dispatch objectives are discussed: the on-time arrival probabilities of requests and the profit of the platform. Experimental studies show that higher service rates, reliability and profits can be achieved by considering travel time uncertainty.

The third part of the thesis presents a deep reinforcement learning based approach to optimize assignment polices in a more far-sighted way. It models the vehicle dispatch problem as a Markov Decision Process (MDP) and uses a policy evaluation method to learn a value function from the historic movements of drivers. The learned value function is employed to score candidate matches to guide a dispatcher optimizing long-term objective, and will be continually updated online to capture the real-time dynamics of the system. It is shown by experiments that the value function helps the dispatcher to yield higher service rates.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. David Parker. Without his encouragement, support, guidance and enthusiasm, this thesis would never have been completed. The second thanks are given to Prof. Jeremy Wyatt, for guiding me along on the start of my research journey. I would also like to thank Prof. Qi Hao for his support along with the journey and the thesis group members, Dr. Peter Hancox and Dr. Mohan Sridharan, for their helpful discussions and advice. Lastly, I want to thank my parents for their unconditional support and love, and my friends for their companionship.

Contents

List of Figures

List of Tables

Acronyms

A2C Advantage Actor-Critic. 34

ADP Approximate Dynamic Programming. 33, 34, 110

ALNS Adaptive Large Neighborhood Search. 26

CMCA Cerebellar Model Arithmetic Computer. 33

DP Dynamic Programming. 48

DQN Deep Q-Network. 33, 34, 110

DVD Deterministic Vehicle Dispatch. 92, 94, 95, 97, 100, 103, 105

FCFS First-Come-First-Served. viii, 19–22, 28, 35, 38, 39, 59, 68

GI Greedy Insertion. 59, 60, 66, 68

ILP Integer Linear Program. 23, 30, 34, 56, 57, 78, 79, 86, 87, 90, 124

LSTM Long Short-Term Memory. 119

LTO Long-Term Optimization. 125, 127, 128, 130–132, 134, 136, 137

MDP Markov Decision Process. ii, 32, 33, 111, 112, 114, 117

MILP Mixed Integer Linear Program. 23, 25, 38

MIP Mixed Integer Program. 23, 28

MoD Mobility-on-Demand. 1–3

MPC Model Predictive Control. 32

OPE Offline Policy Evaluation. 107, 110, 115, 116, 118, 120

OSP Optimal Schedule Pool. 37, 57, 59, 60, 63, 66–68, 77, 78, 90, 103, 126–128, 130–132, 136, 137

P2P Peer-to-Peer. 14, 16, 19, 22, 28

PVD Profit-aware Vehicle Dispatch. xiii, 73, 74, 87, 89, 92, 94, 95, 97, 100, 103, 105

RL Reinforcement Learning. 111, 112, 125

RNN Recurrent Neural Network. 28

RTV Request Trip Vehicle graph. 59, 60, 63, 66–68, 77, 103

RVD Reliability-aware Vehicle Dispatch. 73–75, 87, 89, 92, 94, 95, 97, 100, 103, 105

SAMoD Shared Autonomous Mobility-on-Demand. i, 1, 3, 4, 11, 14, 16, 17, 19, 20, 28, 37–40, 46, 55, 58, 68, 70, 107, 108, 110, 111, 113, 114, 117, 119, 128, 138, 139, 141, 142

SBA Single-request Batch Assignment. 59, 60, 66–68

SSP Stochastic Shortest Path. 24, 25, 79, 80, 82

TD Temporal Difference. 32, 116, 118–121

VMT Vehicle Miles Travelled. 2, 3, 5

VRP Vehicle Routing Problem. 14, 25, 26, 28

VRPPD Vehicle Routing Problem with Pickup and Deliver. 15

Chapter One

Introduction

This thesis studies the optimization of vehicle assignment policies in high-capacity Shared Autonomous Mobility-on-Demand (SAMoD) systems at city scale, where thousands of centrally routed self-driving vehicles transport passengers using ride-sharing in an urban environment. Unlike current Mobility-on-Demand (MoD) systems (e.g., Uber, Lyft, DiDi and Grab), SAMoD systems use autonomous vehicles, which are amenable to centralized control, to enable deployment of fleet-wide policies and to provide lower operational costs than employing human drivers [1–4]. Additionally, through ride-sharing, SAMoD systems are transforming urban transportation in a more environmentally-friendly and more economically-feasible way [5–9].

1.1 Background

1.1.1 Trends in Personal Urban Mobility

Private automobiles have become the dominant form of the mobility landscape by providing fast travel and considerable personal convenience. However, the extensive usage of private vehicles also causes some serious issues: traffic congestion, energy consumption, parking land use, car accidents and exhaust pollution. In 2017, the loss of urban residents in rich countries is almost $1000 while sitting in traffic [10]. Thus, the private vehicle based transportation system is widely acknowledged as unsustainable and a new form of urban mobility is needed.

Taxis provide a relatively convenient transportation choice when people do not have a car or it is inconvenient to drive. With the large-scale popularization of smartphones, MoD systems have been introduced to transform the taxi industry and people can hail a taxi in a more efficient way. MoD systems provide passengers with on-demand and reliable transportation by pairing them with idle vehicles via mobile apps. Passengers input their origin and destination locations, and the systems immediately assign the nearest vehicles to pick them up in a short time. By providing a flexible transportation choice to passengers, MoD systems are shifting "vehicle as a product" to "vehicle as a mobility service".

Although ride-hailing in MoD systems is potentially more efficient than street-hailing and is expected to reduce parking requirements, there is evidence from studies that current MoD systems have increased motorised traffic and congestion [11–15]. Besides the substitution for other car trips, current MoD systems actually add more new cars to the road and increase the Vehicle Miles Travelled (VMT) of the city. Yet up to 50% of ride-hailing VMT in New York is wasted due to unoccupied driving. Moreover, since most drivers live outside the city, their commutes into the city and back home may add more VMT and increase congestion [14]. Another shortcoming commonly observed in current MoD systems is the low vehicle occupancy rate. According to an experiment on ride-hailing rides in Denver, the capital of Colorado, the distance weighted average passenger per ride is 1.3 and would drop down to approximately 0.8 when accounting for unoccupied driving miles [12]. The vehicle occupancy rate of ride-hailing is key to sustainability and ride-sharing (i.e., carrying two or more passengers in one trip) is encouraged to increase it [13]. It is found that a 1% increase in the occupancy rate would lead to around a 1.15% decrease in VMT [15].

A recent study in Manhattan shows that up to 80% of taxi trips can be pairwise shared with very little increase in travel time, which translates into a 40% reduction of the taxi fleet [7]; the effectiveness of ride-sharing is also suggested with sparse taxi fleets and is later validated in multiple cities [16]. In addition, it is estimated in Lisbon that full-scale shared mobility deployment would reduce VMT and CO_2 emissions by up to 50% and reduce parking needs by up to 95% [13]. But the tremendous growth potential for ride-sharing services brings the challenge of expanding the supply scale of MoD systems [3].

Recently, research on autonomous driving has attracted a lot of attention both from academia and industry. Autonomous vehicles can free humans from tedious driving and increase transportation safety. While most of the research has been done on operating a single vehicle running on highways, less attention has been paid to how autonomous driving will affect urban mobility. As autonomous vehicles can increase traffic efficiency by enabling system-wide coordination and rebalancing themselves, they hold great promise for shared mobility [1]. Fully self-driving vehicles would also reduce the cost of taxi trips, by saving on drivers' wages [5, 13]. An estimate in Zurich, Switzerland shows a reduction of around 85% [17]. This reduction is around 50% even if we reduce drivers' salaries to one-fifth.

1.1.2 Shared Autonomous Mobility-on-Demand

Using self-driving vehicles to provide personal on-demand mobility in ride-sharing trips is a transformational technology that is emerging now [18–20]. In SAMoD systems, a predetermined number of multi-occupancy vehicles are dispatched from multiple depots at the beginning of the day to satisfy on-demand requests on a road network. The requests are revealed throughout the day and their arrival times and locations are uncertain. To alleviate the low utilization of available seats in current MoD systems, passengers that travel in similar directions are merged into a ride-sharing trip and served by a single vehicle. Employing ride-sharing service can serve more requests and reduce passenger costs, but at the cost of user experience. To ensure the quality of service, constraints are placed on maximum request waiting times and detour times. Other constraints, such as passenger gender, age, pet allowance and social tendencies, may also be considered for tailor-made matching.

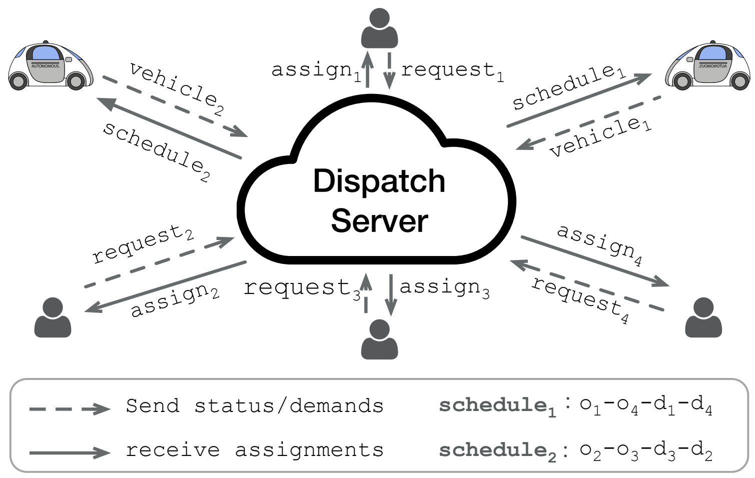


Figure 1.1: Architecture of an SAMoD system with two vehicles and four requests, where vehicle 1 is assigned to serve requests 1 and 4 following schedule 1, and vehicle 2 is assigned to serve requests 2 and 3 following schedule 2.

Figure 1.1 shows the structure of an SAMoD system, the key component of which is a central dispatch server that controls the system in a rolling horizon manner. At each dispatch window (also known as batch window, batch period or epoch), vehicles update their status and passengers submit their transportation demands to the dispatch server, as depicted by broken arrows. The dispatch server will then compute feasible ride-sharing schedules for all vehicles, i.e., satisfying the constraints discussed above. Each schedule indicates a candidate match between a vehicle and a request (or a group of requests that can be served together with minimum detours), and represents the order for the vehicle to pick up and drop off passengers (e.g., $o_1 - o_4 - d_1 - d_4$: pick up passenger 1, pick up passenger 4, drop off passenger 1, drop off passenger 4). Taking all candidate schedules into consideration, the dispatcher server assigns each vehicle a particular schedule in a cooperative way to optimize the objective of the system. Optimization objectives can include number of served requests, the profit of the platform, overall VMT and average request wait time. After receiving the assignments, vehicles will follow the schedules serve passengers. Vehicles can continue to be matched to new requests even if they have passengers onboard, as long as the maximum detour constraint is satisfied and there are seats available. If requests cannot be immediately matched to vehicles, they may choose to wait for another dispatch round or leave the system.

A centralized dispatcher is easy to deploy and is capable of enabling system-wide coordination to reach a global optimum, i.e., utilizing every available seat of the vehicle fleet to serve as many passengers as possible. Compared to decentralized approaches, although centralized optimization may face a computational burden when the number of passengers and vehicles is high, it is computationally feasible at a city-level, e.g., serving 400k passengers a day in Manhattan. Furthermore, the successful deployment of large-scale ride-hailing systems (e.g., Uber, DiDi and Grab) has demonstrated that centralized optimization is preferred [21–23].

1.2 Research Problem and Questions

In this thesis, we study the ride-sharing dispatch problem for high-capacity vehicles, which concerns computing matching policies between vehicles and passengers at the city scale, where thousands of vehicles and hundreds of thousands of passengers are involved, with the primary goal of maximizing the service rate (i.e., percentage of requests served). High-capacity means that the vehicle has a seating capacity more than 4. Following the general setup in the literature [7, 19, 20, 24–33], the general assumptions and constraints made about the dispatch problem are: (1) The dispatcher controls a homogeneous fleet; (2) The number of passengers included in each request is one; (3) A vehicle can be continuously matched to multiple requests, as long as the number of passengers on board does not exceed its capacity; (4) A request can only be matched to a vehicle when two constraints on maximum waiting time and total travel delay are satisfied, such a match is called a feasible match; (5) Once a match is assigned, both the vehicle and the passenger(s) will accept and complete it; (6) If a request cannot be matched it will wait for five epochs before walking away; (7) The dispatcher cannot reject any requests, unless there is no vehicle available.

The assumptions and constraints outlined above are intended to focus the development of the algorithm on the generic characteristics of the optimization for the dispatch problem. The algorithms developed based on them can be extended to more complex assumptions and constraints. For example, extending the second assumption to support the scenario where a request contains multiple passengers requires only one additional variable indicating the number of passengers, which defaults to one in this thesis. In practice, a passenger may reject a match or cancel the request when the vehicle is traveling toward the pickup location. In the industry, a model to estimate the the likelihood of acceptance and/or cancellation is trained separately and incorporated within the object function of the optimization problem [34, 35], so as to relax the fifth assumption. Such an adaptation also applies to the algorithms developed in this thesis.

To illustrate the dispatch problem, we present a motivating example. Imagine that we have 100 four-seat vehicles operating for one hour and receive 10 requests every 30 sec, i.e., the dispatch window is 30 sec; our goal is to fulfill as many of these 1200 requests as possible. At each dispatch epoch, we have 30 sec to compute a matching policy that assigns the received requests to the vehicles. To put it intuitively and simply, the matching policy is produced in two parts: the search for candidate matches that computes the possible options for each vehicle that match it to any no more than 4 out of 10 passengers, and the selection of candidate matches that assigns each vehicle one of its options to maximize the total number of matched passengers and ensure that the options assigned to the 100 vehicles are conflict-free, e.g., no passenger is matched to two vehicles.

Because of the real-time requirements of the dispatch problem (i.e., the matching policies need to be ready within the dispatch window, which in reality is generally no greater than 30 sec), it is difficult to find all options (i.e., candidate matches) when the capacity of vehicles and the number of requests are large. If the dispatcher does not know all the options, then there is no guarantee of the quality of the decisions made (i.e., matching policies), and there is room for improvement. In addition, the assessment of the options will also affect the quality of the decision, especially when there is uncertain information involved, e.g., stochastic travel time and demand.

This thesis works to improve the efficiency and quality of high-capacity vehicle dispatch. Specifically, we focus our attention on three problems: (1) efficient and optimal dispatch, (2) reliability-aware dispatch and (3) long-term service optimization. The first problem concerns efficiency, by tackling at each epoch the complete search of the candidate match space in real-time, to compute the optimal matching policy. The term "optimal" implies that, for the given inputs, the output matching policy is the best regardless of the computation time, e.g., having the highest service rate, with no chance of improvement. Taking the motivating example mentioned above, assuming that the optimal outcome is to serve 1100 out of 1200 passengers, i.e., a service rate of 91.67%, it means that there is no possibility for the dispatcher to increase the number of passengers it can serve to 1101 if it has no more new information (e.g., the distribution of future requests) as input at each dispatch epoch than the 10 new requests it receives. The latter two concern quality, by tackling the evaluation of candidate matches when considering uncertainty information (i.e., stochastic travel times and future requests), to produce more sophisticated dispatch policies.

1.2.1 Efficient and Optimal Dispatch

If the dispatcher knows all possible candidate matches, the optimal match policy can be obtained by solving a maximum (or minimum) matching problem, which is easy to solve using existing algorithms or commercial software. In the single-occupancy vehicle dispatch problem, let n and m denote the number of requests and vehicles at each epoch, respectively, there are only $n \cdot m$ possible candidate matches, We can easily traverse them to find the feasible ones, then use the Hungarian maximum matching algorithm [36] to get the optimal assignment policy. However, in ride-sharing, the number of possible matches grows exponentially with vehicle capacity and the number of requests, making it difficult to compute the optimal match policy in real-time. In terms of the motivating example mentioned above with a 100-vehicle fleet, the number of all possible candidate matches can be up to $100 \cdot ({}^{10}C_1 + {}^{10}C_2 + {}^{10}C_3 + {}^{10}C_4) = 38,500$, whereas when ride-sharing is not considered, that number is only $100 * 10 = 1000$. Moreover, when using high-capacity vehicles, a new problem called scheduling has also emerged, which concerns the ordering of picking and dropping requests for candidate matches. The scheduling problem also suffers from a large search space, if it is not properly tackled, we may mistakenly reject some feasible candidate matches and miss the optimal assignment policy. Thus comes the first research question:

After receiving a batch of requests at each epoch, how can we find all possible candidate matches in real-time and compute the optimal match policy that maximizes the service rate(**Question 1**)*?*

Intuitively, the matching policy would be better if multiple orders could be considered together. However, considering the passenger waiting time, we have to make the assignment at the current epoch when receiving the requests. But when receiving new requests, there are still passengers in the previous epochs who have not yet boarded, which gives us the possibility for re-optimization, i.e., to consider the previous and new requests together to compute the matching policy. Once again, the re-optimization suffers from an explosion of the search space, since the number of passengers being considered simultaneously may increase a lot. Consider the motivating example mentioned above, where the number of requests being considered per epoch is 10, but when re-optimization is enabled, this number may grow to 40-50. Thus comes the second research question:

When the dispatcher receives a batch of new requests, how can we efficiently alter the current matching policy and keep it optimal for all requests **(Question 2)***?*

1.2.2 Reliability-Aware Dispatch

In urban transportation, in addition to travel cost, the most important considerations for passengers also include waiting times and delays caused by ride-sharing. In particular, travel time uncertainty has a significant impact on the effectiveness of ride-sharing dispatch [37]. In some unfortunate cases, a feasible match and routing policy may become infeasible due to a deviation in travel time, which means that passengers may arrive at their destinations exceeding the delay constraint. This will create a bad experience for passengers and reduce their willingness to take ride-sharing trips [38]. In terms of the motivating example mentioned above where 10 passengers are received per epoch, if passengers are always delivered after the delay constraints have been exceeded, we may only receive 5 requests per epoch afterwards. Therefore, in addition to chasing the service rate, we also need to seek the reliability of service, i.e., not exceeding the delay constraint, to increase the willingness of passengers to participate in ride-sharing. Formally, service quality is measured as the probability of on-time arrival. Thus comes the third research question:

If a stochastic travel time model is given, how can we incorporate it into the ride-sharing dispatch problems to optimize the reliability of service (i.e., the on-time arrival prob-

abilities of passengers) **(Question 3)***?*

Another way to increase the willingness of passengers to participate in ride-sharing is to offer compensation to increase tolerance of passengers for late arrivals. In this case, we need to consider the profit of the platform, in addition to the service rate. Thus comes the fourth question:

If passengers accept late arrival when compensation is applied, how can we optimize the profit of the platform **(Question 4)***?*

1.2.3 Long-Term Service Optimization

The "optimal dispatch" discussed earlier can enables a "global" optimization for each dispatch epoch, but is short-sighted in nature, since only known requests are taken into account. This problem is called reactive dispatch. Because of the lack of future information in the following batch windows, an "optimal" assignment decision at present may cause potential efficiency losses in the future [23, 39]. The most straightforward solution is to incorporate predicted future requests into the candidate search procedure, but this would make the problem more complex and may not be solvable in real-time, with the same issues as the example given in the discussion of re-optimization. A more suitable approach for deployment is to learn the longer-term impact of candidate matches to guide the dispatch. Thus comes the fifth research question:

Given historical data and a simulation environment, how can we learn the longer-term impact of matching policies and make more far-sighted decisions without increasing the computational complexity of online matching **(Question 5)***?*

1.3 Contributions

The objective of this thesis is to develop algorithms that can make vehicle dispatch in SAMoD systems more efficient, reliable and far-sighted. By addressing the research questions discussed in Section 1.2, the contributions of this thesis include:

1. A method to efficiently compute all feasible candidate schedules in real-time, so that we can get an optimal allocation and scheduling policy that maximizes the service rate, based on the information collected in the dispatch window. (**Question 1** in Section 1.2.1, Chapter 3.)

2. A re-optimization strategy to dynamically alter the assignment policy when receiving new requests, so that we can always have an optimal matching policy up to date, where all known requests are considered. (**Question 2** in Section 1.2.1, Chapter 3.)

3. A method to leverage stochastic travel time models to compute the reliability scores of the schedules, so that we can, while chasing the service rate, route the vehicles to serve requests with the highest on-time arrival probabilities. (**Question 3** in Section 1.2.2, Chapter 4.)

4. A method to consider the penalty costs due to late arrivals under travel time uncertainty, so that we can, while chasing the service rate, route the vehicles to the most profitable trips to optimize the profit of the platform. (**Question 4** in Section 1.2.2, Chapter 4.)

5. A reinforcement learning based method to predict long-term impact scores of the candidate matches, so that we can dispatch vehicles to optimize the overall service rate over a long horizon, instead of the current dispatch epoch. (**Question 5** in Section 1.2.3, Chapter 5.)

6. Case studies to validate the above methods and provide a deeper understanding of the impact of parameter tuning in vehicle dispatch in real world scenarios. We contribute two open-source repositories on GitHub for case studies: AMoD (https://github.com/Leot6/AMoD) and AMoD2 (https://github.com/Leot6/AMoD2).

1.4 Publications Resulting from the Thesis

Some parts of this thesis are contained in the following publications. Paper [1] is associated with Chapter 3 and paper [2] is associated with Chapter 4. Besides, paper [3] (associated with Chapter 5) is to appear in IROS 2022 in October.

[1] C. Li, D. Parker and Q. Hao, "Optimal Online Dispatch for High-Capacity Shared Autonomous Mobility-on-Demand Systems," 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 779-785.
(DOI: 10.1109/ICRA48506.2021.9561281.)

[2] C. Li, D. Parker and Q. Hao, "Vehicle Dispatch in On-Demand Ride-Sharing with Stochastic Travel Times," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 5966-5972.
(DOI: 10.1109/IROS51168.2021.9636499.)

[3] C. Li, D. Parker and Q. Hao, "A Value-based Dynamic Learning Approach for Vehicle Dispatch in Ride-Sharing" 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2022. (To appear in October.)

1.5 Thesis Structure

The remaining content of the thesis is arranged as follows. In Chapter 2, we review existing research work on the research topics of efficient vehicle-request matching, ride-sharing under travel time uncertainty and predictive vehicle dispatch. The purpose is to identify relevant methods and gaps within the field of research and how this thesis contributes to it. In Chapter 3, we study the problem of computing the optimal assignment policy for high-capacity ride-sharing in real-time, and develop a dispatcher that is able to dynamically keep the optimal assignment policy updated when new requests are received. In Chapter 4, we consider the impact of travel time uncertainty for ride-sharing, and propose a multi-phase dispatch scheme to optimize the assignment policy considering requests' probabilities of on-time arrival and the platform's profit. In Chapter 5, we turn our attention to long-term service optimization, and present a value-based dynamic learning solution for vehicle dispatch in ride-sharing. In Chapter 6, we draw the conclusions of this research and highlight several directions for future work.

Chapter Two

Literature Review

This chapter gives a synoptic review of the literature related to the topics of this thesis, which primarily concerns the following characteristics of the approaches reviewed: large-scale (i.e., the number of vehicles dispatched is in the thousands), high-capacity (i.e., the capacity of the vehicle is greater than four), real-time (i.e., the computation time of the matching policy is shorter than the dispatch window) and optimal (i.e., the output matching strategy cannot be further improved for a given input, regardless of the computation time). Specifically, the review includes three research topics. First, Section 2.1 summarizes studies focusing on efficiently finding a good matching policy between vehicles and requests, including approaches in P2P ride-sharing and SAMoD. Although P2P ride-sharing is a little different from SAMoD in terms of its problem definition, the algorithms in P2P have a lot in common with the algorithms in SAMoD, which could provide some reference and inspiration. Second, Section 2.2 presents approaches that take travel time uncertainty into account to optimize the reliability of matching and routing. As there is a paucity of literature on ride-sharing, we also investigate some work on the Vehicle Routing Problem (VRP) for insights into the role of stochastic travel times. Third, Section 2.3 reviews proactive solution methods that aim to optimize the long-term effectiveness of dispatch. Depending on how future information is considered in assignment, they are classified into two groups: forecast based approaches and value based approaches. Finally, Section 2.4 summarizes this chapter.

2.1 Efficient Vehicle-Request Matching in Dynamic Ride-Sharing

The dynamic ride-sharing problem can be viewed as a special case of the dynamic Vehicle Routing Problem with Pickup and Deliver (VRPPD), which involves scheduling and routing a fleet of vehicles with a given capacity to satisfy transportation requests at different locations [40]. The dynamic setting means that people can send transportation requests at any time and the input data of the problem are not all known in advance, but revealed over time. Therefore, a solution to the problem must be capable of using the revealed information to continuously update the schedules and routes as time goes by. There are two main types of solutions: exact methods and heuristic methods. Exact methods attempt to repeatedly solve an optimal static problem with newly revealed requests [41, 42]. Finding optimal solutions is normally too time consuming and inadequate for real-time deployments. Heuristic methods choose to update the current solution by inserting new requests and interchange moves [43, 44]. By compromising a little on the solution objective, heuristic methods achieve better efficiency for deployment in real-time. The quality of an algorithm is typically measured by the proportion of served requests (i.e., the service rate), the travel distance (or time) of the vehicles and the computation time of the algorithm [45, 46].

The dynamic ride-sharing problem differs from the general VRPPD in that it focuses on transporting passengers and concerns tight constraints on the quality of service, such as door-to-door transportation, maximum travel detour and tight time windows [40]. Depending on whether the vehicles (drivers) are treated as one-time private independent entities like requests (riders) or long-time continuous service providers, the problem can be classified as

P2P ride-sharing or SAMoD. In terms of the motivating example mentioned in Chapter 1, P2P ride-sharing refers that the fleet size is no longer a fixed 100, but varies according to the number of drivers to participate.

In terms of the example in the previous chapter, carpooling means that the fleet size is no longer a fixed 100, but varies according to the number of drivers involved

2.1.1 Peer-to-Peer Ride-sharing

P2P ride-sharing systems work on establishing ride-sharing matches between drivers and riders. The system objectives are maximizing the number of satisfied participants (both drivers and riders) [47] and minimizing system-wide vehicle-miles and travel time [40, 48]. Drivers in P2P ride-sharing have their own travel plans with distinct origin and destination locations. They are willing to share the available seats during the trips to split the cost and will normally leave the system after arriving at their own destination. Therefore, the vehicles have narrow travel time windows, different vehicle types and are not available for later ride requests [49].

Agatz et al. [40] provide a survey discussing the challenges faced in dynamic P2P ride-sharing problems and suggesting that researchers work on decomposing the problem. Based on whether a vehicle can be shared by a single rider or multiple riders and whether a rider is willing to travel with a single vehicle or multiple vehicles by transfer, P2P ride-sharing problems can be distinguished into four variants: one-to-one, one-to-many, many-to-one, and many-to-many [40, 49].

One-to-One

In one-to-one matching, a single driver is only matched with a single rider, and vice versa. The capacity of the vehicles is set to one and at most two more stops are allowed for each vehicle, one for pickup and one for drop-off. For a driver-rider match, the travel cost savings can be calculated by the sum of the cost of two separate trips minus that of the sharing trip. The objectives of maximizing the number of matches and minimizing the travel cost can be converted to maximizing the total travel cost savings. Since the routing problem in one-to-one matching is quite simple, the literature mainly focuses on decomposing the matching to achieve high computational efficiency. Najmi et al. [50] introduce a clustering heuristic, based on k-means and using spatial coordinates, to divide drivers and riders into small groups. As a result, the original matching problem is decomposed into a set of smaller sub-problems and can be solved more efficiently. Using a similar approach, Ketabi et al. [51] apply spectral clustering to generate meaningful spatio-temporal clusters. They propose a similarity measure to calculate both spatial and temporal proximities between two trips, using the average of the point-wise distances. Unlike others, Tafreshian et al. [48] use a dissimilarity measure to develop a graph partitioning methodology. They use a bipartite graph to represent the one-to-one matching problem and partition the graph into smaller sub-graphs to reduce the overall computational complexity. Because searching for candidates can be done easily in one-to-one matching, the above literature is devoted to further improve the efficiency of selecting candidates, which is not the concern in this thesis. When the scale of the instance considered in SAMoD becomes much larger, say tens or hundreds of thousands of vehicles are dispatched, the partitioning methods mentioned may serve as a reference.

One-to-Many

In one-to-many matching, a single driver can serve multiple riders, as they may have sufficient time flexibility. Allowing more stops and detours to be made during a trip, a vehicle would have a longer schedule and a more complex routing problem. Herbawi et al. [52] adapt the model of pickup and delivery problem with time windows and propose a genetic algorithm. They use Solomon's insertion method [53] to create initial solutions and improve the solution with a crossover operator and five mutation operators. Xia et al. [54] present two heuristic methods, one based on Tabu search (smart interchange) and another based on simulated annealing (correlates to natural processes). They initially generate carpooling routes by randomly selecting rides and then try to refine them by searching for neighbouring solutions. However, these evolutionary approaches are normally not capable of converging to a good matching policy in a short time. The computation time can be up to hours. Thus, they are not applicable to large-scale or high-capacity problems. Borrowing the method from [55], Ma et al. [56] propose a recursive algorithm to compute feasible rider sets. Riders that can be served by a single vehicle are grouped together and the one-to-many matching is simplified to one-to-one matching. A delete operator is then performed to reduce the size of the decision region to speedup the model solving. With a small sacrifice in system-wide performance, they can provide matching solutions with good efficiency. Although more efficient than the previous two evolutionary approaches, it still cannot handle large-scale instances in real-time. The computation time for an instance with 1000 double-seat vehicles and 2000 rides is over 2 min. Moreover, optimality is not ensured.

Many-to-One and Many-to-Many

In many-to-one matching and many-to-many matching, riders are allowed to switch between different vehicles. In this setting, riders may sacrifice some convenience to increase match success rates. Riders and drivers can be matched even when they do not share similar directions [57]. These two matching problems are normally considered together and are called a multi-hop ride-sharing problem [49]. Herbawi et al. [58] consider that drivers have fixed routes and use these routes to form a time-expanded graph. They formulate the multi-hop ride-sharing problem as a multi-objective shortest path finding problem, and propose an evolutionary algorithm to minimize the cost, time and number of transfers. Masoud et al. [47] present a binary problem formulation in a time-expanded network. They attempt to decompose the problem into a set of passenger matching problems and solve them iteratively. In each iteration, the sub-problems are solved independently and merged into groups if any conflicting matches are found among them. When there is no conflict, the algorithm converges and outputs a feasible solution. For real-time deployment, Masoud et al. [59] consider matching riders in a First-Come-First-Served (FCFS) manner. A pre-processing procedure is proposed to reduce the search space of each rider, by restricting potentially available drivers in a narrow elliptical area. They then introduce a ride exchange mechanism to further improve the number of matches, by alleviating the negative impact of the FCFS approach. Allowing transfer significantly increases the complexity of matching, thus the existing approaches are focused on computational efficiency. Although transfer is not considered in this thesis, the literature above may serve as a reference for public transit interaction with SAMoD, which is a future direction.

2.1.2 Shared Autonomous Mobility-on-Demand

SAMoD systems work on dispatching a fleet of autonomous vehicles to travel around the city to transport passengers. The key component is assigning suitable vehicles to serve requests. Unlike P2P ride-sharing, the number of vehicles is normally predetermined and all of the vehicles are fully controlled by a central planner to maximize the overall objectives coordinately, e.g., the number of satisfied requests [19] and the total profit of the platform [60]. The vehicles are available during the entire operation time and they can wait at particular locations or be rerouted to different destinations. In some work, this may be called a shared-taxi problem [8]. SAMoD systems are initially studied with unit capacity vehicles and then extended to allow passengers travelling in ride-sharing with multiple capacity vehicles.

Unit Capacity Vehicle

Lee et al. [61] study a unit capacity taxi dispatch system with the simplest assignment heuristic. On an FCFS basis, they put the incoming requests in a queue and find the nearest vehicle for each request. Seow et al. [62] argue that assigning requests in an FCFS fashion ignores the global optimality, as it merely increases individual passenger satisfaction and the effects of the assignment on other awaiting passengers in the queue are not considered. They propose a multi-agent collaboration architecture to consider the match between vehicles and requests in a cooperative way. But their solution is limited to a local optimal with a small number of vehicles due to communication burdens [63]. Zhang et al. [35] present a batch assignment approach to simultaneously match all available vehicles and requests. The vehicle-request matching problem is formulated as a combinatorial optimization problem and optimized over the current batch window, to maximize the probability of request acceptance. A driver-select-order dispatch mode is initially used in [35] and switched to a centralized platform-assign-order dispatch mode in [64]. The switch brings a significant improvement (over 10%) on the request completion rate. Within a batch window, every potential match between a vehicle and a request is represented by an edge, weighted by a desired objective, to generate a bipartite graph. An optimal matching policy can then be found by solving a weighted bipartite matching problem. The results from the literature above suggest that batch assignment is preferred over FCFS, as considering multiple requests simultaneously can output the optimal solution within the batch window.

Multiple Capacity Vehicle

For increasing efficiency in transportation, a natural option is to enable ride-sharing. Due to the high computational complexity, it is intuitive to choose FCFS matching methods to give a low algorithmic response time. Widdows et al. [65] present a real-time ride-sharing service that assigns one request at a time to the best-matched vehicle. For each request, the dispatcher considers the nearest candidate vehicles, enumerates all possible travel schedule insertions and scores each of them using some predefined features, e.g., pickup travel time and overlap between requests. According to the score ranking, the request is assigned to the candidate vehicles in order until it is accepted.

Ma et al. [24] introduce a grid-based index to accelerate the search for candidate vehicles, where a request only checks its nearby areas. Both the spatio-temporal factors on the origin and destination locations of the requests are considered to reduce the search space by pruning out the obviously infeasible vehicles. Zhu et al. [25] try to do the candidate search from the aspect of vehicles. They propose a limited potential search area based algorithm to accelerate the search. Requests out of the area are considered to violate some predefined quality of service constraints and filtered out. This search algorithm is similar to the one used by Masoud et al. [59].

Tong et al. [26] argue that the insertion of a new request into the schedule of a vehicle is the efficiency bottleneck, and propose a dynamic programming algorithm to speed this up. It can compute the minimal increased distance route without enumerating all possible insertions. The time complexity of insertion is reduced from cubic to linear. Cheng et al. [66] propose a bilateral arrangement algorithm that tries to use a replacement procedure to achieve better match quality, by considering two requests at a time. It assigns a greedy candidate vehicle for each request and, if that vehicle is not feasible, it will attempt to replace an unpicked request in the schedule to make the assignment feasible. The time complexity remains the same as in normal FCFS based approaches. The above approaches are efficient for the real-time deployment of large fleet and even high-capacity ride-sharing systems. However, similar to the FCFS based approaches that consider unit capacity vehicles, the global optimality is ignored.

To achieve system-wide coordination, batch assignment is also introduced for multiple capacity vehicles. Incoming requests are collected over batch windows and assigned simultaneously. Potentially better performance can be achieved, but at the cost of much higher computational complexity. The simplest approach is quite similar to the unit capacity vehicle batch assignment solutions (e.g., [35, 64]), where only one-to-one matching between requests and vehicles is considered in each batch window. Simonetto et al. [27] compute all possible insertions of each individual request into the current schedule of vehicles, then find an optimal assignment of such insertions. They enforce that all new requests cannot be combined with each other but only share trips with those received during previous windows to have a low computational complexity. However, when passengers travelling in similar directions send their requests in the same batch window, the quality of matches for them are sacrificed for computational efficiency.

To allow assignment of multiple requests to one vehicle at the same time, Santos et al. [67] use a greedy randomized adaptive search procedure to develop a solution that can be used in practice. They compute a randomized initial assignment of requests to vehicles, then perform a local search for additional possible solutions to improve the quality of assignment. Jung et al. [68] propose a similar method using hybrid simulated annealing. These methods try to find the optimal assignment by randomly exploring the search space, but they cannot guarantee optimality as a maximum number of iterations is set to ensure efficiency. Moreover, similar to the evolutionary approaches in one-to-many P2P ride-sharing, they are not applicable to large-scale real-time systems.

Inspired by the concept of shareability networks, which translates the spatio-temporal matching problem into a graph-theoretic framework [7], Alonso-Mora et al. [19] propose a highly scalable algorithm. They incrementally find all possible matches between a vehicle and a clique of requests to break down the computational burden, then solve an Integer Linear Program (ILP) to match vehicles to requests in a one-to-many manner. Their method is the first to be able to dispatch thousands of high-capacity vehicles in real-time, where up to 3000 ten-seat vehicles are considered. However, some ad hoc heuristics (e.g., only 30 candidate vehicles for each request and a timeout of 0.2 s per vehicle to search feasible cliques of requests) are applied to ensure efficiency and hence the candidate state space is not guaranteed to be fully searched. To improve the candidate search procedure, Lowalekar et al. [28] introduce zone paths to assist in generating more relevant cliques of requests and Riley et al. [29] employ column generation to relax the waiting time constraint to provide service guarantees for all requests. However, none of these solutions tackles the optimal scheduling problem that finds the best orders of picking and dropping multiple requests for high-capacity vehicles, so the optimality of real-time matching is not guaranteed.

The aforementioned approaches solve the problem by decomposition. They first search for as many feasible candidates as possible, either from the aspect of requests or vehicles, and then compute the best matching policy based on the cost of the candidates. There are several other approaches based on mathematical programming. Hosni et al. [8] formulate the ride-sharing match problem as a Mixed Integer Program (MIP) and present a solution based on Lagrangian decomposition. Tsao et al. [20] develop a network flow model and formulate a Mixed Integer Linear Program (MILP) to optimize the matching policy. However, due to computational complexity, these methods are normally limited to small fleet sizes or low ride-sharing trip sizes. For example, the problem instance solved in [20] only considers 400 double-seat vehicles.

2.2 Ride-Sharing Under Travel Time Uncertainty

Besides the development of computationally efficient matching algorithms, the explicit consideration of uncertain travel time is also necessary for the widespread adoption of ride-sharing [69]. If the travel time uncertainty is not properly captured, the travel of vehicles may differ from the planned time schedules and have a negative effect on the flexibility of passengers, which in turn affects the matching rate in ride-sharing [38]. As there are too many unmeasurable traffic variables, rather than estimating a deterministic travel time, it is more valuable to predict a probability distribution over travel time to capture the uncertainty [70]. Taking the travel time distribution into account, one can plan reliable paths for vehicles or provide high quality transportation service to passengers.

2.2.1 Stochastic Shortest Path Finding

Stochastic Shortest Path (SSP) finding works as a basis for capturing the uncertainty of travel time in ride-sharing. It uses past observations of delays on roads to model travel times as probability distributions, and finds paths between two points that maximize the probability of arriving before a given deadline [71].

Assuming that the travel time of each road segment follows an independent normal distribution, Nikolova et al. [72] propose a parametric approach to find the most reliable path based on quasi-convex maximization. They project the path polytope onto a two-dimensional plane and exhaustively enumerate all extreme points of the shadow dominant by finding the parametric shortest paths. Although the optimal solution can be found, the computational complexity is superpolynomial. Nikolova [73] later constructs a fully-polynomial approximation scheme to efficiently search for relevant extreme points. Lim et al. [71] give a graphical connection between the optimal reliable path and the parametric path finding problem, which induces the idea of probe points to accelerate the search for extreme points. By checking the value of probe points, the futile part of the search space can be pruned quickly and therefore the number of parametric path findings is reduced. When compared with the running time of the algorithm in [72], the reduction is by at least a factor of ten. However, it is still impractical to deploy it in real-time city-scale path planning. Lim et al. [74] further propose a two-phase algorithm that runs in poly-logarithmic time. They develop a preprocessing procedure that computes a set of distance oracles for the road network with some approximation related parameter values. With the precomputed oracles, online querying a SSP is restricted to a small set of possible parametric path findings and runs in sub-linear time with worst-case guarantees. Cao et al. [75] study a data-driven approach that can handle various probability distributions. They formulate the SSP problem as a cardinality minimization problem, which is then relaxed by ℓ_1 norm minimization. The relaxed problem is transformed into an MILP to be solved with standard solvers.

In the case of transporting a single passenger, a vehicle needs to visit two locations to finish the job. As the optimal substructure does not hold here, the SSP to the pickup location may not yield the highest probability of visiting the drop-off location before the deadline. To tackle this problem, Lim et al. [76] prove that the optimal path, maximizing the probability of arrival at the destination through a fixed node sequence, is still an extreme point of the shadow dominant. They present a multi-hop SSP finding algorithm based on parametric optimal path finding. Compared to the simple SSP finding algorithm in [71], the complexity is at most N-1 times higher when there are N visiting nodes in the path.

2.2.2 Vehicle Routing with Stochastic Travel Times

Research that takes travel time uncertainty into account mainly focuses on the VRP with stochastic travel times, where a few vehicles leave from the depot to visit dozens (even hundreds) of locations and return to the depot at the end. The durations of routes are typically very long (up to hours), the capacity of vehicles is quite large (up to dozens) and all of the information is known in advance. It is impossible to find the optimal solution, so methods like Tabu search, simulated annealing and genetic algorithms are generally used to find solutions in a reasonable amount of time.

Kenyon et al. [77] study routing for uncapacitated vehicles without time window constraints. The travel time distributions are assumed to be known and the objective is to maximize the probability that all vehicles finish their job and return to the depot before the deadline. They embed a branch-and-cut algorithm in a Monte Carlo solution procedure to solve the routing problem. Li et al. [78] consider a time window constraint for each visiting location and assume that travel times are normally distributed. A Tabu search based heuristic algorithm is proposed to solve the problem, in which a stochastic simulation is performed to compute the expected travel costs of trips. Taş et al. [79] study a similar problem and aim to minimize both the operational cost and passenger inconvenience. They propose a three phase approach that first adapts the Solomon's insertion heuristic I1 [53] to construct an initial feasible solution, then uses a Tabu search metaheuristic to improve the solution and finally applies a post-optimization to further improve and generate the solution. They later propose a branch-and-price solution approach, embedded with a column generation procedure, to generate exact solutions for the problem [80]. Li et al. [81] consider a problem where passengers and parcels are simultaneously served by the same vehicle fleet. They develop an Adaptive Large Neighborhood Search (ALNS) heuristic to maximize the expected profit. The ALNS iteratively improves a randomly generated initial solution through request selection and perturbation, until the solution converges. In general, the solution approaches to VRP usually employ evolutionary methods. They can find a feasible solution at any given time, but are limited to the dispatch of dozens of vehicles, and cannot be applied to real-time systems due to long computation times (up to hours). They show examples of how stochastic travel times are taken into account when dispatching vehicles, e.g., maximizing the probability of arriving before the deadline and minimizing the travel costs.

2.2.3 Ride-Sharing with Stochastic Travel Times

There has been very little literature regarding travel time uncertainty in the ride-sharing field. Yan et al. [82] are the first to study carpooling with stochastic travel times and formulate it as an integer multiple commodity network flow problem. They present a heuristic algorithm that employs a constraint relaxation technique to solve the problem. The algorithm first generates an initial solution by solving a linear relaxation problem, and then adopts a local improvement method to improve the solution. Long et al. [83] propose a static stochastic ride-sharing model, where a generalized trip cost is introduced and analyzed for both driving-alone and ride-sharing trips. They develop a Monte Carlo simulation method to estimate the cost and a bi-objective ride-sharing matching model to maximize both the number of matches and the trip cost savings yield by ride-sharing. It is assumed that all participants announce their travel plans one day before departure. Li et al. [69] study a static mathematical model to provide computationally tractable methods for real life-size problems. They first present a tabu search based heuristic algorithm which adopts an extended insertion algorithm to find initial solutions. For large scale problems with thousands of participants, they then develop a cluster-first-route-second method that uses two clustering technologies, greedy heuristic and k-means clustering, to decompose the problem into multiple small problems for quick matching and routing. The hybrid heuristic approach could serve as a prerequisite to dynamic ride-sharing and could be solved repeatedly to handle dynamic problems. However, evolution based approaches (e.g., tabu search and local improvement) normally do not support solving large-scale problems in real-time, where the number of vehicles for dispatch is thousands.

Li et al. [84] propose a data-driven robust optimization approach to determine the optimal vehicle-request matching policy and find an optimal route for each vehicle. The objective aims to minimize the total travel costs of vehicles and the number of unmatched requests. They adopt a Recurrent Neural Network (RNN) to predict the nominal and deviation of travel times, use a one-stage optimization model to formulate the decision-making problem, and introduce a robust counterpart reformulation to solve that problem. However, the VRP based MIP formulation yields a long computation time and is not capable of handling large-scale instances.

The scenarios studied in the above approaches are one-to-many P2P ride-sharing and normally compute the cost caused by travel time uncertainty from the perspective of drivers. In SAMoD systems, the objectives are more relevant to the effect of travel time uncertainty on passengers. Liu et al. [85] introduce a reliable path concept and work on improving the reliability of on-time arrival for passengers. For computational efficiency, they assign the vehicle with the maximal on-time arrival probability to each request in an FCFS fashion. The assignment consists of three steps: searching for available vehicles, estimating their reliabilities by leveraging a k-shortest path algorithm [86, 87], and selecting the vehicle with the highest reliability. Besides the most reliable path model, they also investigate a α-reliable path model that minimizes the travel cost while keeping a minimum threshold (e.g., 0.9) of the on-time arrival probability for passengers. However, both the FCFS setting and the k-shortest paths based search ignore global optimality. Moreover, the number of vehicles considered in their work is less than 200, suggesting that the approach may not be applicable to real-time large-scale instances.

2.3 Predictive Vehicle Dispatch

Due to the spatial imbalances of the requests from passengers, simply using the revealed request information to optimize the immediate objective function could lead to a spatio-temporal mismatch between vehicles and requests in the future. This mismatch means that some passengers will not be served because there are very few vehicles nearby, while some vehicles are struggling to find passengers in other locations. Therefore, it is desirable to dispatch vehicles in a more far-sighted way to further improve the operational efficiency.

There are two different approaches to working on predictive optimization: demand forecast based dispatch and value based dispatch. The former leverages a forecast of requests in the near future to maximize the number of requests that can be served [30], and the latter considers the long-term impact of the matching policy to balance the distribution of vehicles across the city [23]. Depending on how the forecast information or long-term value is incorporated, an approach can be further classified as anticipatory vehicle-request matching or idle vehicle repositioning. Anticipatory matching would induce the vehicles to give up the best immediate assignments or detour from the minimum cost schedules to increase the chance of serving more requests in the future. For example, a vehicle is assigned to a low immediate reward trip as it ends at a "hot" area [88], or dispatched to visit an "unnecessary" location before picking up its passengers to serve an extra passenger [30]. Idle vehicle repositioning is also called rebalancing in some work. As vehicles tend to build up in low-demand zones, repositioning focuses on guiding idle vehicles to high-demand zones to reduce the idle time of vehicles and the waiting time of passengers [31, 89].

2.3.1 Demand Forecast Based Dispatch

Forecast based approaches usually partition the road network into zones and estimate the future demand distribution over the zones at each time step. The predicted future travel demand is then used to generate a set of artificial requests to compute far-sighted matching and scheduling policies, or used to identify high-demand zones to reposition idle vehicles to balance the match between vehicles and requests.

Anticipatory Vehicle-Request Matching

Alonso-Mora et al. [90] construct a probability distribution based on historical demand to sample future requests. Building on their myopic batch assignment approach [19], they take into account the sampled future requests to enhance the search for candidate cliques of requests and the scheduling of vehicles. Then an ILP is solved to maximize the number of requests that can be served, consisting of both the real requests and the predicted ones. However, with a significant increase in computation time, only a small improvement in service rate is obtained. Huang et al. [91] argue that the demand distribution used in [90] is inefficient. Using the Lebesgue measure based on a Euclidean space approximation of the road network, they propose a network partition algorithm to characterize the demand distribution. They then regularize the distribution and put it into the objective function to balance the assigning of current requests and future ones. The instance addressed in the paper considers 900 four-seat vehicles. Fielbaum et al. [30] propose an approach that does not rely on any exogenous information, but only utilizes the current and recent requests received during operations. They modify the cost function of each candidate assignment by introducing a reward when the vehicle reaches zones with high request generation rates or high rejection rates. They then adopt the method in [90] and combine it with the modified cost function to diminish the mismatch between vehicles and requests. However, the instance addressed in the paper only includes 1000 three-seat vehicles, which is much smaller than that in [90]. Tsao et al. [20] develop a similar method to [90] that leverages forecasts of future demand to improve service quality. They solve the vehicle-request matching problem in a coupled way based on a network flow model. However, as discussed earlier, the computational efficiency of this approach is not high enough to handle large-scale instances.

Lowalekar et al. [92] argue that, even without adding future requests, the search for all possible candidate assignments and the scheduling for vehicles are computationally intractable. They present a two-stage stochastic approximation to handle the extra computational burden brought by introducing future information into the system. In the first stage, candidate generation and assignment are executed solely based on the current information, using their previous matching method in [28]. The second stage employs future information with approximations to efficiently evaluate the assignment results made in the first stage, by simulating the movement of vehicles. Finally, the assignments for vehicles to serve requests are conducted by accounting for future returns. Their approach is similar to a value based approach, but estimates values using forecasts in the near future. However, compared to their previous myopic approach [28], the computation time is 3 times longer.

Idle Vehicle Repositioning

Also building on [19], Wallar et al. [31] present a different approach from Alonso-Mora et al. [90] to utilize demand estimation to improve efficiency. They focus on the proactive assignment of idle vehicles. They use a particle filter to compute the rates of request demand at different zones, solely based on the real-time request stream, and try to relocate idle vehicles to maximize the expected number of requests that would be observed. Liu et al. [93] build a probability distribution of the appearance of requests and introduce a probabilistic rebalancing method, which formulates the assignment of idle vehicles as a one-to-one matching problem to guarantee computational efficiency. Riley et al. [94], building on their previous column generation based method [29], adopt the Model Predictive Control (MPC) algorithm of [95] to jointly consider the movements of both idle and non-idle vehicles to compute the rebalancing policy. In general, the number of idle vehicles is much lower than the total number of vehicles and some inefficient matches in assignment process cannot be corrected through repositioning [30]. Thus, idle vehicle repositioning is not considered in this thesis. But the literature above may serve as a reference for future research.

2.3.2 Value Based Dispatch

Similar to forecast based approaches, value based approaches also use zones to do dispatch. Instead of predicting the demand distribution, they learn a spatio-temporal value function to evaluate the future return of an assignment, e.g., the potential number of requests that can be served or the total income in several hours or a day. The learned value function can later assist in finding the best matching or repositioning policies.

Anticipatory Vehicle-Request Matching

Most of the literature on value based dispatch has focused on unit capacity vehicle fleets. Xu et al. [64] are the first to employ reinforcement learning in large-scale vehicle dispatch systems to optimize long-term efficiency of vehicle dispatch systems. They model the movement of each vehicle as a Markov Decision Process (MDP) and adopt a policy iteration method to learn a tabular state value function using a Temporal Difference (TD) update. The vehicle-request matching problem is transformed into a bipartite graph matching problem which can be efficiently solved by the Hungarian algorithm [36]. The weight of each edge is computed with a tabular value and an instant reward. However, this tabular approach is susceptible to data sparsity and does not support knowledge transfer. Wang et al. [96] adopt the deep reinforcement learning framework [97] and employ a Deep Q-Network (DQN) to estimate the state-action value function of the vehicle. The network is trained from a single vehicle perspective to maximize the total reward throughout the day. A transfer learning method is also introduced to increase the learning efficiency for multiple city deployments. Building upon [64], Tang et al. [88] model the activities of each vehicle as a semi-MDP that uses temporally extended actions. They propose a cerebellar value network, based on Cerebellar Model Arithmetic Computer (CMCA) [98], that considers coordinations among multiple vehicles to provide a more accurate spatio-temporal value estimation. The historical training data (i.e., vehicle trajectories) is combined with contextual information (i.e., supply and demand conditions) to learn a better state-action representation than the one in [96]. To capture the variations of real-time dynamics, Tang et al. [99] later propose an ensemble method that augments the offline training scheme [88] with a fast online learning. They maintain a centralized value function that is continuously updated with new online experiences and periodically 'reinitialize' it with an offline trained value function that records general time varying patterns.

Due to the huge action space and complex movements of vehicles in ride-sharing, only a few studies on value based dispatch consider multiple capacity vehicle fleets. Jindal et al. [100] consider a scenario where at most two requests can share a vehicle. They define three optional actions for each vehicle: staying at the current location, assigning a single passenger trip and assigning a ride-sharing trip. A double-DQN [101] is used to learn the state-action value function for vehicles. But the action space is not granular enough for complex vehicle-trip matching. Yu et al. [102] model dynamic ride-sharing as a multistage stochastic program and apply Approximate Dynamic Programming (ADP) to solve it. They focus on studying the structure of value functions and prove that the optimal value function is monotone, which indicates that a vehicle will have a higher expected value in step $t+1$ if starting with a higher expected value in step t. They also assume that a vehicle can be shared by no more than two requests at a time, meaning that the approach may not be applicable to high-capacity vehicles. Shah et al. [32] adopt DQN to learn value functions based on an ADP formulation of the vehicle-request matching problem. They use post-decision state to decompose the value function to fit it into an ILP that computes the assignments. The value function is then learned through a general Bellman update. The assignment framework, similar to the one in [64], is built on the candidate search algorithm in [19] to dispatch high-capacity vehicles. However, the value function is trained in a simulation environment, which is a time-consuming process (it takes around one week [92]) and faces a "reality gap" when deployed in the real world. Moreover, computing matching policies simply on a value function trained on historical data does not handle dynamic environments very well [92, 99].

Idle Vehicle Repositioning

When working on the rebalancing of idle vehicles, the capacity of vehicles has very little impact on the problem formulation and complexity, as each vehicle can only visit one repositioning location at a time. Wen et al. [103] divide the neighbouring area of each vehicle into grids and consider nine actions that either stay or move to an adjacent grid. A DQN agent is trained to select actions for vehicles to maximize the savings in passenger waiting time, compared to the case without rebalancing. Jiao et al. [89] consider a hexagonal grid system and use a similar training framework to [88] for learning a state-action value function. They differ from [88] by maintaining and updating two value networks to account for both dispatched and idle states of vehicles. Tang et al. [99] later propose a unified value function that tackles both vehicle-request matching and repositioning tasks. Gammelli et al. [104] argue that repositioning can be considered as a node-wise decision-making problem and leverage graph neural networks to exploit the connectivity of the road network. They employ the Advantage Actor-Critic (A2C) algorithm [105] to learn the state value function aggregating information across nodes in the road network.

2.4 Chapter Summary

This chapter provides an overview of various research topics that relate to or motivate the research of this thesis. Considering efficient vehicle-request matching, approaches can be categorized to three types. The first type concerns efficiency most and uses greedy algorithms that assign one request at a time to the best-matched vehicle to reduce the computational complexity (e.g., [24, 26, 65]). These FCFS based solutions are good for initial deployment because of their simplicity, but not for long-term operation as the notions of global optimality are ignored. The second type investigates batch assignment for system-wide optimization. Batch assignment has been extensively employed in real-world large-scale ride-hailing platforms (e.g., Uber and DiDi [64]) and shown to yield optimal matching in each batch window [23], but ride-sharing is normally not allowed on these commercial platforms. To overcome computational burdens, literature considering ride-sharing employs heuristic algorithms to accomplish the computation of matching policies in meeting real-time requirements (e.g., [19, 27, 68]). Although some approaches can theoretically obtain optimal matching policies if given sufficient time, similar to evolutionary algorithms, the optimum cannot be achieved in a reasonable amount of time. The third type formulates the vehicle dispatch problem as a network flow model and guarantees optimality of the solution, but are limited to solve small-scale instances in real-time (e.g., [20]). By contrast, we employ the batch assignment and focus on improving the efficiency of generating candidate matches while guaranteeing a complete search, to allow the *optimal* dispatch of *large-scale* fleet (i.e., thousands of vehicles) with *high-capacity* (i.e., up to 10 seats) in *real-time* to fully utilize the benefits of ride-sharing (Chapter 3).

In the context of ride-sharing under travel time uncertainty, only limited research has been conducted. There have been some approaches focusing on the vehicle routing problem with stochastic travel times (e.g., [78, 79, 81]), but they are limited to dispatching tens of vehicles, and cannot be applied to real-time systems because the computation time is up to 5 hours. Several studies consider ride-sharing with stochastic travel times (e.g., [69, 83–85]). However, they either assume all passengers announce their trip schedules one day before, only support ride-sharing with no more than two passengers, or are limited to small-scale problems. Our work differs from them by considering *large-scale high-capacity on-demand* ride-sharing. In addition to optimizing the on-time arrival probibilities of passengers, we also consider the optimization of the profit of the service provider (Chapter 4)

The literature on predictive vehicle dispatch is categorized into two groups: forecast based and value based dispatch. Forecast based dispatch uses a predicted distribution of requests in the near future to compute sophisticated matching policies that serve more requests than myopic approaches (e.g., [30, 90, 92]). However, inserting future requests into the computing process results in a huge computational burden and makes it difficult to solve problems in real-time. Moreover, the lookahead duration is normally limited to several minutes, because considering more future requests yields higher computational complexity that requires tighter heuristics to solve the problem, which would in turn reverse the improvement achieved by lookahead [30, 92]. Value based dispatch runs in a far-sighted way by leveraging a value function to evaluate the potential return of matching policies throughout the day and does not affect the computational complexity of matching. But most of them do not allow ride-sharing (e.g., [64, 88, 99]). The only one that considers high-capacity ride-sharing (i.e., [32]) relies on training with a simulator and suffers from a long training time. Our work focuses on using historical taxi trajectories to efficiently learn a value function and combining that value function with a *re-optimization* procedure to optimize long-term objectives of *high-capacity* ride-sharing better (Chapter 5).

Chapter Three

Optimal Online Dispatch in High-Capacity Ride-Sharing

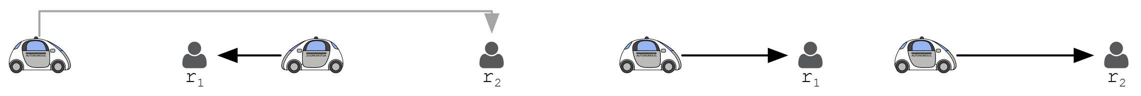
Shared Autonomous Mobility-on-Demand (SAMoD) systems hold great promise for improving the efficiency of urban transportation, but are challenging to implement due to the huge scheduling search space and highly dynamic nature of requests. In this chapter, we develop a novel Optimal Schedule Pool (OSP) assignment approach to optimally dispatch high-capacity ride-sharing vehicles in real-time, including: (1) an incremental search algorithm that can efficiently compute the exact lowest-cost schedule of a ride-sharing trip with a reduced search space; (2) an iterative online re-optimization strategy to dynamically alter the assignment policy for new incoming requests, in order to maximize the service rate. Experimental results based on New York City taxi data show that our proposed approach outperforms the state-of-the-art in terms of service rate and system scalability.

3.1 Introduction

The basic idea behind SAMoD systems is to assign suitable vehicles to each request and group multiple requests into ride-sharing trips if they are travelling in similar directions. There are several technical challenges that need to be addressed for the significant adoption of SAMoD systems:

1. *Large-scale.* A typical urban taxi system has over ten thousand vehicles. The state space of ride-sharing combinations and routes grows exponentially as the number of requests or the capacity of vehicles increases. Dispatching large fleets is a major computational challenge.
2. *Time-sensitive.* Passengers are sensitive to the time of service. Each request needs to be assigned within a few seconds and completed as soon as possible. Matching decisions are made on-the-fly.
3. *Dynamic.* Requests are received continuously throughout the day, instead of being known in advance. An optimal assignment at a given time may not be the best when considering additional new requests.

Many approaches to controlling and analyzing SAMoD systems have been studied, such as multi-commodity flow models [20], queuing network models [106] and search-based models [19, 26]. Due to computational complexity, most existing work is restricted to small-scale and double-occupancy fleets for optimal assignments. Algorithms for optimal assignment usually formulate the problem as a MILP, which is impractical when thousands of vehicles are needed. For practical applications, greedy methods have been used to accelerate the computation for large fleet ride-sharing. The most straightforward and most popular solution is assigning requests to the vehicle fleet in a FCFS queue, where passengers are sequentially assigned to their best-matched vehicles. The FCFS based methods are very efficient but may not be effective. As the optimality of assignment is not guaranteed, the benefits of ride-sharing cannot be fully achieved. Figure 3.1(a) shows a case where a bad assignment would be produced if request 1 is sent to the dispatch server ahead of request 2.



(a) Best assignment only for request 1 (b) Best assignment for both request 1 & request 2

Figure 3.1: An example of assigning two requests to two vehicles (a) in a FCFS queue or (b) in a batch planning manner.

Batch planning is a promising approach to producing high quality and even optimal online assignment policies. Compared to the FCFS based greedy methods, system-wide coordination between vehicles is considered to improve the quality of assignments, as shown in Figure 3.1(b). The key component of batch assignment is solving a one-to-many matching problem between vehicles and requests, and the quality of matching greatly depends on two problems: candidate filtering and vehicle scheduling [107, 108]. The former, searching for feasible ride-sharing trips that each vehicle is able to serve, has attracted a lot of study (e.g., [19, 28]). While the latter, computing the best orders of picking and dropping multiple requests and verifying the feasibility of a ride-sharing trip for each vehicle, has received very little attention. Existing work either uses greedy insertion or exhaustive search to compute schedules, which in practice cannot produce good solutions. The state-of-the-art batch assignment method proposed by Alonso-Mora et al. [19] theoretically guarantees optimality if all steps are executed until termination, but this method cannot converge in real-time settings as the vehicle scheduling problem is not properly tackled.

This chapter proposes an efficient and optimal online batch assignment scheme that optimizes the service rate for dispatching high-capacity SAMoD systems in a practical timeframe. We use an incremental computation heuristic to reduce the search space of scheduling, and an iterative re-optimization procedure to dynamically and efficiently alter the assignment policy for better performance. The proposed approach is executable in real-time settings and can guarantee the optimality of the assignment at each dispatch epoch, for all received requests. Optimality means that the service rate cannot be further improved based on the information already revealed. To summarize the work in this chapter, we:

1. Develop an incremental search algorithm to efficiently compute the optimal schedule of a ride-sharing trip, which reduces global search to local search through heuristics while ensuring optimality. (Section 3.3.1.)
2. Combine the optimal schedule search algorithm with the feasible trip search algorithm of [19] to generate all possible ride-sharing trips for each vehicle, along with the optimal schedule for each trip. (Section 3.3.2.)
3. Develop an iterative re-optimization strategy to avoid myopic optimality, which takes into account both previous and new requests to optimize long-term system effectiveness. (Section 3.3.3, 3.3.4.)
4. Perform simulations with large-scale taxi data to evaluate our proposed approach and compare it to three representative algorithms. (Section 3.4.)

The remainder of the chapter is arranged as follows. In Section 3.2, we introduce the definition of an SAMoD system, formulate the optimal dispatch problem and give an overview of the online dispatch framework. In Section 3.3, we present the technical details of the proposed dispatch method. In Section 3.4, we evaluate our approach using the taxi data in New York City. In Section 3.5, we draw our conclusions.

3.2 Preliminaries

As passenger requests appear throughout the day, the vehicle dispatch method presented in this chapter adopts the industrial practice [23, 35] where submitted requests are periodically packed and allocated together to suitable vehicles.

3.2.1 Definitions

A central dispatcher computes vehicle-request matches in a rolling horizon framework, the time window length of which is ΔT.

The dispatcher controls a fleet of m vehicles $V = \{v_1, \dots, v_m\}$, of which the capacity of each one is κ. The state of each vehicle is defined as a tuple $\langle q_v, s_v \rangle$, where q_v is its current position and s_v is a planned schedule consisting of a sequence of pick-up and drop-off tasks to serve assigned requests.

At each epoch, the dispatcher considers a set of n requests $R = \{r_1, \dots, r_n\}$ submitted by passengers. Each request is defined as a tuple $\langle o_r, d_r, t_r \rangle$, where o_r is the origin (i.e., pick-up location), d_r is the destination (i.e, drop-off location) and t_r is the time when the request is submitted. To measure the quality of passenger experience, a waiting time ω_r (i.e., the difference between when it is actually picked up and when it is submitted) and a total travel delay δ_r (i.e., the difference between when it is actually dropped off and when it is expected to arrive if travelling alone) are associated with each request.

A group of requests that can be assigned to a single vehicle via ride-sharing is presented as a trip $\Gamma = \{r_1, \dots, r_{n_\Gamma}\}$. A trip might have more than one candidate vehicle to serve it and vice versa. For example, trip $\{r_1, r_2\}$ may be served either by vehicle v_1 or by v_2; vehicle v_1 may either serve $\{r_1, r_2\}$ or $\{r_2, r_3\}$.

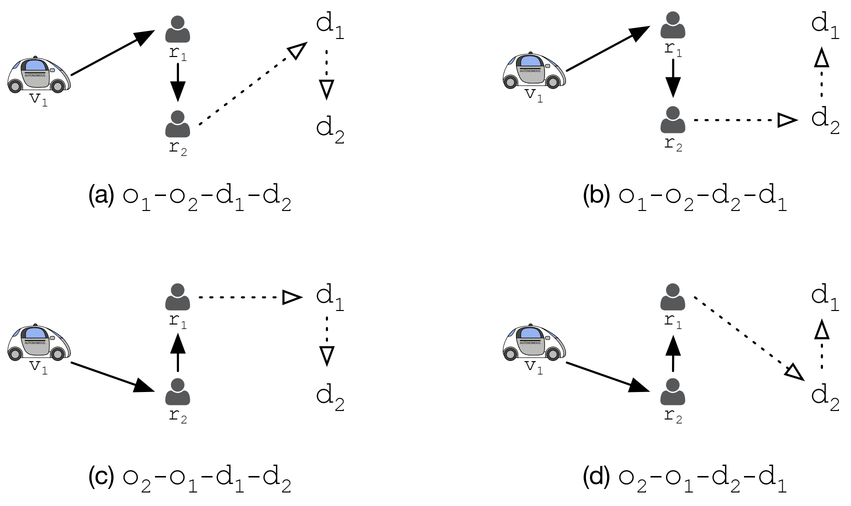


Figure 3.2: All possible schedules for a ride-sharing trip containing two requests.

For a specific vehicle-trip match, the order for the vehicle v to pick up and drop off requests in the trip Γ is defined as a schedule $s_{v,\Gamma} = \{o_1, \dots, o_2, \dots, d_1, \dots, d_{n_\Gamma}\}$. There might be more than one feasible schedule; the set of all feasible schedules is denoted by $S_{v,\Gamma}$ and the optimal one is denoted by $s^*_{v,\Gamma}$, i.e., the one serving the requests with the minimum delay. For example, there are four possible schedules for a two request sharing trip, as shown in Figure 3.2; we may find that $S_{v_1,\{r_1,r_2\}} = \{schedule\ (c), schedule\ (d)\}$ and $s^*_{v_1,\{r_1,r_2\}} = \{o_2, o_1, d_1, d_2\}$. A feasible schedule must satisfy the following constraints:

- *Capacity constraint.* For each vehicle, the number of onboard passengers cannot be larger than its capacity.
- *Delay constraint.* For each request, its waiting time ω_r and total travel delay δ_r must be lower than two thresholds, Ω and Λ, respectively.
- *Precedence constraint.* For each request in a schedule $s_{v,\Gamma}$, its origin must be visited before its destination.

3.2.2 Problem Statement

Based on the batched requests and the current statuses of vehicles, the dispatcher works on serving as many requests as possible with a minimum travel detour resulting from ride-sharing. Using R_{miss} to denote the set of requests that cannot be matched during the current dispatch epoch, we define the cost of a dispatch policy as:

$$C_{Delay} = \sum_{v \in V} delay(s_v) + \sum_{r \in R_{miss}} p_{miss} \tag{3.1}$$

where $delay(s_v)$ is the sum of travel delays of the requests included in a schedule and p_{miss} is a very large penalty for rejecting a request.

Problem 1 (Optimal online dispatch) *Given a set of requests R and a set of vehicles V at an epoch with a length of ΔT, the problem of optimal online dispatch is to find maximum ride-sharing allocations for requests and compute feasible schedules for vehicles in real-time, so that the cost function (i.e., Equation (3.1)) is minimized, subject to the constraint that each request must be served by exactly one vehicle.*

3.2.3 System Framework

The dispatcher runs an assignment planner, as shown in Figure 3.3, periodically every ΔT (e.g., 30 sec), where all unpicked-up requests $R = R_{new} \cup R_{prev}$ and all vehicles V are pooled and matched simultaneously. Not only requests received at the current epoch, denoted by R_{new}, but also requests received earlier but not yet picked up by vehicles, denoted by R_{prev}, are considered to compute the best assignment policy.

At each dispatch epoch, the dispatcher considers the locations and delay constraints of requests $R = R_{new} \cup R_{prev}$, which are depicted as "New Request Pool" and "Not Yet Picked Up Request Pool", as well as the latest "Statuses" (locations and routes) of all vehicles

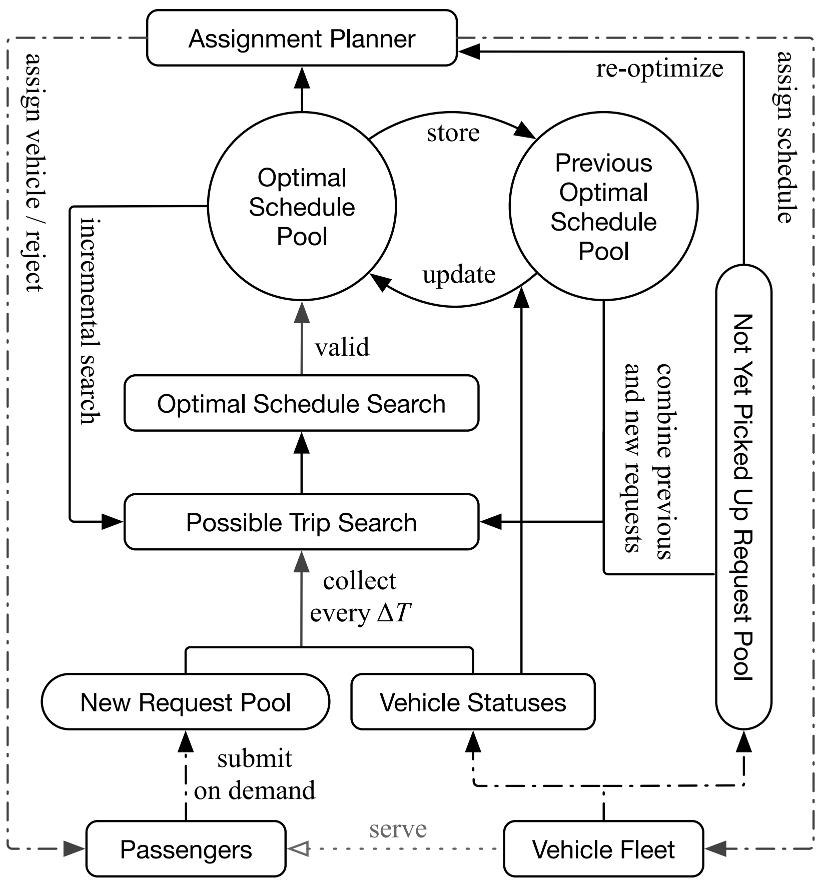


Figure 3.3: Framework of the optimal dispatch logic.

V, to search for all possible candidate vehicle-trip matches, which is depicted as "Possible Trip Search". The search is processed independently for each vehicle and incrementally by increasing the size of shared trips. For each possible candidate match considered by "Possible Trip Search", the dispatcher solves a vehicle scheduling problem to verify whether or not it is feasible, which is depicted as "Optimal Schedule Search". If any feasible schedules are found, this candidate match is marked valid and its optimal schedule $s^*_{v,\Gamma}$ is added to the "Optimal Schedule Pool". After the search for candidate matches is finished, the dispatcher will have a complete candidate space for each vehicle, denoted by $\mathcal{F}^*_v = \{s^*_{v,\Gamma_1}, s^*_{v,\Gamma_2}, \dots\}$, where each $s^*_{v,\Gamma}$ is called a candidate schedule that represents a candidate vehicle-trip match and contains the optimal routing policy for the match. In addition, a "Previous Optimal Schedule Pool" is maintained and updated to save repetitive computations.

After the optimal schedule pool has been generated, the dispatch assigns each vehicle a suitable schedule, ensuring that the assignment policy is conflict-free, to maximize the system performance, i.e., minimizing Equation (3.1). Considering all received requests together makes it possible to reject some requests assigned from the previous epoch to achieve a higher service rate, but this is not allowed in this thesis to ensure the satisfaction of passengers.

3.3 Optimal Online Dispatch Scheme

We introduce an incremental search algorithm to reduce the search space of the optimal vehicle scheduling problem, which is further coupled with the possible trip search algorithm of [19] to efficiently generate the optimal schedule pool. We then present a re-optimization strategy with speed-up heuristics to improve long-term system performance. Based on the optimal schedule pool, a constrained optimization problem is formulated to produce the optimal allocation policy.

3.3.1 Optimal Schedule for a Single Ride-Sharing Trip

Computing the optimal schedule for a vehicle serving all requests in a trip is computationally expensive in general, as it is a generalization of the Travelling Salesman Problem with Precedence Constraints [107]. The search space of possible schedules increases exponentially with the trip size. For example, a ride-sharing trip of two requests only has four possible schedules, shown in Figure 3.2, but a trip of three requests has 52 possible schedules, while a trip of four requests has more than 576 possible schedules. The minimum-cost schedule for a low-capacity ride-sharing trip can be computed via exhaustive search, but it is computationally intractable for large trips.

In SAMoD systems, the tight constraints on maximum travel delay of passengers naturally narrows the solution space [109], which means that the number of feasible schedules for each possible vehicle-trip match is normally small. If we can identify that a certain set of schedules is infeasible, the optimal schedule for a ride-sharing trip can be quickly found without an exhaustive and time-consuming search. The following observation leads us to the idea of incrementally searching for all feasible schedules, where the optimal one lies, for a large ride-sharing trip.

Lemma 1 *A schedule $s_{v,\Gamma}$ can be feasible only if any sub-schedule $s_{v,\Gamma} \backslash \{o_r, d_r\}$ (obtained by removing one request) of it is feasible, where $s_{v,\Gamma} \backslash \{o_r, d_r\} \in S_{v,\Gamma\backslash r}$. Therefore, a schedule $s_{v,\Gamma}$ only needs to be checked for feasibility if, for any r in the schedule, the schedule set $S_{v,\Gamma\backslash r}$ is not empty.*

Using Lemma 1, it is found that all $s_{v,\Gamma} \in S_{v,\Gamma}$ can be obtained by inserting a request r into some $s_{v,\Gamma\backslash r} \in S_{v,\Gamma\backslash r}$. We propose Algorithm 1 to efficiently compute the exact minimum-cost schedule of a high capacity ride-sharing trip Γ by only searching potentially feasible schedules instead of all schedule permutations. It generates the feasible schedule set $S_{v,\Gamma}$ incrementally by extending the initial schedule set and returns the optimal schedule $s^*_{v,\Gamma}$ or an empty schedule if there is no feasible schedule. S^a_v and S^b_v are defined as two sets of all feasible schedules for vehicle v to serve size $k-1$ and size k trips. In lines 6-7, it computes the set of all feasible schedules S^b_v by extending schedules in S^a_v. The function *InitScheduleSet*(v) returns all the feasible schedules for the vehicle dropping passengers on board. For example, if a vehicle has two passengers onboard and it is feasible to drop either one first, the initial schedule set will be $\{\{d_1, d_2\}, \{d_2, d_1\}\}$. This is to ensure that all possible schedules are considered in the subsequent schedule searches. Since the onboard passengers have already been picked up, we cannot re-assign them to other vehicles, and the init schedule set will not be empty. The function *BasicScheduleInsersion*$(schedule, r)$, detailed in Algorithm 2, tries

Algorithm 1 Optimal Schedule Computing

Input : A vehicle v and a trip Γ.

Output: The new optimal schedule $s^*_{v,\Gamma}$ for the vehicle v.

```
1:  $S^a_v \leftarrow InitScheduleSet(v)$;
2:  $k \leftarrow 1$;
3:  while $k <= n_\Gamma$ do
4:      $S^b_v \leftarrow \emptyset$;
5:      $r \leftarrow \Gamma.pop()$;
6:      for each $schedule \in \mathcal{S}^a_v$ do
7:          $S^b_v \leftarrow S^b_v \cup BasicScheduleInsersion(schedule, r)$;
8:      $S^a_v \leftarrow S^b_v$;
9:      $k \leftarrow k + 1$;
10:     if $S^a_v = \emptyset$ then
11:         break;
12: $s^*_{v,\Gamma} \leftarrow$ the minimum-cost schedule from $S^b_v$;
```

to insert o_r and d_r into all possible places to obtain new schedules and returns a set of all feasible new schedules or an empty set.

Assuming an output schedule of Algorithm 1 has n_s visiting locations, in the worst case, the number of searched schedules is $O(n_s!)$, which is in line with the scheduling problem being a generalization of the Travelling Salesman Problem. Nonetheless, this can only happen when passengers have unlimited tolerance for wait times and detours, which is not possible in reality. The tight detour constraint means that the size of any schedule set $S_{v,\Gamma}$ is normally small. Algorithm 1 reduces finding the optimal schedule for a vehicle serving a trip from enumeration to a couple of basic schedule insertion processes, by pruning out the infeasible subset of all possible schedules. As a result, the search space of possible schedules is significantly reduced and the feasibility of a vehicle-trip match can be quickly and

Algorithm 2 Basic Schedule Insertion

```
Input  : A vehicle's current schedule $s_v$ with length $n$ and a request $r$.
Output: A set of all new feasible schedules $S_v$ for the vehicle $v$.
1: $S_v \leftarrow \emptyset$;
2: for $i \leftarrow 1$ to $n$ do
3:     for $j \leftarrow i$ to $n$ do
4:         $s'_v \leftarrow insert\ o_r\ and\ d_r\ into\ the\ i-th\ and\ j-th\ locations\ of\ s_v$;
5:         if $s'_v\ is\ feasible$ then
6:             $S_v \leftarrow S_v \cup \{s'_v\}$;
```

accurately verified. Figure 3.4 shows an example where the optimal schedule for vehicle v_1 to serve trip $\{r_1, r_2, r_3\}$ is found by checking half of all possible schedules. In practice, the number of searched schedules is orders of magnitude lower than $O(n_s!)$. The experimental study in Section 3.4 shows that, when applying a maximum wait time constraint of 5 min and a maximum travel delay constraint of 10 min, the average number of searched schedules is less than 2000 for size 10 ride-sharing trips, where $10 < n_s <= 20$. Note that, when travelling alone, the mean trip time is 10.5 min and about 90% of passenger trips have trip times less than 20 min [110].

Algorithm 2 enumerates all possible insertions of o_r and o_d, and checks whether each location is visited without constraint violation. The time complexity it is $O(n_s^3)$. The efficiency of Algorithm 2 can be further improved by employing some heuristic search methods, such as the Dynamic Programming (DP) based algorithm in [26]. As this is not the computational bottleneck of the research problem discussed in this thesis, we refer the reader to Tong et al. [26] for further interest.

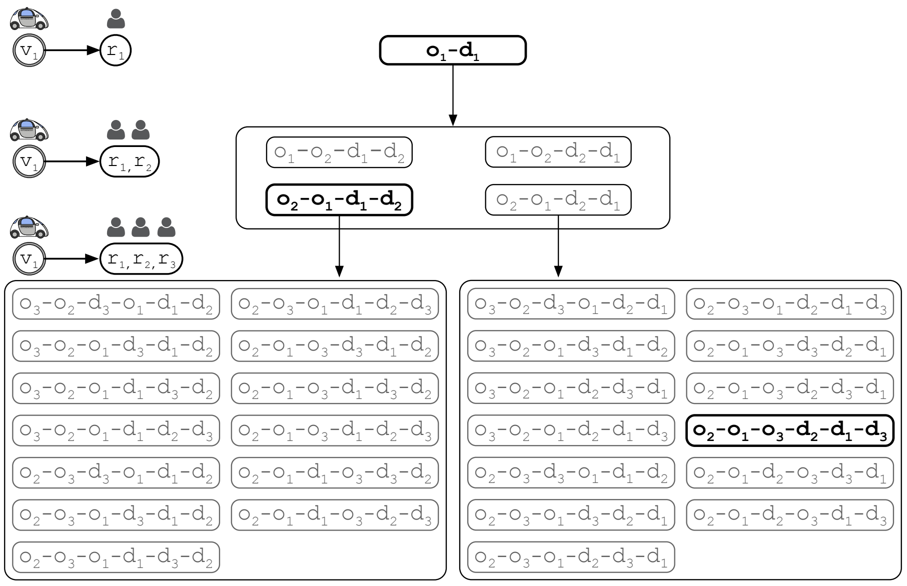

v_1
r_1
v_1
r_1, r_2
v_1
r_1, r_2, r_3
o_1-d_1
$o_1-o_2-d_1-d_2$
$o_1-o_2-d_2-d_1$
$o_2-o_1-d_1-d_2$
$o_2-o_1-d_2-d_1$
$o_3-o_2-d_3-o_1-d_1-d_2$
$o_2-o_3-o_1-d_1-d_2-d_3$
$o_3-o_2-o_1-d_3-d_1-d_2$
$o_2-o_1-o_3-d_3-d_1-d_2$
$o_3-o_2-o_1-d_1-d_3-d_2$
$o_2-o_1-o_3-d_1-d_3-d_2$
$o_3-o_2-o_1-d_1-d_2-d_3$
$o_2-o_1-o_3-d_1-d_2-d_3$
$o_2-o_3-d_3-o_1-d_1-d_2$
$o_2-o_1-d_1-o_3-d_3-d_2$
$o_2-o_3-o_1-d_3-d_1-d_2$
$o_2-o_1-d_1-o_3-d_2-d_3$
$o_2-o_3-o_1-d_1-d_3-d_2$
$o_3-o_2-d_3-o_1-d_2-d_1$
$o_2-o_3-o_1-d_2-d_1-d_3$
$o_3-o_2-o_1-d_3-d_1-d_2$
$o_2-o_1-o_3-d_3-d_2-d_1$
$o_3-o_2-o_1-d_2-d_3-d_1$
$o_2-o_1-o_3-d_2-d_3-d_1$
$o_3-o_2-o_1-d_2-d_1-d_3$
$o_2-o_1-o_3-d_2-d_1-d_3$
$o_2-o_3-d_3-o_1-d_1-d_2$
$o_2-o_1-d_2-o_3-d_3-d_1$
$o_2-o_3-o_1-d_3-d_2-d_1$
$o_2-o_1-d_2-o_3-d_1-d_3$
$o_2-o_3-o_1-d_2-d_3-d_1$

Figure 3.4: An example of optimal schedule computation for a vehicle to serve three requests, where the optimal one is marked in bold. By extending the two feasible schedules for trip $\{r_1, r_2\}$, $\{o_2, o_1, d_1, d_2\}$ and $\{o_2, o_1, d_2, d_1\}$, only 26 possible schedules for trip $\{r_1, r_2, r_3\}$ need to be considered to check its feasibility and find its optimal schedule.

3.3.2 Optimal Schedule Pool for a Vehicle Fleet

Generating the optimal schedule pool requires finding all candidate vehicle-trip matches, which depends on two problems: candidate filtering and vehicle scheduling. To tackle the candidate filtering problem, Alonso-Mora et al. [19] propose a feasible trip search algorithm to incrementally compute candidate vehicle-trip matches. Based on the observation in Lemma 2, an infeasible vehicle-trip match can be quickly identified without solving the scheduling problem. However, their approach may mistakenly consider a vehicle-trip match to be infeasible even if it passes the sub-trip feasibility check, as they only search for a limited number of schedules for the sake of efficiency.

Lemma 2 *A vehicle-trip match between v and Γ can be feasible only if any match between v and $\Gamma \backslash r$ (i.e., a sub-trip of Γ, obtained by removing one request) is feasible.*

Algorithm 3 Optimal Schedule Pool Generating

```
Input  : A vehicle $v$ and a set of requests $R$.
Output: The optimal schedule pool $\mathcal{F}_v^*$ for the vehicle $v$.
 1: $\mathcal{F}_v^k \leftarrow \emptyset, \forall k \in \{0, 1, \ldots, \kappa_v\}$;
 2: $S_{v,\{\emptyset\}} \leftarrow InitScheduleSet(v)$;
 3: $\mathcal{F}_v^0 \leftarrow \mathcal{F}_v^0 \cup \{S_{v,\{\emptyset\}}\}$;
 4: for each $r \in R$ do
 5:     $S_{v,\{r\}} \leftarrow SearchFeasibleScheduleSet(S_{v,\{\emptyset\}}, r)$;
 6:     $\mathcal{F}_v^1 \leftarrow \mathcal{F}_v^1 \cup \{S_{v,\{r\}}\}$;
 7: $k \leftarrow 2$;
 8: while $\mathcal{F}_v^{k-1} \neq \emptyset$ and $k <= \kappa_v$ do
 9:     for all $S_{v,\Gamma_i}, S_{v,\Gamma_j} \in \mathcal{F}_v^{k-1}$ with $|\Gamma_i \cup \Gamma_j| = k$ do
10:         Denote $\Gamma_i \cup \Gamma_j = \Gamma^k = \Gamma_i \cup \{r_{new}\}$;
11:         $S_{v,\Gamma^k} \leftarrow SearchFeasibleScheduleSet(S_{v,\Gamma_i}, r_{new})$;
12:         $\mathcal{F}_v^k \leftarrow \mathcal{F}_v^k \cup \{S_{v,\Gamma^k}\}$;
13:     $k \leftarrow k + 1$;
14: $\mathcal{F}_v \leftarrow \mathcal{F}_v^1 \cup \cdots \cup \mathcal{F}_v^{\kappa_v}$;
15: $\mathcal{F}_v^* \leftarrow ExtractOptimalSchedule(\mathcal{F}_v)$;
```

Borrowing the algorithmic idea of sub-trip feasibility from [19], we develop a procedure that can truly compute all feasible trips for a vehicle, along with the optimal schedule for each trip. It is called joint trip searching and optimal scheduling, and is illustrated in Algorithm 3, where $\mathcal{F}_v = \{S_{v,\Gamma_1}, S_{v,\Gamma_2}, S_{v,\Gamma_3}, \ldots\}$ is defined as the feasible schedule pool, containing all feasible schedule sets for a vehicle v. Algorithm 3 incrementally computes $\mathcal{F}_v$ by increasing trip size and outputs the optimal schedule pool $\mathcal{F}_v^* = \{s^*_{v,\Gamma_1}, s^*_{v,\Gamma_2}, \ldots\}$, which contains all candidate optimal schedules of vehicle v serving all trips $\Gamma_i \subseteq R$. The function *SearchFeasibleScheduleSet*$(S_{v,\Gamma}, r)$ in lines 5 and 12, as in lines 6-7 of Algorithm 1, tries to compute schedules of trip $\Gamma \cup \{r\}$ and returns the set of feasible schedules $S_{v,\Gamma\cup\{r\}}$ if possible. Algorithm 3 can be parallelized across different vehicles.

Algorithm 3 further reduces the computation of $s^*_{v,\Gamma}$ to $|S_{v,\Gamma\backslash r}|$ calls to function $BasicScheduleInsersion(schedule, r)$. Many repetitive computations are saved, as the feasible schedule set $S_{v,\Gamma}$ does not need to be computed completely from scratch on each request in trip Γ. Figure 3.5 shows an example of joint trip searching and optimal scheduling for a vehicle-trip match between v_1 and $\{r_1, r_2, r_3, r_4\}$. The optimal schedule $s^*_{v_1,\{r_1,r_2,r_3,r_4\}}$ can be found by two runs of the basic schedule insertion function. The search process for schedule $\{o_2, o_1, o_3, d_1, d_2, d_3\}$ and $\{o_2, o_1, o_3, d_2, d_1, d_3\}$ does not need to be re-computed once more, because it has been completed when computing the optimal schedule for vehicle v_1 and trip $\{r_1, r_2, r_3\}$.

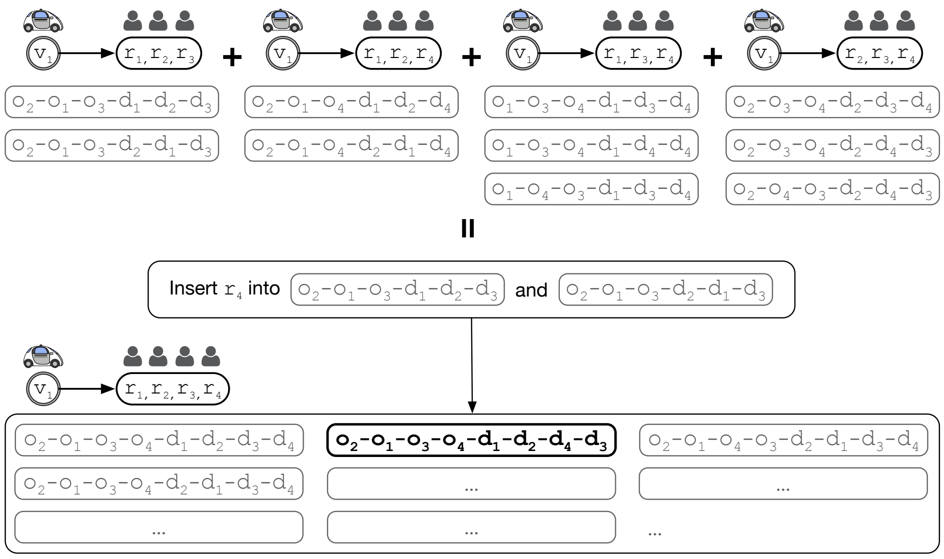


Figure 3.5: An example of joint trip searching and optimal scheduling for a vehicle to serve four requests. Possible schedules for trip $\{r_1, r_2, r_3, r_4\}$ can be directly built on its sub-trip's feasible schedules.

By considering all feasible schedules for each trip, Algorithm 3 generates a complete candidate space for each vehicle. Although [19] can also theoretically generate a complete candidate space if given enough computational time, it cannot do this in a reasonable time.

To make the algorithm solverable in real-time, a set of timeouts are used to trade-off optimality for tractability. Therefore, some feasible vehicle-trip matches will be mistakenly ignored, as discussed in Section 3.4. By tackling the scheduling problem, Algorithm 3 explores all possible vehicle-trip pairs and finds more candidate matches than [19].

3.3.3 Re-Optimization and Iterative Updating

Although batch assignment approaches consider multiple requests simultaneously, there may still be cases where a later request, that can be served if the length of the batch period is larger, will be rejected due to the lack of feasible seat. Figure 3.6 (a) shows an example that the best assignment for r_1 and r_2, when the appearance of r_3 is not known, leaves no available vehicle for r_3. Yet, v_1 and v_2 have the chance to serve all 3 requests, as shown in Figure 3.6 (b). Extending the length of the dispatch window to consider more requests at the same time is an easy solution to think of, but some requests will waste their time waiting for the response from the dispatcher and eventually have a bad service experience. Therefore, re-optimization is introduced to escape from a local minimum by removing the previous assignment and computing a new one for each vehicle based on all known requests. Previously assigned requests are allowed to be re-assigned for a better system-wide match, i.e., serving more requests, when the dispatcher receives new requests. Take the example in Figure 3.6, r_2 is assigned to v_2 at t and re-assigned to v_1 at $t + \Delta T$, so that the dispatcher can re-route v_2 to serve r_3. The dispatcher still computes the assignments for requests in a short time window, and will continually alter the assignments in the following dispatch periods until the requests are picked up for better performance.

However, it is very computationally expensive if we directly compute the optimal schedule pool for all unpicked-up requests. So we design an *updating heuristic* to reduce computation by generating the optimal schedule pool from previous schedules $\mathcal{F}^*_{v,prev}$.

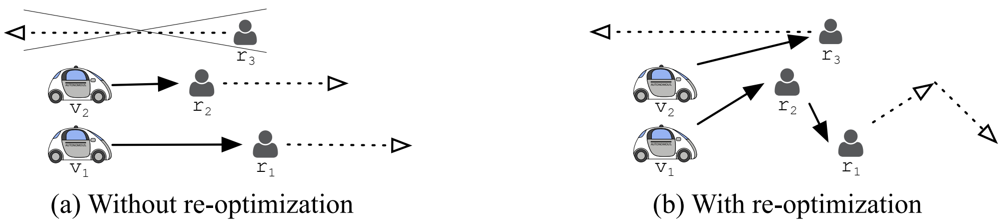


Figure 3.6: An example of online assignment, where r_1 and r_2 are revealed at t and r_3 is revealed at $t + \Delta T$. Considering re-optimization, all three requests can be served.

The dispatcher does not need to re-compute the possible ride-sharing combinations among R_{prev}. The procedure is shown in Algorithm 4. Function *UpdatePreviousSchedule*$(\mathcal{F}^*_{v,prev}, v)$ updates schedules in $\mathcal{F}^*_{v,prev}$ based on the current status of the vehicle, removes the infeasible ones and outputs the feasible ones $\mathcal{F}_{v,prev} = \mathcal{F}^1_{v,prev} \cup \cdots \cup \mathcal{F}^{\kappa_v}_{v,prev}$. Function *Size1ScheduleSet*$(v, \mathcal{R}_{new})$ is equivalent to lines 4-6 in Algorithm 3 and computes the optimal schedule set $\mathcal{F}^1_v$ for R_{new}.

Only computing combinations between the new requests R_{new} and the previous ones R_{prev}, rather than between all known unpicked-up requests $R = R_{new} \cup R_{prev}$, makes the algorithm more efficient. Suppose we have total n_{all} unpicked-up requests and n_{new} of them are newly submitted; the number of combinations for naive computation is $1/2 \cdot n_{all} \cdot (n_{all} - 1)$, while the number with the updating heuristic is $1/2 \cdot (2n_{all} - n_{new} - 1) \cdot n_{new}$. We define $z = n_{new}/n_{all}$, as normally $n_{new} \gg 1$, then:

$$\begin{aligned}\frac{1/2 \cdot (2n_{all} - n_{new} - 1) \cdot n_{new}}{1/2 \cdot n_{all} \cdot (n_{all} - 1)} &= \frac{2 \cdot n_{new}}{n_{all}} - \frac{n_{new} \cdot (n_{new} - 1)}{n_{all} \cdot (n_{all} - 1)} \\ &= 2z - z \cdot \frac{n_{new} - 1}{n_{all} - 1} \\ &\approx 2z - z^2\end{aligned}$$

In our experiments (see Section 3.4), the number of new requests is normally less than 500 while the number of all unpicked-up requests is up to 2600. There are significantly fewer combinations between 500 and 2600 requests than between 2600 and 2600 requests. Based

Algorithm 4 Schedule Pool Updating

```
Input  : A vehicle $v$ with its previous optimal schedule pool $\digamma^*_{v,prev}$ and a set of new requests $R_{new}$.
Output: The new optimal schedule pool $\digamma^*_v$.
 1: $\digamma_{v,prev} \leftarrow UpdatePreviousSchedule(\digamma^*_{v,prev}, v)$;
 2: $\digamma^k_v \leftarrow \emptyset, \forall k \in \{0, 1, \dots, \kappa_v\}$;
 3: $\digamma^1_v \leftarrow Size1ScheduleSet(v, R_{new}) \cup \digamma^1_{v,prev}$;
 4: $k \leftarrow 2$;
 5: while $\digamma^{k-1}_v \neq \emptyset$ and $k <= \kappa_v$ do
 6:     $n_{new\_trip} = len(\digamma^{k-1}_v) - len(\digamma^{k-1}_{v,prev})$;
 7:     for $S_{v,\Gamma_i} \in \digamma^{k-1}_v[0 : n_{new\_trip}]$ do
 8:         for $S_{v,\Gamma_j} \in \digamma^{k-1}_v[1 :]$ do
 9:             Denote $\Gamma_i \cup \Gamma_j = \Gamma^k = \Gamma_i \cup \{r_{new}\}$;
10:             $\digamma^k_v \leftarrow \digamma^k_v \cup SearchFeasibleScheduleSet(S_{v,\Gamma_i}, r_{new})$;
11:     $\digamma^k_v \leftarrow \digamma^k_v \cup \digamma^k_{v,prev}$;
12:     $k \leftarrow k + 1$;
13: $\digamma_v \leftarrow \digamma^1_v \cup \dots \cup \digamma^{\kappa_v}_v$;
14: $\digamma^*_v \leftarrow ExtractOptimalSchedule(\digamma_v)$;
```

on the following observation, $\digamma^*_v$, as generated by Algorithm 4, still contains all possible optimal schedules and no feasible vehicle-trip match is missed. Lemma 3 describes a fact that if no feasible schedule is found for a request r at $t-\Delta T$, it means that there is no vehicle available to serve r in meeting the delay constraints, in either a solo trip or ride-sharing; then no vehicle will be able to serve him at t either, unless some onboard passengers are kicked out of the vehicle to free up seats.

Lemma 3 *A schedule $s_{v,\Gamma}$ can be feasible at time t only if it is feasible at $t - \Delta T$. Thus, a* *feasible optimal schedule* $s^*_{v,\Gamma}$ $(\forall\ \Gamma \subseteq R_{prev})$ *at time* t *must is included in* $\mathcal{F}^*_{v,prev}$.

With re-optimization, an SAMoD system can find a balance between the response time for requests and the optimality of the system performance. A small period yields a short response time, whereas considering all known requests makes the assignment policy optimal at any given time.

3.3.4 Constrained Allocating Based on Optimal Schedule Pool

After all candidate schedules have been generated for the vehicle fleet, the dispatcher will select a set of schedules for allocation to minimize Equation (3.1). Figure 3.7 shows an example with three vehicles and three requests. The dispatcher finds 13 candidate schedules, each one indicating a feasible vehicle-trip match, and selects s_{v_1,r_1,r_2} and s_{v_3,r_3} for assignments. Vehicle v_2 is assigned to follow its current schedule or remain idle.

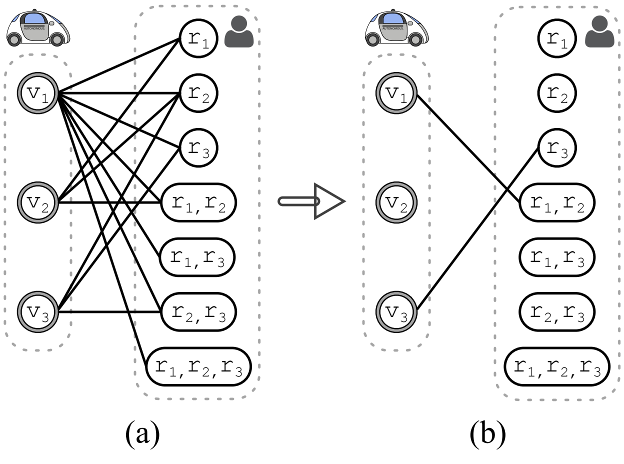


Figure 3.7: An example of constrained allocating for three vehicles and three requests. (a) The generated optimal schedule pool, where each schedule is represented by a link that connects a vehicle and a trip. (b) The final assignment made by the dispatcher.

Given a vehicle fleet V, a set of previously matched yet not served requests R_{prev}, a batch of new incoming requests R_{new}, and an optimal schedule pool $\mathcal{F}^*_v$, the dispatcher needs to assign each vehicle v a particular schedule s and the goal is to serve as many requests as possible and minimize the overall travel cost raised by ride-sharing. This is formulated as an Integer Linear Program (ILP), as shown in the following, because the assignment is all about deciding whether each candidate should be selected or ignored, and the constraints are linear. An ILP formulation can model this problem very well and can be solved very efficiently. Additionally, an ILP formulation provides flexibility to add or remove constraints. For example, by simply removing constraint (3.5) in the following formulation, the dispatcher will be allowed to reject previously matched requests for a higher service rate.

$$\underset{x_{v,s},\epsilon_r}{\operatorname{argmin}} \quad \sum_{v \in V} \sum_{s \in \mathcal{F}_v^*} x_{v,s} \cdot c(s) + \sum_{r \in R_{new} \cup R_{prev}} \epsilon_r \cdot p_{miss} \tag{3.2}$$

$$s.t. \quad \sum_{s \in \mathcal{F}_v^*} x_{v,s} = 1, \quad \forall v \in V \tag{3.3}$$

$$\sum_{v \in V} \sum_{s \in \mathcal{F}_v^*} x_{v,s} \cdot \Theta_s(r) + \epsilon_r = 1, \quad \forall r \in R_{prev} \cup R_{new} \tag{3.4}$$

$$\epsilon_r = 0, \quad \forall r \in R_{prev} \tag{3.5}$$

where $c(s)$ is the cost of the planned schedule s, p_{miss} is a very large penalty for rejecting a request in R_{new} and $\Theta_s(r)$ is an indicator function, i.e., $\Theta_s(r) = 0$ if $r \notin s$ and $\Theta_s(r) = 1$ otherwise. Binary variable $x_{v,s} \in \{0, 1\}$ indicates whether a schedule is ignored ($x_{v,s} = 0$) or assigned ($x_{v,s} = 1$) to vehicle v. Binary variable $\epsilon_r \in \{0, 1\}$ indicates whether a request is assigned ($\epsilon_r = 0$) or ignored ($\epsilon_r = 1$). Constraint (3.3) enforces that each vehicle is assigned exactly one optimal schedule, constraint (3.4) enforces that each request can only be assigned to at most one vehicle, and constraint (3.5) enforces that no previously assigned requests will be rejected even if serving them yields a higher system cost. The cost of a schedule can be the mileage driven, the travel time or the revenue [26]. In this chapter, the cost is defined as the sum of wait times and travel delays of the requests: $c(s) = \sum_{r \in s}(\omega_r + \delta_r)$.

3.3.5 Discussion of Optimality and Correctness

The generation of the optimal schedule pool explicitly explores all feasible combinations of currently received requests $R_{new} \cup R_{prev}$ and vehicles V. Yielded by pruning out the infeasible schedules and reducing repetitive searches, the savings of redundant computations make it possible to do this generation in real-time at a city-scale, i.e., the time consumed by the generation is shorter than the dispatch window length. The computation of the optimal schedule for each trip ensures that no vehicle-trip match is mistakenly ignored. Then, solving the ILP presented above considers all possible assignment policies and returns the one that produces the minimal value of the cost function, i.e., Equation (3.2).

3.4 Experimental Study

In this section, our method is evaluated and compared to the leading online dispatch algorithm in [19] and two other representative algorithms [26, 27]. Since existing implementations are unavailable, we reimplement all of the algorithms and run them on the same machine to ensure a fair comparison. The performance measures include the service rate (i.e., percentage of requests served), the response time (i.e., mean computation time per batch period epoch), and the number of candidates found at each epoch.

Experiments are designed to evaluate the following hypotheses: (H3.1) our OSP dispatcher can compute a complete candidate space in real-time and therefore has higher service rates than the state-of-the-art (i.e., [19]), especially when the problem becomes more complexity, e.g., high capacity and long delay constraints; (H3.2) our approach also scales well to larger instances; and (H3.3) the improvement in service rate for our approach over the simple ride-sharing algorithm is mainly contributed by the re-optimization, which relies on having a complete candidate space.

3.4.1 Simulation Details

The experiments are conducted on the the largest public taxi dataset from New York City [111], which has been widely used in existing ride-sharing studies [7, 19, 26, 31, 32, 109]. The complete road network of Manhattan presented in [7] is used here, containing 4,091 nodes and 9,452 directed edges. We extracted request data from three Wednesdays: 11th, 18th and 25th May in 2016. These have similar characteristics and we use them to synthesize five scenarios of varying size (400k–800k requests) to test the scalability of these algorithms.

We simulate a ride-sharing environment following the settings in [19]. The initial locations of vehicles are uniformly distributed over the road network at the start of the experiment. Daily mean travel time for each road segment is used to calculate vehicle movement. For detailed methods that computes the travel time based on data from a large amount of taxi trips, we refer the reader to Santi et al. [7], Wang et al. [110], and Bertsimas et al. [112]. The shortest paths between all nodes in the road network are pre-computed and stored in look-up tables. So that the density of the network does not affect the online running time of the algorithms. We assume the travel times on road segments are deterministic and not affected by the dispatch policy. It is reasonable and commonly used in the literature, as the number of taxi trips is a small percentage of all vehicle trips in the city. For example, the number of taxi trips in London is less than one-tenth of the number of private car trips [113]. But when SAMoD systems become dominant in urban traffic, this assumption needs to be extended to tackle congestion-aware dispatch [33, 114]. Table 3.1 summarizes the major experimental parameters. As well as increasing the number of requests, we also increase the fleet size to have instances of realistic scale. The maximum travel delay is set to be twice the maximum wait time $\Lambda = 2\Omega$. For comparing performances with varying capacities, we simulate the entire day. Then, for other parameters, we run the simulation for the hour with peak demand (19:00-20:00). This is the most challenging part and is enough to show the characteristics of the algorithms. All experiments were run on a machine with 56 Intel(R) Xeon(R) E5 2.60GHz processors and 512GB RAM in Python 3.7. The simulation implementation is single-threaded and the results are averaged over five experiment runs.

Table 3.1: Parameter settings (defaults in bold).

Parameters	Settings
Instance Scale ($\lvert R\rvert, \lvert V\rvert$)	**(400k, 2000)**, (500k, 2300), (600k, 2600), (700k, 2900), (800k, 3200)
Vehicle Capacity κ	2, 4, 6, **8**, 10
Maximum Waiting Time Ω (sec)	120, 180, 240, **300**, 360, 420
Batch Period ΔT (sec)	2, 5, 10, **30**, 60, 120

3.4.2 Algorithm Comparison

Our OSP algorithm is compared with the following representative algorithms:

- GI (Greedy Insertion) [26]: This assigns requests sequentially to the best available vehicle in a FCFS manner (i.e., an exhaustive version of [24]).
- SBA (Single-request Batch Assignment) [27]: This takes the new requests for a batch period and assigns them together in a one-to-one match manner, where at most one new request is assigned to a single vehicle.
- RTV (Request Trip Vehicle graph) [19]: This is a more sophisticated batch assignment algorithm that copes with multiple request assignments during the same batch period. It uses ad hoc heuristics (e.g., only 30 candidate vehicles for each request, finding optimal schedules for trips up to size 4 and a timeout of 0.2 sec per vehicle to search for feasible trips) to make it run in real-time.

All algorithms are equipped with the same simple rebalancing method from [19], which repositions idle vehicles to the nearest locations of unassigned requests, under the assumption that more passengers are likely to appear in the zones where not all requests can be served.

3.4.3 Results

(H3.1) Figure 3.8 shows the results of varying vehicle capacities. As the capacity increases, all algorithms provide better service rates. Compared to double-occupancy, high-capacity ride-sharing significantly increases the service rate. However, 10-capacity only brings a 0.21% improvement over 8-capacity, which is negligible compared to the increase in vehicle investment. All batch assignment methods outperform GI. Our OSP algorithm has the highest service rate and shows a 3.85% improvement over SBA at capacity 10. RTV is marginally worse than OSP when the capacity is less than 6, but it only achieves a 1.99% higher service rate than SBA at 10-capacity, which is only around half of the improvement brought by OSP. SBA outperforms GI more at double-occupancy than high capacity, because two seats would rarely allow multiple request assignments happen during the same batch period. The gap between SBA and RTV/OSP at double-occupancy is mainly caused by re-optimization. The response times of GI and SBA are almost unchanged at varying capacity, while that of RTV and OSP increase linearly.

(H3.1) Figure 3.9 plots the results of varying values of the maximum waiting time constraint. With longer waiting times, the service rates of all algorithms increase. The reason is that a longer waiting time and travel delay allows a larger detour, and thus more requests are served. Batch assignment approaches achieve significantly higher service rates than GI when the maximum waiting time is low. But while the tolerance to delay increases, the ability to leverage complex combinations of requests and their optimal schedules is needed to maintain this advantage. The response time of GI and SBA is still stable. While the

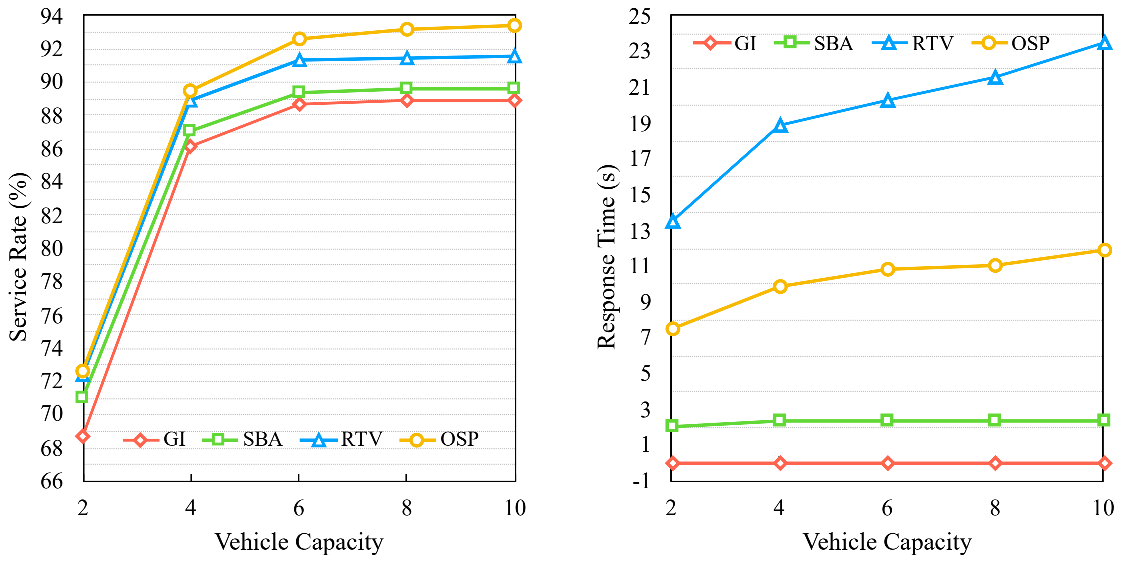


Figure 3.8: A comparison of performance metrics during the whole day for varying vehicle capacities ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

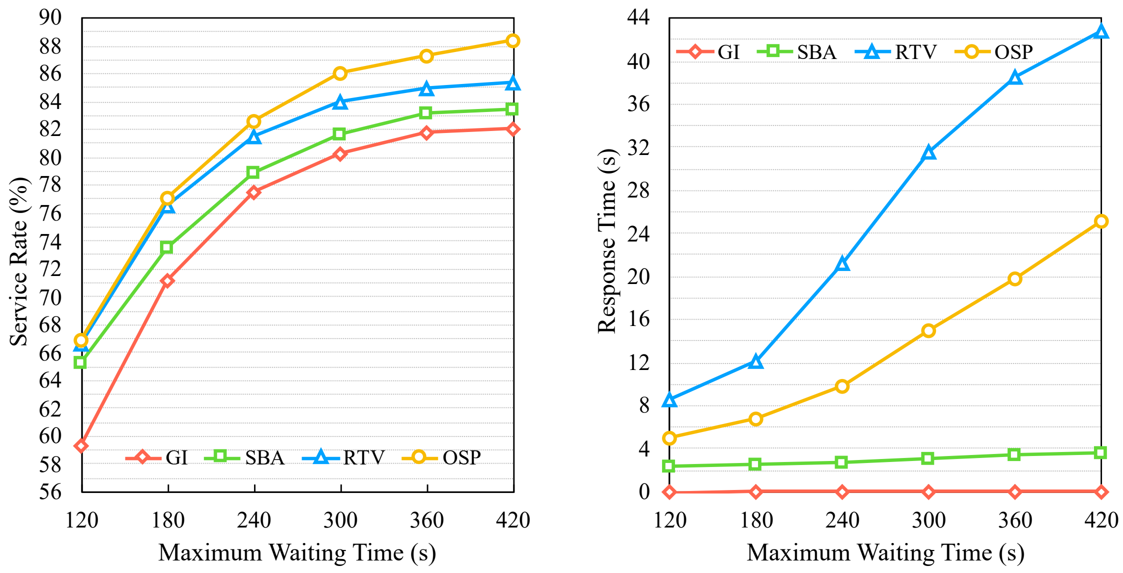


Figure 3.9: A comparison of performance metrics during the peak hour for varying values of maximum waiting time constraint ($|R| = 400k,\ |V| = 2000,\ \kappa = 8,\ \Delta T = 30\,\text{sec}$).

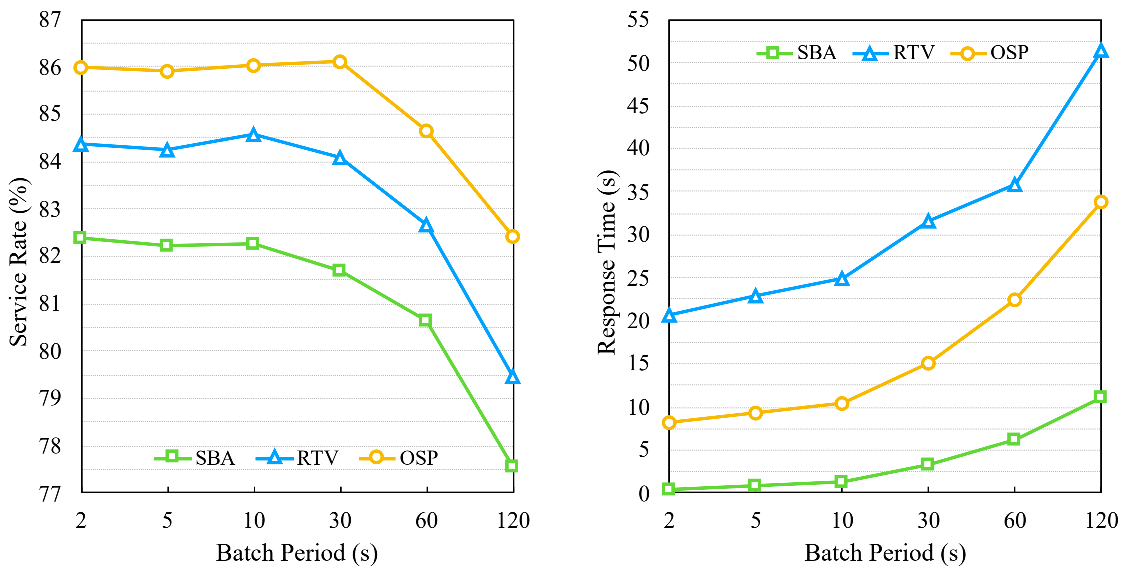


Figure 3.10: A comparison of performance metrics during the peak hour for varying lengths of batch period ($|R| = 400k, |V| = 2000, \kappa = 8, \Omega = 300\,\text{sec}$).

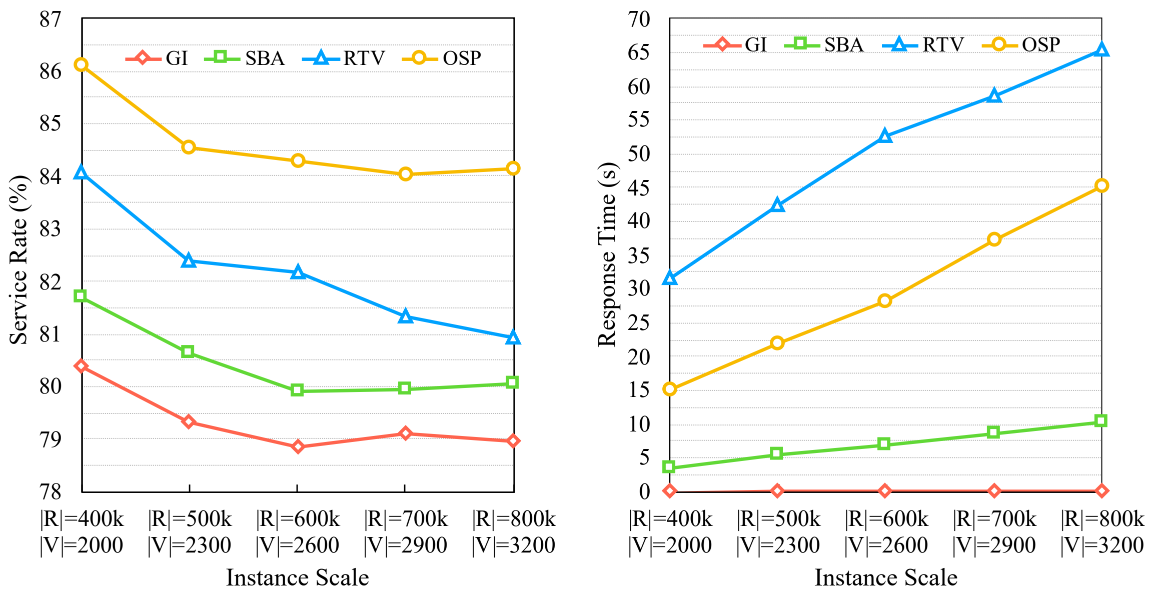


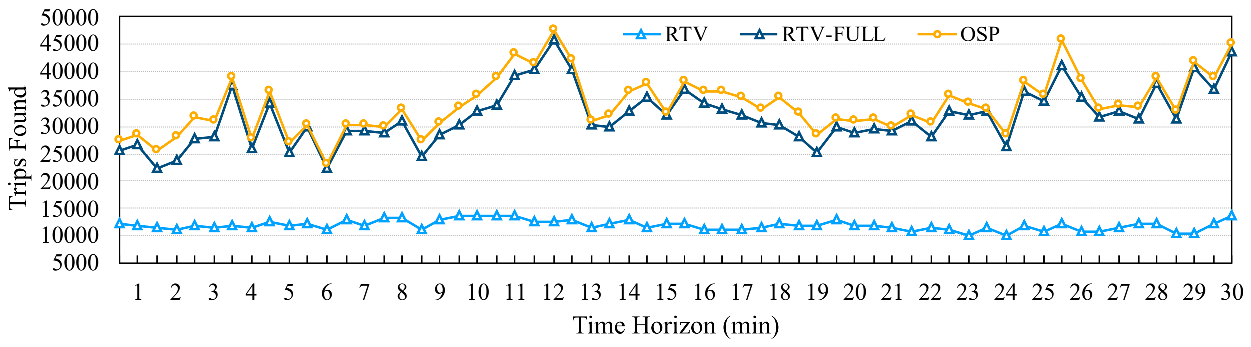
Figure 3.11: A comparison of performance metrics during the peak hour for varying instance scales ($\kappa = 8,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

running time of OSP grows almost linearly, that of RTV increases exponentially. The rate of growth in RTV's running time decreases when $\Omega > 300\,\text{sec}$ because the time limit slows down the increase in computation time and in turn results in a bad performance.

(H3.1) Figure 3.10 shows the results of varying lengths of batch period. With a larger period, more requests are processed at each epoch and thus the response time of batch approaches grows. Considering more requests over the same period could theoretically have a better matching quality. However, a longer period yields a longer wait and a lower detour tolerance in practice, and results in a lower service rate. When the batch period is longer than 60 s, the service rate decreases significantly.

(H3.2) Figure 3.11 plots the results of varying instance scales. All algorithms scale well to large problem instances with a linear increase in response time. However, the service rate of RTV has a considerable decrease compared to others. The difference between OSP and RTV is expanded from 2.04% to 3.21%. This indicates that, although the ad hoc heuristics used by RTV ensure a good scalability on response time, as optimality cannot be guaranteed, they yield a worse matching performance at large instance scales.

(H3.1) Figure 3.12 shows the total number of feasible trips found and the number of matched new requests by different algorithms from 19:30 to 20:00, where the average number of incoming new requests is 438 per 30 sec. In each dispatch period, all algorithms are fed with the exact same requests and vehicles to remove other distractions (e.g. the position distribution of requests). RTV-FULL, without time limit and allowed to keep every feasible vehicle for each request, is introduced to provide a comprehensive comparison between OSP and RTV. It can be seen that OSP has the best performance at each dispatch epoch, as it finds every feasible vehicle-trip match, along with the optimal schedule, and makes the best match based on the complete candidate space. For all the requests that are waiting to be picked up, RTV, RTV-FULL and OSP find an average of 11,954, 31,792 and 33,984 feasible



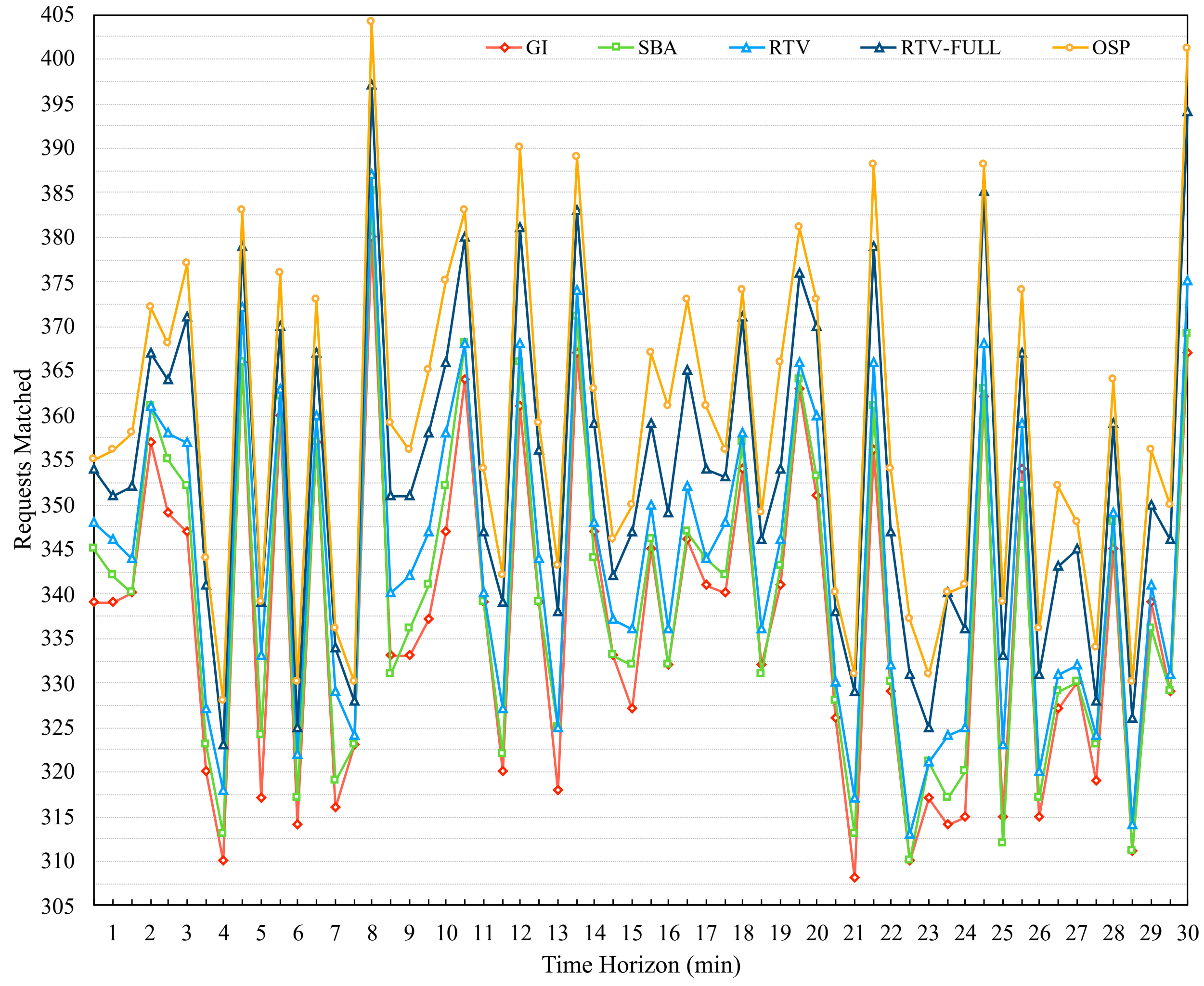


Figure 3.12: Number of feasible trips, each of them representing a candidate vehicle-trip match, found for all vehicles (upper plot) and number of matched new submitted requests (lower plot) at different dispatch epochs during 19:30-20:00 ($|R| = 800k,\ |V| = 3200,\ \kappa = 8,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

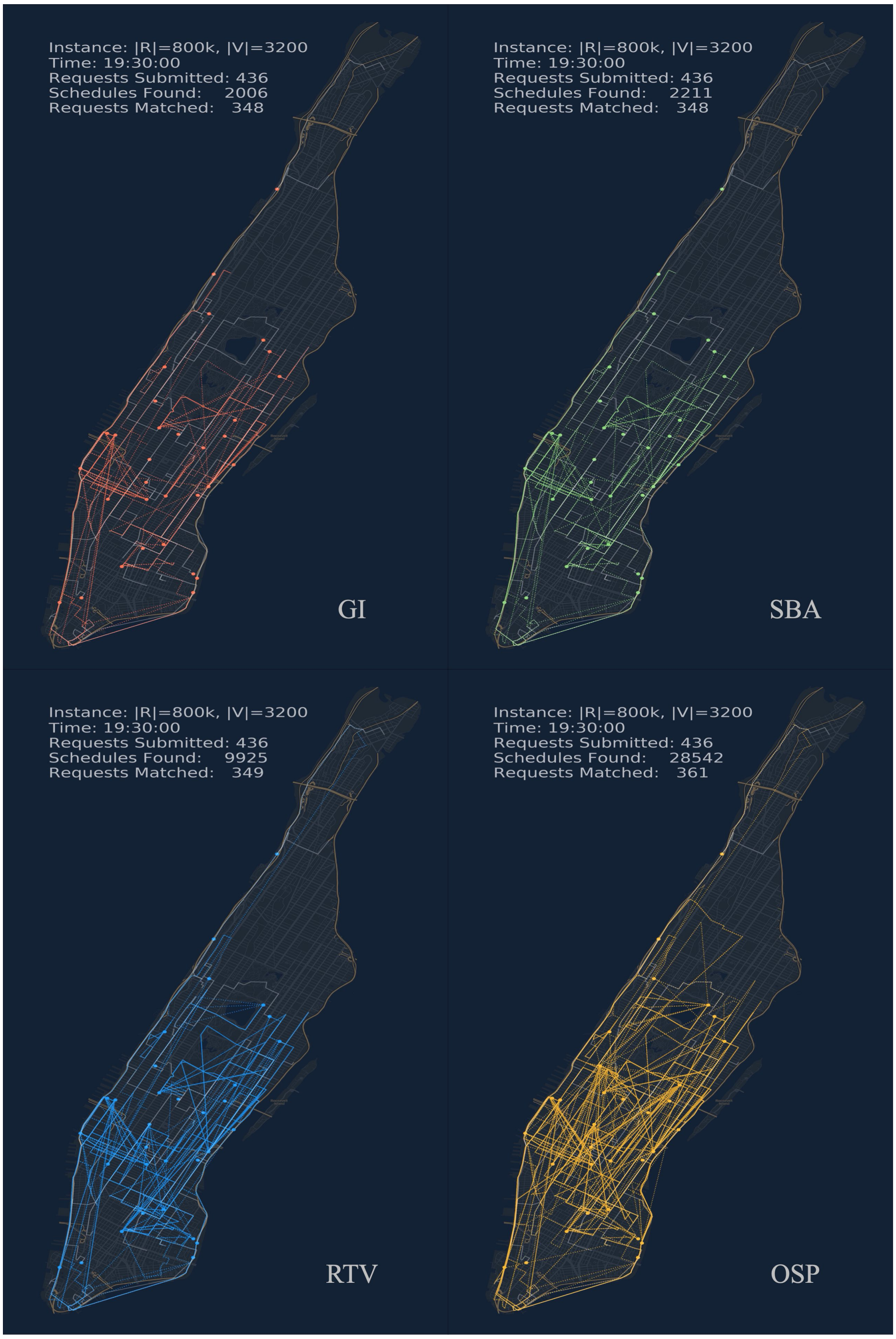


Figure 3.13: A visualization of candidate vehicle-trip matches found for 32 vehicles by different algorithms at 19:30 ($|R| = 800k,\ |V| = 3200,\ \kappa = 8,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

trips. The difference between OSP and RTV-FULL is caused by the fact that RTV-FULL only does exhaustive search for trips up to size 4. Despite this, the response time of RTV-FULL is 763.98 sec, while that of RTV and OSP are 65.48 sec and 45.38 sec, respectively. If we allow RTV-FULL to do exhaustive search for size 5 trips, the response time increases to an incredible 3,734.77 sec.

(H3.1) Figure 3.13 shows a comparison of the number of candidate matches found by different algorithms when fed the same information about requests and vehicles at 19:30. The submission times of requests are in the 30 sec time window before 19:30. The computation is carried out with 3,200 vehicles, but only 32 of those vehicles are plotted for visualization purposes. During this epoch, the system receives 436 new requests. By optimizing across multiple epochs and allowing many-to-one matching, OSP finds 28,542 candidate vehicle-trip matches, while that of RTV, SBA and GI are only 9,925, 2,211 and 2,006, respectively. Eventually, OSP serves 361 of 436 requests, and outperforms RTV, SBA and GI by 12, 13 and 13 requests, respectively.

Table 3.2: Number of schedules searched by different algorithms

Algorithms	Trip Size							
	3	4	5	6	7	8	9	10
Exhaustive Search	54	1,150	6,983	53,110	272,810	N/A	N/A	N/A
OSP	11	50	159	433	595	776	1468	1920

(H3.1) We further investigate the number of possible schedules considered by OSP to see how much of the search space is pruned compared to exhaustive search. Table 3.2 shows the counts from trips of size 3 to 7. Exhaustive search is unable to complete the search when the trip size is larger than 7, while OSP only considers 1,920 schedules for size 10 trips.

(H3.3) Finally, we investigate the role of allowing many-to-one match for requests

Table 3.3: A comparison of service rate (%) during the peak hour for varying instance scales ($\kappa = 8,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Algorithms	Instance Scales ($\lvert R\rvert, \lvert V\rvert$)				
	(400k, 2000)	(500k, 2300)	(600k, 2600)	(700k, 2900)	(800k, 3200)
SBA	81.69	80.64	79.92	79.96	80.04
OSP-NR	82.24	81.21	80.67	80.53	80.56
OSP	86.12	84.55	84.30	84.05	84.14
OSP-FR	87.20	85.48	85.39	85.04	85.16

during the same dispatch epoch and the role of re-optimization across multiple epochs. Two variants of OSP are introduced: (1)OSP-NR (No Re-optimization), only considering the requests, R_{new}, received at the current epoch to remove the effect of re-optimization; (2) OSP-FR (Full Re-optimization), removing the constraint (3.5) to maximize the system performance by allowing the rejection of assigned requests. Table 3.3 shows the service rates of SBA, OSP-NR, OSP and OSP-FR, for varying instance scales. Compared to SBA, OSP-NR only achieves an average of 0.6% increase in the service rate, while that of OSP is 4.18%. OSP-FR further yields an average of 1.02% improvement over OSP. Therefore, re-optimization, considering requests from multiple epochs together, is the key to improving the performance of dispatch. But re-optimization requires a complete search for candidate vehicle-trip matches, so as to make the optimal assignment; otherwise, when making an assignment based on an incomplete candidate space, worse results will come when instance scaling grows, as is shown by the performance of RTV in Figure 3.11.

3.4.4 Discussion

Batch assignment provides better performance than the FCFS based approach, by assigning requests together. During the peak hour, solely using batch assignment, SBA yields an improvement of around 1% in the service rate compared to GI. Our proposed approach, OSP, further achieves a service rate about 4% higher than SBA by optimizing the dispatch policy across multiple epochs; while that of RTV, which cannot generate a complete candidate space, is only around 2% higher than SBA. Our OSP's optimal scheduling can achieve twice the improvement of RTV over SBA. The improvement of OSP relative to RTV also increases with the complexity of the instance. If OSP is allowed to reject previously assigned requests to maximize the effectiveness of the system, it could yield an improvement over SBA in the service rate of around 5%.

The incremental schedule search method and the iterative re-optimization strategy together reduce the scheduling space to less than one percent of that of exhaustive search, when the ride-sharing size is larger than six. These work as the foundation for OSP by computing the complete candidate space in real-time. The computation of the schedule pool is naturally parallelisable across different vehicles, thus OSP can be deployed in real-time with even larger instance scales.

3.5 Chapter Summary

In this chapter, we have proposed an optimal online dispatch method for high capacity SAMoD systems. Our work aims to study optimal batch assignment for SAMoD systems running in real-time, and tries to reach the upper-bound of performance for reactive dispatch. An incremental search method is developed to speed up computing schedules and find all feasible vehicle-trip matches by retaining the completed search space. An iterative re-optimization strategy is then developed to efficiently find better ride-sharing trips by looking at long-term combinations of requests while keeping a prompt response to new requests. Numerical experiments on real large-scale datasets show the proposed method improves the state-of-the-art in terms of service rate (up to 3.21% at peak hour) and system scalability. Typically, a 1% improvement is considered significant on real taxi systems [64].

Chapter Four

Vehicle Dispatch with Stochastic Travel Times

In Shared Autonomous Mobility-on-Demand (SAMoD) systems, the quality of passenger experience and the profit achieved by these platforms are strongly affected by the vehicle dispatch policy. However, existing ride-sharing research seldom considers travel time uncertainty, which leads to inaccurate dispatch allocations. In this chapter, we present a general framework for dynamic vehicle dispatch that leverages stochastic travel time models to improve the performance of a fleet of shared vehicles. The novelty of this work includes: (1) a stochastic on-demand ride-sharing scheme to maximize the service rate (i.e., percentage of requests served) and reliability (i.e., probability of on-time arrival); (2) a technique based on approximate stochastic shortest path algorithms to compute the reliability for a ride-sharing trip; (3) a method to maximize profit when a penalty for late arrivals is introduced. Based on New York City taxi data, it is shown that by considering travel time uncertainty, on-demand ride-sharing services achieve higher service rates, reliability and profits.

4.1 Introduction

When evaluating the feasibility and benefits of schedules for ride-sharing trips, the most important consideration is the timing of passengers, because travel delay is a more constraining factor than others, e.g. spare capacity [40]. To have accurate estimations of the benefits from on-demand ride-sharing and make the best vehicle dispatch policy, the following technical challenges need to be properly tackled:

1. *Travel time uncertainty.* Due to various stochastic factors, travel times on roads can exhibit considerable variability in urban environments. Solutions based on deterministic routing may deteriorate when deployed in scenarios with stochastic travel times.

2. *Reliability estimation.* In reality, vehicles may not arrive at their planned destinations on time, i.e, exceeding the delay times of passengers. Passengers are more concerned about the reliability of their trip than its duration. A high frequency of violating the delay constraint would result in the loss of passengers. Thus, estimation of the probability of satisfying the delay constraint is needed to optimize reliability.

3. *Profit estimation.* Passengers may accept uncertainty if compensation is granted for exceeding the delay times. However, the prices of trips are normally determined immediately after the submission of requests and remain fixed, regardless of the allocation policy or the actual travel time, while the travel costs and compensations are affected by variability in travel times. Thus an assignment policy based on inaccurate estimations of trips' profits may differ from the profit-optimal solution.

Ride-sharing has been receiving increased attention from the academic community. However, most previous studies fail to consider the uncertainty of travel time. They only compute assignments and routes with the shortest expected time for the sake of simplicity, and ignore the reliability of on-time arrival, which is of high value for passengers. When addressing the uncertainty of travel time, the optimal dispatch policy may change. Figure 4.1 shows an example.

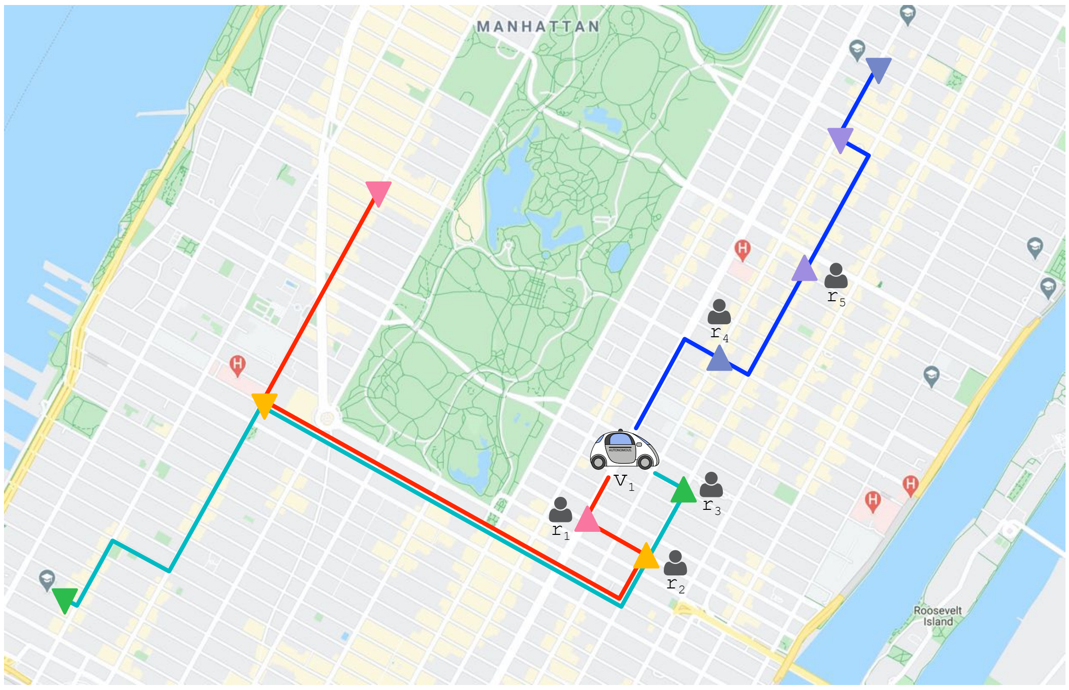


Figure 4.1: An example with five requests r_1, r_2, r_3, r_4, r_5 and one vehicle v_1. Triangles ($\triangle$) and inverted triangles ($\triangledown$) represent the origins and destinations, respectively. There are three possible dispatch policies for v_1 and they exhibit no difference in service rate when travel time uncertainty is not considered. When taking stochastic travel time into account, it may be found that serving r_2 and r_3 is the best allocation as it has the highest reliability in terms of arrival. Or, considering the penalty costs due to late arrivals, serving r_1 and r_2 yields the highest profit and is the best allocation to make.

A similar approach to our work, which also considers on-time arrival reliability in on-demand ride-sharing, is presented in [85]. They introduce a reliable path concept for selecting a vehicle with the maximum reliability for each request. However, the reliable path is computed from the precomputed k-shortest paths, which may not be the optimal one. Also, only pairwise sharing is allowed and the fleet size considered is less than 200.

Our work differs from [85] by providing high-capacity on-demand ride-sharing. For example, we can dispatch a fleet of 3000 vehicles with a capacity of 6. The prices of requests and the profits from trips, which are important concerns for the platform, have also received little attention. A dispatcher considering request price is developed in [60] to maximize the platform's profit. But it does not take into account the penalty costs arising from failing to arrive on time. In contrast, we optimize profit with stochastic travel times.

In this chapter, we study the role of travel time uncertainty in on-demand ride-sharing services and propose a multi-phase constrained optimization scheme, which considers stochastic travel time information to increase the passengers' on-time arrival reliabilities, and further incorporates the concept of late arrival compensation to optimize the platform's profit. To summarize the work in this chapter, we:

1. Develop a method for Reliability-aware Vehicle Dispatch (RVD), which takes travel time uncertainty into account in both the allocating of requests to vehicles and the routing of vehicles. (Section 4.2.3.)
2. Develop an algorithm to estimate the probability of on-time arrival for ride-sharing trips (i.e., satisfying the delay constraint), which is then incorporated into the main dispatch method to optimize the reliability of service. (Section 4.3.)
3. Develop a method for Profit-aware Vehicle Dispatch (PVD), which estimates the profit of a trip and optimizes the profit of the platform, considering the penalty costs due to late arrivals, i.e., exceeding the delay constraint. (Section 4.4.)
4. Conduct a case study on Manhattan data to investigate the benefits of our travel time uncertainty aware dispatchers and compare them to the state-of-the-art deterministic approach. (Section 4.5.)

The remaining content of the chapter is arranged as follows. In Section 4.2, we introduce the notion of stochastic travel time, the problem statement and a multi-phase planning scheme for reliability optimization. In Section 4.3, we present the details of reliability estimation and allocation. In Section 4.4, we introduce the concept of late arrival compensation and attempt to maximize the profit of vehicle dispatch based on the multi-phase planning scheme. In Section 4.5, we evaluate our two approaches, RVD and PVD, using real-world taxi data. In Section 4.6, we draw our conclusions.

4.2 Preliminaries

Similar to the approach presented in Chapter 3, the dispatchers proposed in this Chapter solve a static problem repeatedly in a rolling-horizon framework, i.e., batch planning. For the sake of brevity, the notion of re-optimization is not duplicated in this chapter, but re-optimization is used.

4.2.1 Definitions

Following the definitions in Section 3.2 of Chapter 3, a dispatcher assigns requests at a time epoch ΔT and batches a set of n new requests R. It considers a fleet of m vehicles V and computes a candidate schedule pool $\digamma_v$ for each vehicle, indicating all possible vehicle-trip matches. A trip, denoted by $\Gamma \subseteq R$, is a set of requests that can be merged using ride-sharing. A schedule, denoted by $s_{v,\Gamma} = \{o_1, \ldots, d_1, \ldots, o_2, \ldots, d_{n_\Gamma}\} \in \digamma_v$, is a sequence of visiting positions for a vehicle v to pick up and drop off requests in a ride-sharing trip Γ with minimum cost, i.e., the lowest expected travel delays for passengers. The expression "optimal schedule", defined in Section 3.2 of Chapter 3, is not used in this chapter. Because the minimum-cost schedule may not be the optimal one when considering travel time uncertainty. For a schedule to be feasible, it must satisfy the capacity (κ), waiting time (Ω), total delay

(Λ) and precedence constraints.

Next, we introduce the notion of vehicle travel with stochastic travel times. Following the assigned schedules, vehicles travel on a predefined road network $G = (I, E)$. We define two functions, $\tau(i_1, i_2)$ and $\varrho(i_1, i_2)$, to compute the mean travel time and the travel distance of the minimum expected time path from i_1 to i_2, respectively. For the sake of simplicity, the travel time for each edge is assumed to follow an independent Gaussian distribution $N(\mu_e, \sigma_e^2)$. This assumption is commonly used in stochastic networks [71, 85, 115–117]. It is shown in [71] that the independent Gaussian assumption is very similar to real world taxi trajectories in Massachusetts and holds well for stochastic planning. The multi-phase dispatch scheme introduced in the following sections is not limited to the independent Gaussian assumption, and can be further enhanced by incorporating other fast stochastic routing algorithms for more accurate estimations, such as [118, 119].

For a specific schedule $s_{v,\Gamma}$, the detailed route that travels from q_v to d_{n_Γ} is denoted by π. A portion of the route that travels from q_v to d_r ($r \in \Gamma$) is denoted by $\pi(r)$. There is more than one possible route for a specific schedule, of which the optimal one is denoted by π^*, i.e., the one serving the requests with the highest reliability.

4.2.2 Problem Statement

Besides maximizing the service rate of the service, taking travel time uncertainty into consideration, we also optimize the overall reliability for all passengers. Using R_{miss} to denote the set of requests that cannot be served at the current dispatch epoch, we define the objective of Reliability-aware Vehicle Dispatch (RVD):

$$O_{Reliab} = \sum_{v \in V} best\_prob(s_v) - \sum_{r \in R_{miss}} p_{miss} \tag{4.1}$$

where $best\_prob(s_v)$ is the maximum expected mean probability of dropping off requests on time (i.e., satisfying the delay constraint) for a schedule and is affected by the routing policy (discussed in Section 4.3.1) and p_{miss} is a large cost for not serving a request. Equation (4.1) works on reliability optimization by finding the best matching policy that serves the largest number of passengers with the highest probability of on-time arrival.

Problem 2 (Reliability-aware dispatch) *Given a set of requests R and a set of vehicles V at a dispatch epoch with a length of ΔT, the problem of reliability-aware dispatch is to assign vehicles particular candidate schedules to serve as many requests as possible with the highest reliability, so that the objective (i.e., Equation (4.1)) is maximized, subject to the following constraints: (1) Each request must be served by exactly one vehicle. (2) Each schedule must be feasible when considering the mean travel time.*

4.2.3 Multi-Phase Dispatch Scheme

Figure 4.2 illustrates our proposed scheme to solve Problem 2. It is a multi-phase method that breaks down the computational burden and can work as a general scheme for different optimization purposes. The multi-phase method builds on the method in Chapter 3, which works well on searching for candidate schedules (representing vehicle-trip matches). The processes for schedule generation (b) and request allocation (d) are inspired by [19] and have been discussed in Chapter 3. The stochastic travel time information is incorporated in schedule scoring (c) and vehicle routing (e). The reliability of a schedule is affected by the routing plan and the term $\sum_{v \in V} best\_prob(s_v)$ in Equation (4.1) is affected by both the

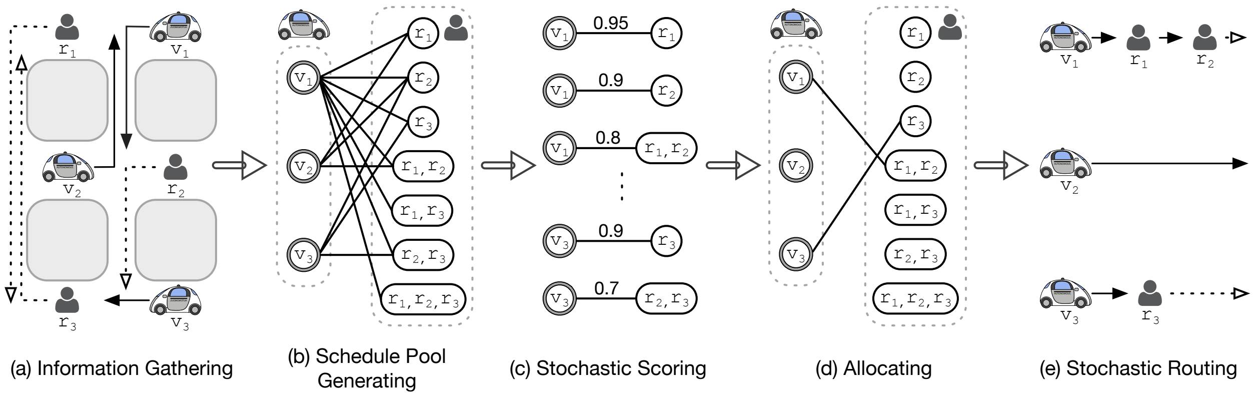


Figure 4.2: Schematic overview of our proposed multi-phase approach. (a) An epoch with three vehicles and three requests. The solid lines present the current planned routes for vehicles and the dashed lines present the shortest paths for requests. (b) A candidate schedule pool that connects vehicles to servable ride-sharing trips. Each link represents a schedule. (c) Scored candidate schedules with reliability information. Each schedule is associated with its stochastic optimal route. (d) Allocation of requests to vehicles that maximizes the sores, where requests r_1 and r_2 are served by vehicle v_1 and request r_3 is served by vehicle v_3. (e) Vehicles travelling on the stochastic optimal routes following the assigned schedules.

routing plan and the allocating results, which cannot be omitted. The main steps of the method are:

- Based on the locations of requests and vehicles, the dispatcher computes feasible ride-sharing combinations for each vehicle to generate the candidate schedule pool $\mathcal{F}_v$. This is done by using the mean travel time to be efficient. Either the OSP method discussed in Chapter 3, the RTV method proposed in [19], or other candidate vehicle-trip match search methods can be employed for the generation of $\mathcal{F}_v$.

- Using the travel time distribution information, the dispatcher scores each candidate schedule by solving a stochastic routing problem that maximizes the mean on-time arrival probability. Additionally, the associated minimum mean travel time route for each schedule is changed to the stochastic optimal one. A method to do so is discussed later in Section 4.3.1.

- Considering the scored candidate schedules, the dispatcher allocates requests by solving a maximum weight matching problem that maximizes the number of allocated requests and reliability. This is formulated as an ILP and presented in Section 4.3.2
- Following assigned schedules, the dispatcher routes vehicles along the stochastic optimal routes to pick up and drop off requests.

As stated in [19] and discussed in Section 3.3.5 of Chapter 3, given enough time, all possible vehicle-trip matches can be found and the ILP-based allocations are optimal. However, as travel time distributions are not considered when generating the schedule pool, there are two limitations to the current multi-phase scheme. (1) Some potentially feasible vehicle-trip matches may be mistakenly ignored [83]. In practice, longer delay constraints can be set so that searching for feasible ride-sharing combinations has a probabilistic guarantee. For example, setting the total travel delay constraint as $\Lambda^{'} = \Lambda + 3 \cdot \sigma_{\pi}$, where σ_{π} is the largest standard variance of any possible ride-sharing route, will mean that less than 0.27% trips can be ignored. Consequently, a longer computation time and more candidate vehicle-trip matches will be produced. (2) The optimal schedule for a candidate vehicle-trip match at the scoring step may be different from the one found at the pool generation step using mean travel time. During the pool generation step, the dispatcher searches for candidate vehicle-trip matches and only keeps the minimum mean delay schedules (e.g., F_v^* in Chapter 3). However, when considering the stochastic route during the scoring step, the minimum mean delay schedule may not always be the one that has the highest probability of satisfying the delay constraints of requests. In the future, we can further couple the stochastic scoring procedure with OSP generation to search for candidate vehicle-trip matches in a unified way.

4.3 Reliability Optimization

Given a candidate schedule pool, we handle the scoring for each schedule by finding the optimal route that has the highest mean probability of dropping off multiple requests under the travel delay constraints. Then, considering the scored candidate schedule pool, an ILP is solved for allocation of requests.

4.3.1 On-time Arrival Probability Estimation

To find the route that maximizes the reliability of a ride-sharing schedule, we adapt the approximate Stochastic Shortest Path (SSP) query algorithm in [74] and the multi-hop routing algorithm in [76] to on-demand ride-sharing systems. The generalized method consists of two procedures: preprocessing and online scoring. It precomputes a set of parametric shortest paths offline and finds the optimal route online. By completing complex computations offline, we can achieve high efficiency online path finding and meet real-time requirements of vehicle dispatch. For each candidate schedule, the method computes the maximum mean probability of dropping off all requests on time for scoring (step (c) in Figure 4.2) and returns the corresponding optimal route for the routing procedure after allocation (step (e) in Figure 4.2).

Preprocessing

Estimating the reliability of a schedule relies on computing the routes that maximize the probability of on-time arrival for each request in the schedule. The problem of finding the SSP between two locations is introduced first in the preprocessing part, based on which the problem of finding the stochastic optimal route for a ride-sharing schedule is discussed in the online scoring part.

Consider the road network $G = (I, E)$ defined in Section 4.2, the edge weights of which are stochastic rather than deterministic, and given an origin o_r, a destination d_r and a deadline $\tau(o_r, d_r) + \Lambda$, SSP algorithms commonly aim to find (1) the path with the maximum probability of reaching the destination within the deadline, or (2) the path with the minimum mean time and a probability of making the deadline larger than a predefined value [72, 74, 76]. In this chapter, our goal is to find the maximum probability path that arrives within time $\tau(o_r, d_r) + \Lambda$, which is solving:

$$\begin{aligned} \pi^* &= \underset{\pi \in \Pi}{\operatorname{argmax}} \;\; Pr(travel\ time\ of\ \pi \leqslant \tau(o_r, d_r) + \Lambda) \\ &= \underset{\pi \in \Pi}{\operatorname{argmax}} \;\; \Phi(\frac{\tau(o_r, d_r) + \Lambda - \mu_\pi}{\sigma_\pi}) \end{aligned} \tag{4.2}$$

where Π is the set of all possible routes from o_r to o_d, μ_π is the mean travel time of route π, σ_π is the variance of travel time and $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution. As $\Phi(\cdot)$ is monotone increasing, finding the SSP for a request r that travels alone with a travel delay constraint Λ is equivalent to solve:

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} \;\; \frac{\tau(o_r, d_r) + \Lambda - \mu_\pi}{\sigma_\pi} \tag{4.3}$$

Equation (4.3) can be converted to a parametric optimization problem [71, 72]. Denote path π by a point (μ_π, σ_π) in a mean-variance plane, as shown in Figure 4.3, and denote $\phi(\pi) = \frac{\tau(o_r, d_r) + \Lambda - \mu_\pi}{\sigma_\pi}$, we will have:

$$\sigma_\pi^2 = \frac{1}{\phi(\pi)^2}[\mu_\pi - (\tau(o_r, d_r) + \Lambda)]^2 \tag{4.4}$$

Intuitively from Equation (4.4) and Figure 4.3, the SSP for Equation (4.3) is the one lying on the parabola with the least curvature, at which point $\phi(\pi)$ has the maximum value.

The target point (i.e., the SSP) on the mean-variance plane can be found by enumerating all of the extreme points, which is converted to iteratively finding the α-shortest path. Consider setting the weight of each edge to a parameter dependent length $\alpha \cdot \mu_e + \sigma_e^2$, the parametric shortest path for this edge weight is called the α-shortest path. The parametric

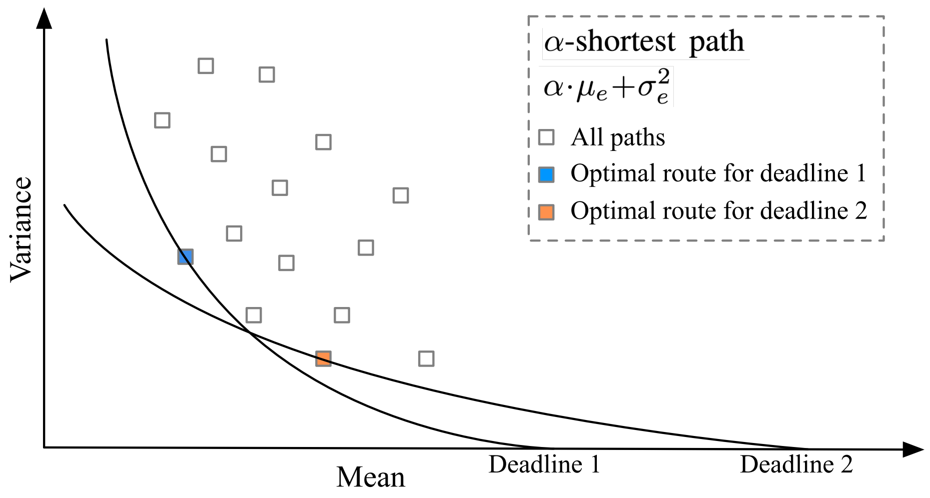


Figure 4.3: Projection of various α-shortest paths onto a mean-variance plane. Each one lies on a parabola. The optimal route is the one with the smallest curvature and has the highest probability of arriving before the deadline.

optimization discussed in [71, 72] works on finding the parameter values $\alpha \in (0, \infty)$ at which the shortest path changes, so as to enumerate the extreme points. Computing the exact solution to Equation (4.3) is computationally expensive, as there are $n^{O(\log n)}$ extreme points [120]. Also, the target α value varies when the origin-destination pair or deadline is different.

There is a $\sqrt{1 - \epsilon^2/(2 + \epsilon^2)}$-approximation algorithm that runs in time polynomial in $1/\epsilon$, for any user-specified level of accuracy $0 < \epsilon < 1$ [73, 74]. Generalized from [74], the approximate solution can be found by enumerating α-shortest paths for $\alpha \in \{L, (1+\xi)L, (1+\xi)^2 L, \ldots, U\}$. The values of ξ, L and U are determined as:

$$\xi = \epsilon/2 \tag{4.5}$$

$$L = \frac{2 \cdot \min_e \sigma_e^2}{d_L} \geqslant \frac{2 \cdot \min_e \sigma_e^2}{\max_r \tau(o_r, d_r) + \Lambda} \tag{4.6}$$

$$U = \frac{2 \cdot \sum_e \sigma_e^2}{\epsilon \cdot d_U} \leqslant \frac{2 \cdot \max_\pi \sigma_\pi^2}{\epsilon \cdot \min_e \mu_e} \tag{4.7}$$

where $\min_e \sigma_e^2$ is the smallest variance of any edge in the road network, $\max_r \tau(o_r, d_r)$ is the largest mean travel time of any single request trip, $\max_\pi \sigma_\pi^2$ is the largest variance of any possible ride-sharing route and $\min_e \mu_e$ is the smallest mean travel time of any edge.

Denoting the set of the approximation parameters α as A, we precompute the shortest paths for all values $\alpha \in A$ and store them in look-up tables. Therefore, the most reliable route for a travel alone request can be found by checking the tables $|A|$ times. In Section 4.5, the experimental study, we set $\epsilon = 0.5$ and find 21 different α values for preprocessing. According to the experimental results in [74], the approximation ratio is less than 0.02, which is a quality measure defined as the difference between the approximated solution and the optimal solution, divided by the optimal one.

Online Scoring

Similar to finding the SSP between two locations, the maximum probability of visiting a schedule's last position $s_{v,\Gamma}[-1]$ and the corresponding route can be found by enumerating α-shortest paths, as presented in [76]. But in ride-sharing, a vehicle normally visits more than one dropoff point in a single trip. Not only the on-time arrival probability of the last position, but also those of other requests' destinations need to be optimized. We consider maximizing the mean probability of on-time arrival for requests in one trip, which is denoted by:

$$mean\_prob(s_{v,\Gamma}) = \frac{1}{|\Gamma|} \sum_{r \in \Gamma} \Phi(\frac{\tau(o_r, d_r) + \Lambda - \mu_{\pi(r)}}{\sigma_{\pi(r)}}) \tag{4.8}$$

Adopted from the algorithm in [76], a method to find the α-optimal route for a schedule is described in Algorithm 5. It tries to find the best α value that maximizes Equation (4.8) by enumerating all $\alpha \in A$. The pre-route π_{pre} is a predefined route from q_v to $s_{v,\Gamma}[0]$. Function $Find\alpha ShortestPath(s_{v,\Gamma})$ returns the α-shortest path that visits a series of locations in the order determined by the schedule $s_{v,\Gamma}$. Function $ComputeMeanProbability(\Gamma, \pi)$ returns the mean probability of on-time arrival when requests are served by a predefined route π, defined as Equation (4.8).

Algorithm 5 Find α Optimal Route

```
Input  : A schedule $s_{v,\Gamma}$ and its pre-route $\pi_{pre}$.
Output: The mean probability of on-time arrival $mean\_prob$ and the $\alpha$ optimal route $\pi^*_\alpha$.
1: $mean\_prob \leftarrow 0$;
2: $\pi^*_\alpha \leftarrow \emptyset$;
3: for each $\alpha \in A$ do
4:     $\pi \leftarrow Find\alpha ShortestPath(s_{v,\Gamma}, \pi_{pre})$;
5:     $prob \leftarrow ComputeMeanProbability(\Gamma, \pi)$;
6:     if $prob > mean\_prob$ then
7:         $mean\_prob \leftarrow prob$;
8:         $\pi^*_\alpha \leftarrow \pi$;
```

However, unlike the deterministic route planning that satisfies the optimal substructure property, the optimal route π^* found by Algorithm 5, which aims to maximize Equation (4.8) with stochastic travel times, may be a concatenation of the α-optimal paths for each request with different α values. Thus, we propose a recursive algorithm to explore all possible combinations of different α values for the requests in a schedule, as shown in Algorithm 6. It is built on Algorithm 5. Function $GetRequestsInSchedule(s_{v,\Gamma})$ returns the set of requests included in the schedule $s_{v,\Gamma}$. Function $CutSchedule(d_r, s_{v,\Gamma})$ breaks the schedule into two sub-schedules based on the drop-off position of r, the reliabilities and routes of which are then computed by Algorithms 5 and 6, respectively. Besides the two sub-schedules, it also outputs the number of drop-off visiting locations in both schedules, denoted by n_1 and n_2, which will be used to calculate the mean probability in line 8. As illustrated in the recursion tree, shown in Figure 4.4, n_1 is the length of a link and n_2 is the number of remaining locations. For example, if we cut the schedule $\{q_v, \dots, n_p\}$ at the 2^{nd}

Algorithm 6 Schedule Reliability Estimation (SRE)

Input : A schedule $s_{v,\Gamma}$ and its pre-route π_{pre}.

Output: The maximum mean probability of on-time arrival *best_prob* and the optimal route π^*.

```
1: $R_s \leftarrow GetRequestsInSchedule(s_{v,\Gamma})$;
2: $best\_prob \leftarrow 0$;
3: $\pi^* \leftarrow \emptyset$;
4: while $(r \leftarrow R_s.pop()) \neq \emptyset$ do
5:     $n_1, s^1_{v,\Gamma}, n_2, s^2_{v,\Gamma} \leftarrow CutSchedule(d_r, s_{v,\Gamma})$;
6:     $prob_1, \pi_1 \leftarrow Find\alpha OptimalRoute(s^1_{v,\Gamma}, \pi_{pre})$;
7:     $prob_2, \pi_2 \leftarrow SRE(s^2_{v,\Gamma}, \pi_1)$;
8:     $mean\_prob \leftarrow (prob_1 \cdot n_1 + prob_2 \cdot n_2)/(n_1 + n_2)$;
9:     if $mean\_prob > best\_prob$ then
10:        $best\_prob \leftarrow mean\_prob$;
11:        $\pi^* \leftarrow \pi_1 + \pi_2$;
```

node, we will have $n_1 = 2$, $s^1_{v,\Gamma} = \{1, 2\}$, $n_2 = n_p - 2$ and $s^2_{v,\Gamma} = \{2, \ldots, n_p\}$.

Assuming the input schedule of Algorithm 5 has n_d drop-off visiting locations. Line 4-5 in Algorithm 5 takes $O(n_d)$ times to compute the α-shortest path and the on-time arrival probability for each drop-off visiting location. As $|A|$ is precomputed and independent from the length of schedule, the time complexity of Algorithm 5 is $O(n_d)$.

Assuming the input schedule of Algorithm 6 has n_p passengers, i.e., n_p drop-off visiting locations, Algorithm 6 produces a recursion tree, as shown in Figure 4.4. Considering the

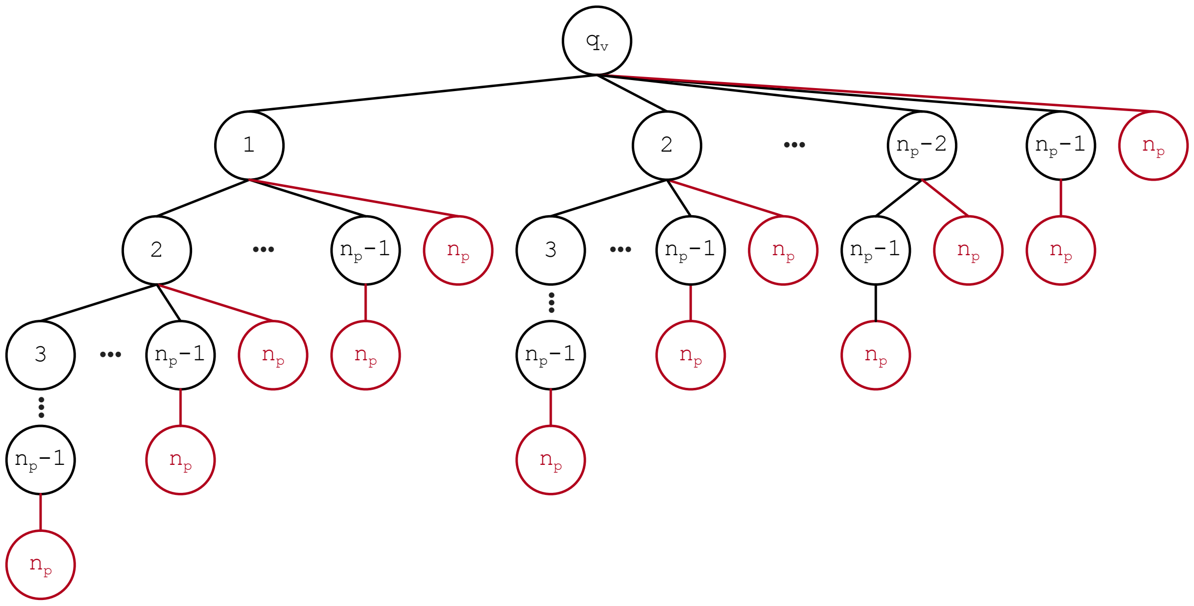


Figure 4.4: Illustration of the search tree of Algorithm 6, where n_p denotes the n_p^{th} drop-off location and each link denotes a running of function *FindαOptimalRoute*, i.e., Algorithm 5. Each path from the root node q_v to a leaf node n_p represents a possible combination of different α values. Red links represent the added computation caused by the n_p^{th} location, compared to when there are only $n_p - 1$ locations in the schedule.

complexity of Algorithm 5, we can tell that the recurrence relation in Algorithm 6 is:

$$\begin{aligned}
&T(n_p)\\
&= T(n_p-1) + [n_p-(n_p-1)] * 2^{n_p-2} + [n_p-(n_p-2)] * 2^{n_p-3} + \cdots + (n_p-1) * 2^0 + n_p\\
&= T(n_p-1) + 2^{n_p-1} * [1/2 + 2/2^2 + 3/2^3 + \cdots + (n_p-1)/2^{n_p-1}] + n_p\\
&= T(n_p-1) + 2^{n_p-1} * [2-(n_p+1)/2^{n_p-1}] + n_p\\
&= T(n_p-1) + 2^{n_p} - 1 \qquad (4.9)
\end{aligned}$$

Thus, the time complexity of Algorithm 6 is $O(2^{n_p})$. As Algorithm 6 explores all possible combinations of different α values for the drop-off locations in a schedule to find the best route, it has a high time complexity. Despite this, Algorithm 6 places only a small computational burden on the dispatcher. Note that $n_p = 6$ is considered a very large number in real-time ride-sharing. Having too many passengers sharing the same trip may drastically decrease user experience. The experimental study in Section 4.5 shows that the computation time of the stochastic scoring procedure is less than 3 sec.

Given a set of candidate schedules $\mathcal{F}_v$ for a vehicle, which is generated in step (b) in Figure 4.2, it can be processed in parallel on-the-fly by Algorithm 6 to find the stochastic optimal routes with the highest mean on-time arrival probabilities for scoring.

4.3.2 Reliable Allocating

At each round, the dispatcher considers the state of the fleet V, a set of requests R, and a set of scored schedules $\mathcal{F}_v$ for each vehicle. The goal is to serve as many requests as possible and dispatch the vehicles towards the most reliable trips so that the overall probability of on-time arrival is maximized. The allocating of requests is done by selecting a set of candidate sechedules, illustrated in step (d) in Figure 4.2, and is formulated as the following ILP:

$$\underset{x_{v,s},\epsilon_r}{\text{argmax}} \quad \sum_{v\in V}\sum_{s\in\mathcal{F}_v} x_{v,s}\cdot best\_prob(s) - \sum_{r\in R}\epsilon_r\cdot p_{miss} \tag{4.10}$$

$$s.t. \quad \sum_{s\in\mathcal{F}_v} x_{v,s} = 1, \quad \forall v\in V \tag{4.11}$$

$$\sum_{v\in V}\sum_{s\in\mathcal{F}_v} x_{v,s}\cdot\Theta_s(r) + \epsilon_r = 1, \quad \forall r\in R \tag{4.12}$$

where $best\_prob(s) = \max_\pi mean\_prob(s)$.

A binary variable $x_{v,s}\in\{0,1\}$ is introduced for each schedule in every $\mathcal{F}_v$, where $x_{v,s}=1$ indicates that s is assigned to v. A binary variable $\epsilon_r\in\{0,1\}$ is introduced for each request, where $\epsilon_r=1$ indicates that r is ignored. An indicator function $\Theta_s(r)\in\{0,1\}$ is introduced to indicate whether $r\notin s$ or $r\in s$. Constraint (4.11) guarantees that each vehicle is assigned exactly one schedule and constraint (4.12) guarantees that no requests are double-assigned to two vehicles. After allocating, each vehicle travels via the maximum reliability route π^* to serve the requests.

4.4 Profit Optimization

In practice, profit is one of the main concerns of a platform and passengers may accept longer travel times if there is compensation. In this section, we introduce the concepts of request prices, travel costs and late arrival penalties to the multi-phase scheme to optimize the profit of dispatch. Similar to RVD, PVD handles the scoring by finding the route with the maximum expected profit and solves an ILP for allocation.

We consider that passengers will receive compensation for being late under the travel delay constraint Λ, which could be caused by detour and travel time uncertainty. The objective of PVD is to maximize the service rate and total profit, defined as:

$$O_{Profit} = \sum_{v \in V} best\_profit(s_v) - \sum_{r \in R_{miss}} p_{miss} \tag{4.13}$$

where $best\_profit(s_v)$ is the maximum expected profit for a vehicle-trip match and is affected by the routing policy (discussed in Section 4.4.1). p_{miss} is a large rejection cost. Equation (4.13) works on profit optimization by finding the best matching policy that serves the largest number of passengers with the highest profit, while taking the compensation for late arrivals into account.

Problem 3 (Profit-aware dispatch) *Given a set of new requests R and a set of vehicles V at a dispatch epoch with a length of ΔT, the problem of profit-aware dispatch is to assign vehicles particular candidate schedules to serve as many as requests as possible with the highest expected profit, so that the objective (i.e., Equation (4.13)) is maximized, subject to the following constraints: (1) Each request must be served by exactly one vehicle. (2) Each schedule must be feasible when considering the mean travel time.*

4.4.1 Profit Estimation

Maximizing the platform's profit requires assigning high profit schedules to vehicles. The profit of a schedule is equal to the price sum of included requests minus the expenses of vehicle travel and the compensation due to late arrivals, defined as:

$$profit(s_{v,\Gamma}) = \sum_{r \in \Gamma} [rev(r) - penalty(r)] - cost(s_{v,\Gamma}) \tag{4.14}$$

where $rev(r)$ is the revenue of serving r, $penalty(r)$ is the compensation for dropping r off late, and $cost(s_{v,\Gamma})$ is the expense of the service. They are estimated as:

$$rev(r) = \beta \cdot \varrho(o_r, d_r) \tag{4.15}$$

$$penalty(r) = \gamma \cdot (ES(r) - \tau(o_r, d_r) - \Lambda) \tag{4.16}$$

$$cost(s_{v,\Gamma}) = \eta \cdot \varrho(\pi) \tag{4.17}$$

where β is the charge per unit distance, γ is the compensation per unit time, η is the expense per unit distance, $ES(r)$ is the expected travel time for r when violating the travel delay constraint, and $\varrho(\pi)$ is the travel distance of route π.

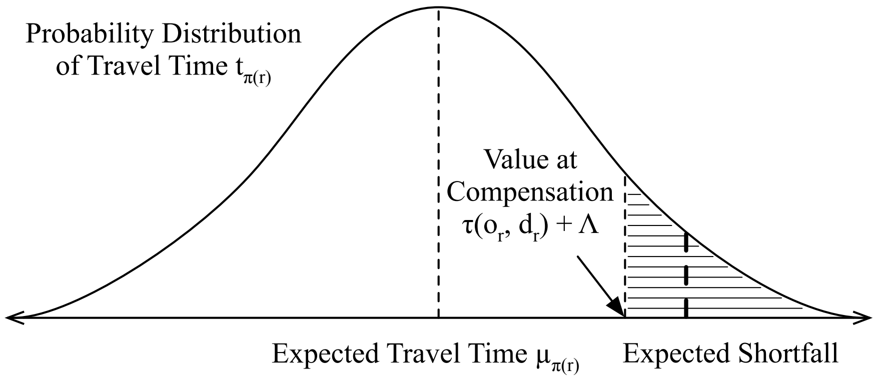

Probability Distribution of Travel Time $t_{\pi(r)}$
Value at Compensation $\tau(o_r, d_r) + \Lambda$
Expected Travel Time $\mu_{\pi(r)}$
Expected Shortfall

Figure 4.5: Graphical illustration of expected shortfall, the conditional expectation of a request's travel time given that the travel time is beyond the compensation level.

The term $ES(r)$ (expected shortfall of the request) is a risk measure proposed by Artzner et al. [121]. As shown in Figure 4.5, we have the probability distribution of the travel time of a request r, which follows a normal distribution $N(\mu_{\pi(r)}, \sigma^2_{\pi(r)})$. Given a maximum travel delay constraint Λ, the value at compensation is $\tau(o_r, d_r) + \Lambda$, meaning that penalties will be applied for travel time larger than this threshold. Using the formulation for expected shortfall of the Normal distribution in [122], the term $ES(r)$ can be computed as:

$$ES(r) = E[t_{\pi(r)} | t_{\pi(r)} > (\tau(o_r, d_r) + \Lambda)] \tag{4.18}$$

$$= \mu_{\pi(r)} + \frac{\sigma_{\pi(r)} \cdot \varphi(\frac{\tau(o_r, d_r) + \Lambda - \mu_{\pi(r)}}{\sigma_{\pi(r)}})}{1 - \Phi(\frac{\tau(o_r, d_r) + \Lambda - \mu_{\pi(r)}}{\sigma_{\pi(r)}})} \tag{4.19}$$

where $\varphi(\cdot)$ is the probability density function of the standard normal distribution.

As $rev(r)$ is normally an upfront fare and fixed regardless of the routing plan, and $\tau(o_r, d_r)$ and Λ are not affected by the routing plan, maximizing $profit(s_{v,\Gamma})$ is equivalent to solve:

$$\pi^* = \underset{\pi}{\operatorname{argmin}} \sum_{r \in \Gamma} \gamma \cdot ES(r) + \eta \cdot \mu_\pi \tag{4.20}$$

Equation (4.20) can be solved by enumerating the combinations of α-shortest paths, for all $\alpha \in A$, similar to Algorithm 6. Each schedule $s \in \digamma_v$ will find its optimal route that brings the maximum profit and will be scored with that profit by solving Equation (4.20).

4.4.2 Profit-Aware Allocating

The goal of allocating is to serve as many requests as possible and dispatch the vehicles towards the most profitable trips so that the overall profit of the platform is optimized. Similar to RVD, with a profit scored candidate schedule pool, the allocation policy in PVD is formulated as the solution to the following ILP:

$$\begin{aligned} \underset{x_{v,s},\epsilon_r}{\operatorname{argmax}} \quad & \sum_{v \in V} \sum_{s \in \mathcal{F}_v} x_{v,s} \cdot best\_profit(s) - \sum_{r \in R} \epsilon_r \cdot p_{miss} \\ s.t. \quad & \text{constraints (4.11) and (4.12)} \end{aligned} \tag{4.21}$$

where $best\_profit(s) = \max_\pi profit(s)$.

The objective (4.21) represents the sum of the expected profit each vehicle would earn and the total number of allocated requests. Constraints (4.11) and (4.12) guarantee the feasibility of the assignment, in which no conflict between requests and vehicles exists. After allocating, each vehicle will travel on the maximum profitable route π^* found in the process of profit estimation.

4.5 Experimental Study

In this section, the proposed two methods are implemented and evaluated using historical taxi request data from New York City [111]. They are compared to the state-of-the-art deterministic approach [19]. Since the implementation of [19] is unavailable, we reimplement it and run it on the same machine to ensure a fair comparison. Additionally, our two dispatchers employ the method in [19] to generate the candidate schedule pool. The reason we do not use the approach proposed in Chapter 3 for comparison is that, by the time we were preparing the content of this chapter as a paper, the submission containing Chapter 3 was still under review. But the trends would remain the same if we employed OSP as the candidate schedule pool generator. The performance measures include the service rate (i.e., percentage of requests served), the violation rate (i.e., percentage of late arrivals), the profit (i.e., Equation (4.14)), the request distance served (related to the revenue from passengers), the vehicle distance travelled (related to the expense of the platform), the vehicle mean load (i.e., average number of passengers in a vehicle), and the additional computation time yielded by considering the uncertainty of travel times (i.e., mean scoring time per dispatch epoch).

Experiments are designed to evaluate the following hypotheses: (H4.1) our proposed methods, reliability-aware and profit-aware dispatchers, can reduce the violations of delay constraints and improve the profit of the platform than deterministic dispatcher, respectively; (H4.2) the two methods can work well with different problem scales and levels of travel time stochasticity; and (H4.3) the two methods can be deployed in real time.

4.5.1 Simulation Details

This chapter follows the experimental environment in Chapter 3, except that the travel times are stochastic. The evaluations are conducted using request data from the 11th, 18th and 25th of May 2016. These have similar characteristics and are synthesized into three simulation scenarios of varying number of requests: 400k, 600k and 800k. Simulations are run for the hour with peak demand (19:00-20:00). A stochastic model of Manhattan is computed using the method in [7]. The complete road network contains 4,091 nodes and 9,452 directed edges. The travel times for each road segment for each hour of the day are computed using the pick up / drop off times of taxi trips. These are then processed to compute the daily mean and standard deviation for the travel time for each road segment.

Table 4.1 summarizes the major experimental control variables. Since the results when the capacity is equal to 6 are sufficient to illustrate the performance of the algorithm at high capacity ride-sharing, the default capacity of vehicles is set to 6 in the experiments in this chapter to save the running time of the simulation. When analysing the results of varying instance scales, it is designed to keep the service rates at the same level (i.e., the

Table 4.1: Parameter settings (defaults in bold).

Parameters	Settings
Fleet Size $\|V\|$	1000, **2000**, 3000
Max Waiting Time Ω (sec)	180, **300**, 420
Vehicle Capacity κ	2, 4, **6**, 8
Number of Requests $\|R\|$	**400k**, 600k, 800k

difference is less than 2%) and the fleet sizes chosen are a little different from Chapter 3, due to the travel time being stochastic in this chapter. We set the maximum travel delay to be twice the maximum wait time $\Lambda = 2\Omega$. The charge, compensation and expense parameters are set as $\beta = \$2/\text{km}$, $\gamma = \$0.02/\text{sec}$ and $\eta = \$1/\text{km}$. The dispatch epoch is set as $\Delta T = 30$ sec. The results are averages of ten experiment runs. All experiments were run on a machine with 56 Intel(R) Xeon(R) E5 2.60GHz processors and 512GB RAM in Python 3.7.

4.5.2 Algorithm Comparison

We examine the effectiveness of the following algorithms:

- DVD (Deterministic Vehicle Dispatch): A method that maximizes the service rate without considering travel time uncertainty, i.e., the one presented in [19].
- RVD (Reliability-aware Vehicle Dispatch): The method from Section 4.3 that considers the probability of on-time arrival for each request.
- PVD (Profit-aware vehicle Dispatch): The method from Section 4.4 that optimizes the profit of the platform.

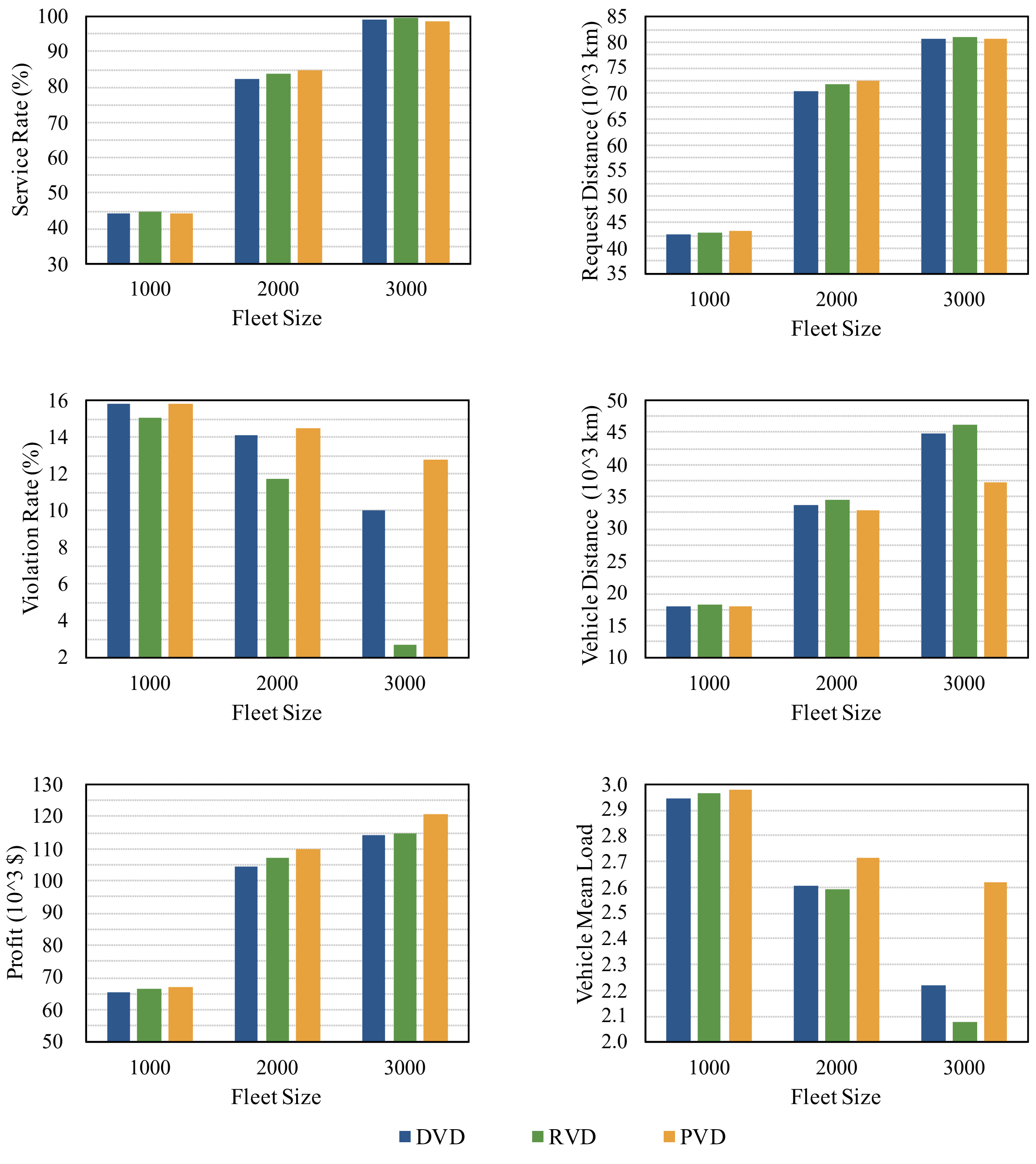


Figure 4.6: A comparison of performance metrics during the peak hour for varying fleet sizes ($|R| = 400k, \kappa = 6, \Omega = 300\,\text{sec}$).

4.5.3 Results

(H4.1) Figure 4.6 shows the results of varying fleet sizes. The performance of all the algorithms improves for larger fleet sizes. A larger fleet brings more seats and allows the dispatcher to allocate more mid-size ride-sharing trips to reduce detours for passengers, which reduces the violation rate. By dispatching vehicles to less uncertain trips and routes, RVD achieves lower violation rates than DVD. It also has higher service rates and produces greater profits than DVD. RVD achieves a 0.77% lower violation rate than DVD when there are 1000 vehicles, and with more flexibility brought in by 3000 vehicles, this reduction is increased to 7.30%. To achieve this significant reduction in the violation rate compared to DVD, RVD tends to dispatch small-size ride-sharing trips, having a lower vehicle mean load and has a longer vehicle distance travelled. To make more profit, PVD accepts late arrivals, despite yielding high violation rates, and prefers allocating large-size ride-sharing trips to save on vehicle travel costs. It achieves a 2.09% higher profit than DVD with a fleet of 1000 vehicles. Large fleet size also increases the improvement in profit brought in by PVD. It makes 6.08% more profit than DVD when the fleet size is 3000. The vehicle mean load of PVD is 2.62 when there are 3000 vehicle, this high occupancy rate yields a considerable saving in expense, as vehicle distance travelled is much lower than the other two algorithms.

(H4.1) When the number of vehicles is sufficiently large, both RVD and PVD can yield more improvements in reliability and profit, respectively, at the cost of profit and reliability, respectively. As shown in Figure 4.6, when the number of vehicles is increased from 2000 to 3000, the improvement of RVD relative to DVD in terms of profit decreases from 2.40% to 0.75%, and the deterioration of PVD relative to DVD in terms of violation rate widens from -0.44% to -2.82%. This is also validated in the results of simulations conducted between 4:00-5:00, when the number of requests is only 6.85% of the peak hour. As shown in Table 4.2, the service rates of all algorithms are very close to 100% due to a sufficient

Table 4.2: A comparison of performance metrics for different methods during the hour with lowest number of requests. ($|R| = 400k,\ |V| = 2000,\ \kappa = 6,\ \Omega = 300\,\text{sec}$)

Metrics	DVD	RVD	PVD
Service Rate (%)	99.57	99.81	99.43
Violation Rate (%)	4.29	0.27	8.26
Profit ($)	7,671.04	7,085.13	9,262.84
Request Distance (km)	7,375.03	7,378.08	7,359.96
Vehicle Distance (km)	7,009.79	7,665.62	5,304.61
Vehicle Mean Load	1.21	1.03	1.62

number of vehicles. RVD achieves a close to zero violation rate, but has a 7.64% lower profit than DVD. Similarly, PVD makes a 20.75% higher profit than DVD, with a 3.97% higher violation rate.

(H4.1) Figure 4.7 shows the results of varying values of the maximum waiting time constraint. A longer travel delay constraint allows more detours and increases the occupancy rate. Vehicles could serve more requests while reducing miles traveled, thus leading to higher service rates and greater profits. The violation rates do not significantly change with waiting time constraints, except that RVD yields a 0.84% increase when the maximum waiting time constraint increases from 180 sec to 300 sec. When the constraint increases from 180 sec to 420 sec, although the reduction in violation rate yielded by RVD over DVD has a small decrease from 2.9% to 2.3%, the increase in service rate increases from 1.19% to 2.21%. This change is in line with our expectations, as our primary optimization goal is the service rate. If we simply seek to minimize the violation rate, the dispatcher will choose not to serve a single passenger to keep the violation rate at zero. PVD utilizes the increase in detour tolerance well and achieves higher improvements over DVD in both the service rate (from 1.23% to 4.22%) and the profit (from 3.43% to 6.01%), when the maximum waiting time

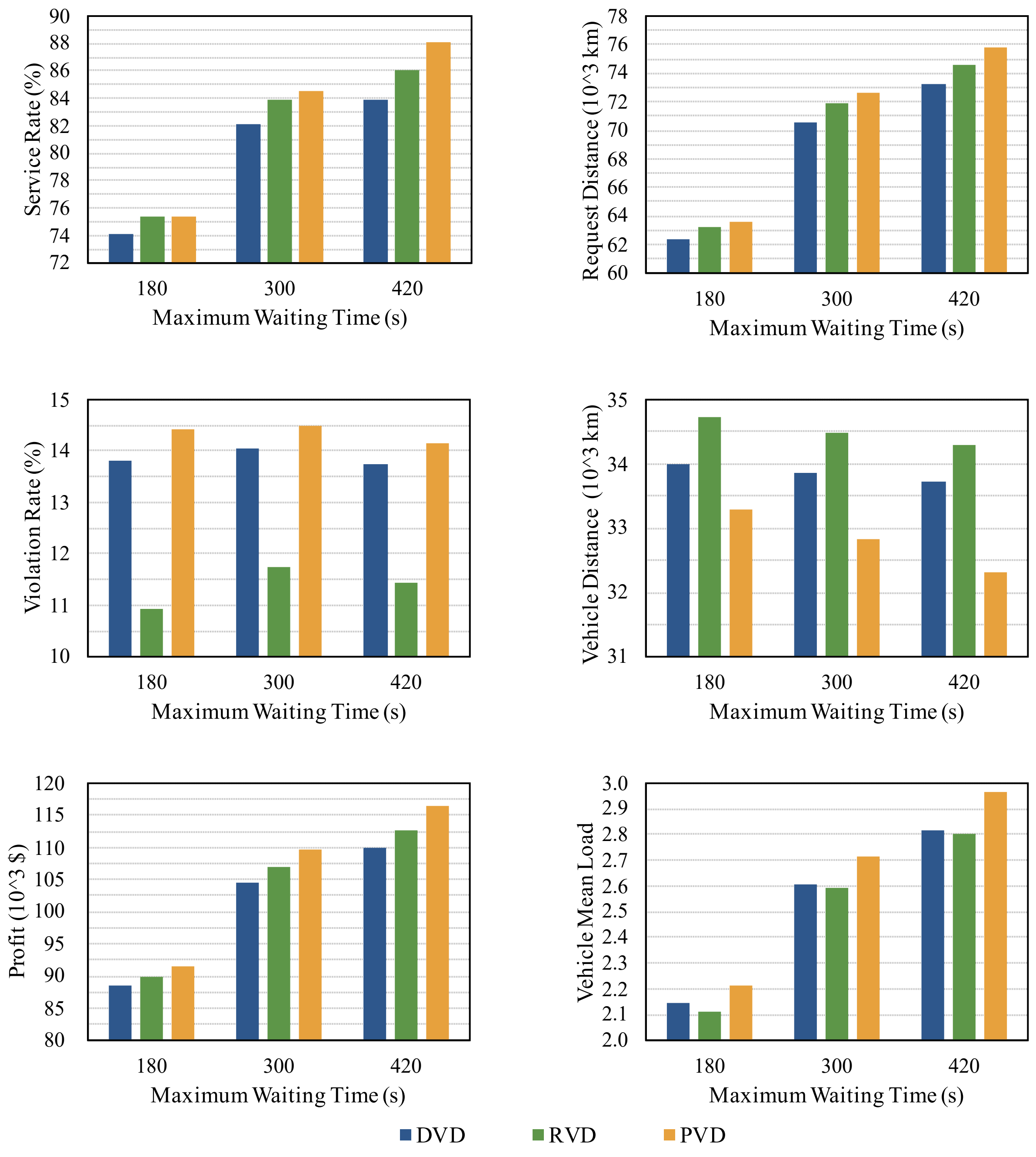


Figure 4.7: A comparison of performance metrics during the peak hour for varying values of maximum waiting time constraint ($|R| = 400k,\ |V| = 2000,\ \kappa = 6$).

constraint increases from 180 sec to 420 sec. This is because service rates and profits are usually positively correlated.

(H4.1) Figure 4.8 shows the results of varying vehicle capacities. All algorithms achieve higher service rates and higher profits with higher capacity, as more seats allow more requests to share their trips. The violation rate also increases as capacity rises, possibly due to detours caused by more ride-sharing. The performance differences among DVD, RVD and PVD are not significantly affected by differences in vehicle capacity, especially when it comes to high-capacity (i.e., $\kappa \geq 4$) ride-sharing, where the reductions in violation rates brought in by RVD over DVD are around 2.30($\pm$ 0.05)% and the increases in profit brought in by PVD over DVD are around 4.89($\pm$ 0.05)%. When the capacity of vehicles is 2, PVD achieves a 4.43% higher profit than DVD, but uncommonly has a 0.85% lower violation rate and a 1.21% lower service rate. This is because the expense savings of low-capacity ride-sharing are low and some requests might not be profitable considering the late arrival compensation, thus PVD chooses to ignore them to maximize the profit. We also find that PVD yields a 1.22% higher request distance than DVD when capacity is 2, which indicates that PVD prefers long trips.

(H4.2) Figure 4.9 shows the results of varying instance scales. When the number of requests increases, the fleet size is also increased to examine the scalability of the algorithms. The reductions in violation rates brought in by RVD over DVD are 2.30%, 1.98% and 2.13%, respectively. The improvements in profit produced by PVD over DVD are 4.87%, 5.69% and 6.40%, respectively. This indicates that a ride-sharing service can benefit from taking into account the stochastic travel time, regardless of the instance scale. Moreover, as the size of the problem grows, RVD and DVD achieve even more improvements. In terms of the service rate, when the number of requests is doubled from 400k to 800k, the gain of RVD over DVD increases from 1.77% to 2.61% and that of PVD increases from 2.41% to 4.19%. This indicates that the information of travel time uncertainty is more valuable when the

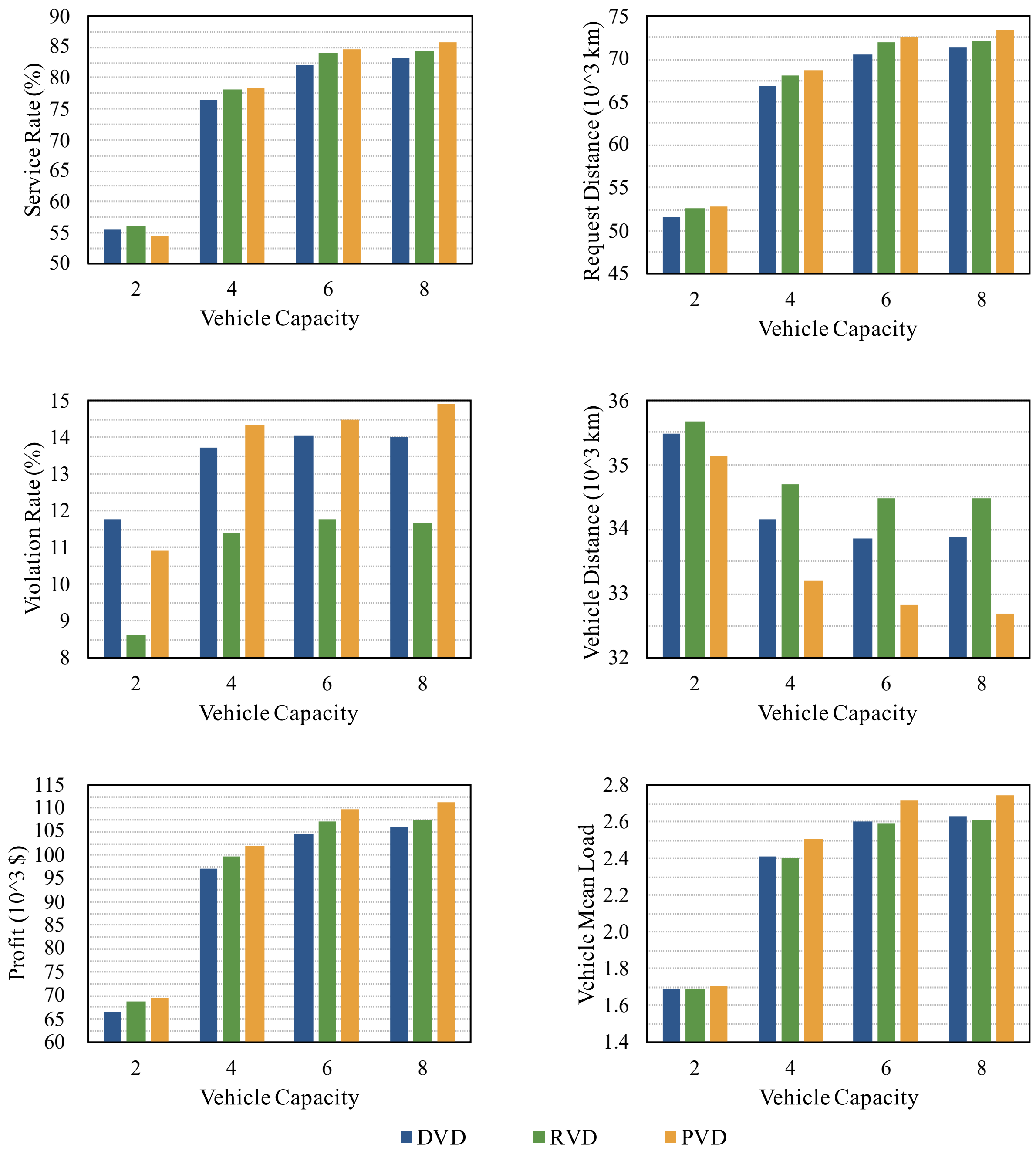


Figure 4.8: A comparison of performance metrics during the peak hour for varying vehicle capacities ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec}$).

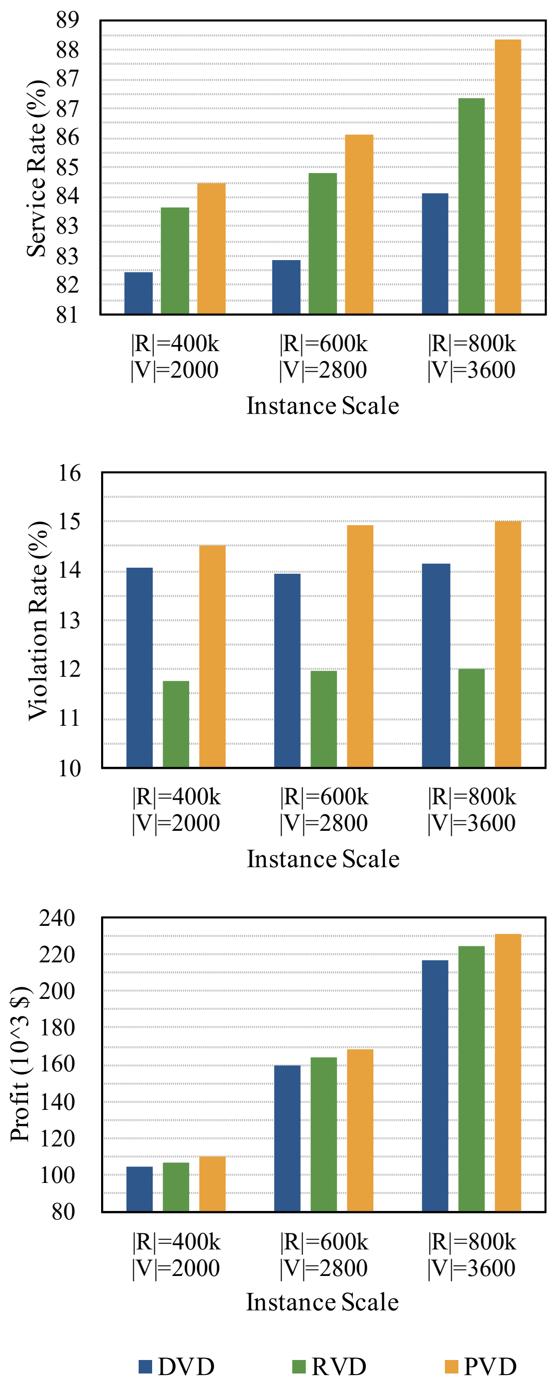


Figure 4.9: A comparison of performance metrics during the peak hour for varying instance scales ($\kappa = 6,\ \Omega = 300\,\text{sec}$).

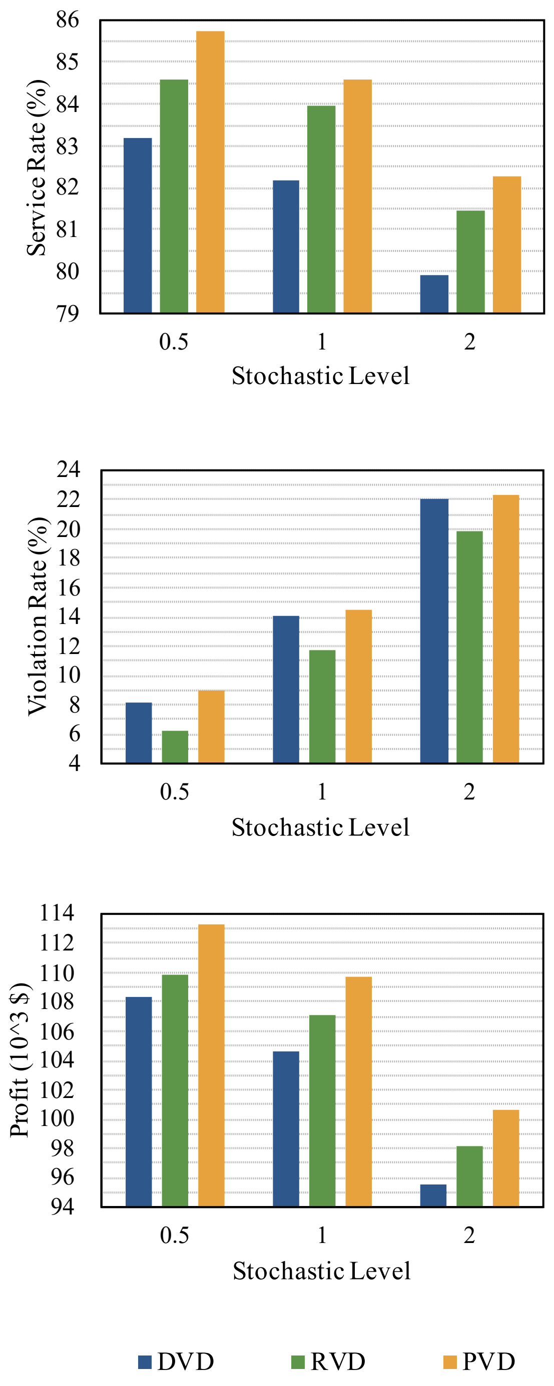


Figure 4.10: A comparison of performance metrics during the peak hour for varying stochastic levels ($|R| = 400k,\ |V| = 2000,\ \kappa = 6,\ \Omega = 300\,\text{sec}$).

density of passengers is higher, as ride-sharing is more likely to happen.

(H4.2) Figure 4.10 shows the results of varying stochastic levels. We analyze the robustness of algorithms with different variances, i.e., $\{0.5, 1, 2\} \cdot \sigma_e^2$. Although all algorithms have worse performance with higher stochastic levels, the improvements yielded by RVD and PVD over DVD are stable, e.g., {1.42%, 1.77%, 1.56%} (RVD) and {2.56%, 2.41%, 2.39%} (PVD) regarding the service rate. This suggests that the proposed approaches are robust to different stochastic travel time models.

Table 4.3: A comparison of performance metrics during the peak hour for varying penalties for PVD ($|R| = 400k,\ |V| = 2000,\ \kappa = 6,\ \Omega = 300\,\text{sec}$).

Metrics	Penalty γ (\$/sec)				
	0.02	0.04	0.06	0.08	0.10
Service Rate (%)	84.58	84.66	85.34	85.50	86.10
Violation Rate (%)	14.50	14.25	14.02	13.15	12.96
Profit (10^3 $)	109.67	107.06	105.19	103.20	101.50
Relative Profit Difference With Respect to DVD (%)	4.87	5.07	6.03	6.90	8.13

We also investigate the effect of changing the penalty value considered by PVD. The results are shown in Table 4.3. It is not a surprise to find that profit is dropping when we raise the penalty for late arrivals, considering the growth in the compensation paid to passengers. But raising the penalty improves all other metrics. PVD even yields a lower violation rate than DVD when the penalty is larger than $0.06/sec. In practice, passengers would prefer a lower delay constraint, under which PVD yields fewer improvements, as shown in Figure 4.7. Large compensation for long detours could potentially increase passengers' tolerance to delays and eventually increase the profit of the platform. For example, if raising the penalty from $0.02/sec to $0.08/sec could increase the maximum waiting time constraint from 180 sec to 300 sec, we might have an approximate 10.10% (i.e., 85.50% minus 75.40%) higher service rate, an approximate 1.27% (i.e., 14.42% minus 13.15%) lower violation rate and an approximate \$11.72k (i.e., \$103.20k minus \$91.48k) higher profit.

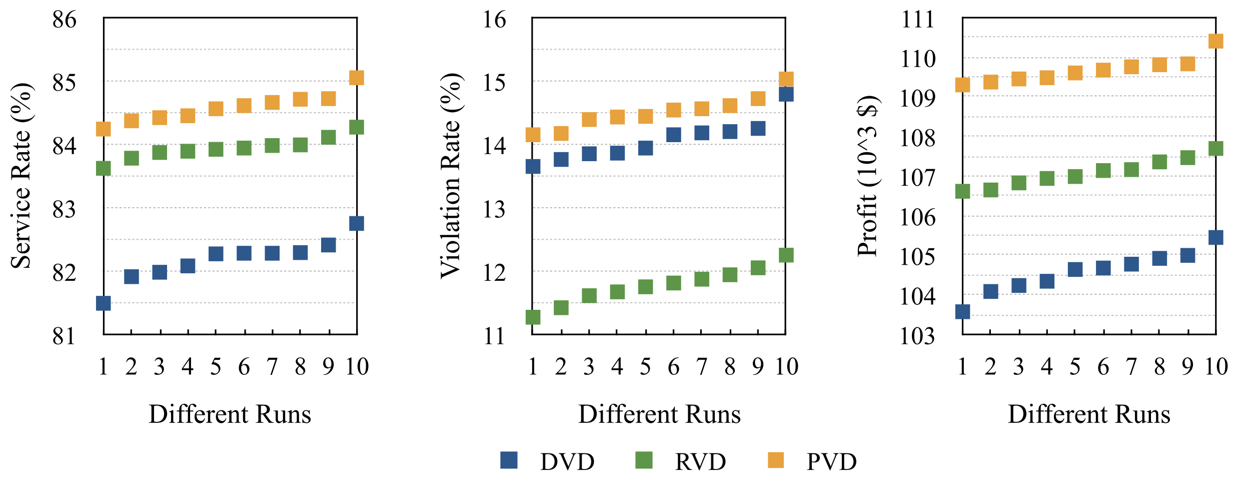


Figure 4.11: A comparison of the results of DVD, RVD and PVD under 10 simulation runs ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

The statistical significance of the results above is tested by using a large sample size, where the number of requests counted in experiments with the default parameter set is around 22,631. Statistical measures of the simulation results of service rates, violation rates and profits are shown in Tables 4.4, 4.5 and 4.6, respectively. It can be seen that the mean and median of the results are very close and the standard deviation is small, indicating that the improvements yielded by RVD and PVD over DVD are stable and are not due to random chance. Moreover, even when using the worst results of RVD and PVD to compare the best results of DVD, improvements still exist. We also investigate the distribution of results from multiple experimental runs, as shown in Figure 4.11, where the results are plotted in ascending order. It is demonstrated that both RVD and PVD have stable improvements versus DVD. The low standard deviation of the results is because of the large sample size. Table 4.7 shows the standard deviation for varying the number of counted requests. By varying the number of vehicles, we can see that the standard deviation of the violation rate

Table 4.4: Statistical measures of service rates (%) for different algorithms ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Algorithms	Mean	Minimum	Maximum	Median	Standard Deviation
DVD	82.17	81.49	82.75	82.28	0.34
RVD	83.94	83.62	84.27	83.93	0.18
PVD	84.58	84.24	85.05	84.59	0.23

Table 4.5: Statistical measures of violation rates ($) for different algorithms ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Algorithms	Mean	Minimum	Maximum	Median	Standard Deviation
DVD	14.06	13.65	14.79	14.05	0.33
RVD	11.76	11.27	12.25	11.78	0.29
PVD	14.50	14.15	15.03	14.49	0.26

Table 4.6: Statistical measures of profits (10^3 $) for different algorithms ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Algorithms	Mean	Minimum	Maximum	Median	Standard Deviation
DVD	104.57	103.57	105.45	104.66	0.53
RVD	107.08	106.61	107.69	107.06	0.35
PVD	109.67	109.30	110.40	109.64	0.32

Table 4.7: A comparison of standard deviation of violation rates (RVD) for varying numbers of vehicles ($|R| = 400k,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Number of Vehicles	1	10	100	500	1000	2000
Number of Served Requests	14	122	1,240	6,144	12,127	22,631
Standard Deviation of Violation Rate	17.36	3.88	1.42	0.73	0.32	0.29

decreases as the number of requests counted rises. These suggests that the improvement discussed based on the mean of the experimental results is statistically significant.

We further investigate the effect of changing the method to generate the candidate schedule pool from RTV (i.e., the method in [19]) to OSP. Tables 4.8, 4.9 and 4.10 show the comparisons of service rates, violation rates and profits for varying capacities, respectively, where "-OSP" indicates that the candidate schedule pool is generated by OSP. In line with the results in Chapter 3, all dispatchers perform better after switching to OSP, and the improvements grow with capacity. We also see enhancements in regards to the improvements yielded by RVD and PVD over DVD. For example, the reduction in violation rate brought in by RVD-OSP over DVD-OSP is 2.92% and the increase in profit brought in by PVD-OSP over DVD-OSP is 5.05%, when the capacity is 6, while those two numbers are 2.30% and 4.87% before the switching to OSP. However, these enhancements are much smaller than the differences between RTV and OSP. Moreover, the trends of the results are the same, whether the candidate schedule pool is generated by RTV and OSP. Thus, the improvements yielded by RVD and PVD over DVD do not rely on the choice of candidate generation methods.

(H4.3) Finally, we investigate the additional computation time caused by the introduction of the stochastic scoring step. Table 4.11 shows the computation times of the scoring step for varying capacities. DVD-OSP only needs to retrieve the already calculated travel delays from the candidate schedule pool, so it takes very little time. Although RVD-OSP and PVD-OSP take 10 times more computation time for scoring, the time cost is less than 3 sec. Therefore, RVD-OSP and PVD-OSP are able to dispatch vehicles in real-time, i.e., the total computation time is less than 30 sec while the dispatch window is 30 sec.

Table 4.8: A comparison of service rate (%) during the peak hour for varying vehicle capacities ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Vehicle Capacity	Algorithms					
	DVD	DVD-OSP	RVD	RVD-OSP	PVD	PVD-OSP
2	55.53	55.81	56.25	56.59	54.32	54.62
4	76.38	76.92	78.07	78.77	78.47	79.05
6	82.17	83.46	83.94	85.29	84.58	86.01
8	83.19	84.95	84.43	86.35	85.70	87.57

Table 4.9: A comparison of violation rate (%) during the peak hour for varying vehicle capacities ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Vehicle Capacity	Algorithms					
	DVD	DVD-OSP	RVD	RVD-OSP	PVD	PVD-OSP
2	11.77	11.42	8.65	8.02	10.92	10.35
4	13.71	13.11	11.37	9.99	14.34	13.96
6	14.06	13.07	11.76	10.15	14.50	13.72
8	14.01	13.08	11.70	10.06	14.91	14.08

Table 4.10: A comparison of profit (10^3 \$) during the peak hour for varying vehicle capacities ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Vehicle Capacity	Algorithms					
	DVD	DVD-OSP	RVD	RVD-OSP	PVD	PVD-OSP
2	66.37	66.48	68.56	68.68	69.31	69.57
4	97.02	97.41	99.57	100.21	101.80	102.42
6	104.57	105.90	107.08	108.79	109.67	111.25
8	106.08	107.67	107.71	109.63	111.26	113.30

Table 4.11: Computation time (sec) of different procedures during the peak hour for varying vehicle capacities ($|R| = 400k,\ |V| = 2000,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Procedures	Vehicle Capacity			
	2	4	6	8
Scoring Process in DVD-OSP	0.09	0.13	0.15	0.15
Scoring Process in RVD-OSP	1.03	1.75	1.97	2.13
Scoring Process in PVD-OSP	1.08	1.84	2.06	2.23

4.5.4 Discussion

Taking the uncertainty of travel time into consideration, our proposed approaches, RVD and PVD, are able to assign vehicles to the most reliable trips and most profitable trips, so that the performance of vehicle dispatch is significantly improved. Along with the drop in the violation rate and the growth in profit, there is also an increase in the service rate. Both RVD and PVD achieve greater gains over DVD for larger fleet sizes (especially when the number of vehicles is sufficient to handle almost all requests) and looser delay constraints. Additionally, the performances of RVD and PVD scale well across different vehicle capacities and instance sizes. Although our assumption that edge travel times follow independent Gaussian distributions may limit the accuracy of our model, experiments with different levels of travel time uncertainty show that our proposed multi-phase scheme is robust to different stochastic travel time models and can be easily coupled with other stochastic routing and estimating methods. For example, if a taxi trajectory dataset is available, the learning method proposed by Li et al. [123] to estimate the travel time distribution could be employed to enhance the accuracy of our model.

4.6 Chapter Summary

In this chapter, we have presented a travel time uncertainty aware vehicle dispatch scheme for on-demand ride-sharing, a method to allocate the most reliable trips and a method to optimize the profit of vehicle dispatch. Numerical simulations on Manhattan taxi datasets show that, by considering stochastic travel times, the proposed methods improve upon the state-of-the-art deterministic dispatcher in terms of the reliability (up to 7.3% at peak hour), the profit (up to 8.13% at peak hour) and the service rate (up to 4.22% at peak hour).

Chapter Five

Learning-Based Dispatch for Long-Term Optimization

To ensure real-time response to passengers, existing solutions to the vehicle dispatch problem typically optimize dispatch policies with small batch windows and ignore the temporal dynamics over the long-term horizon. In this chapter, we focus on improving the long-term performance of Shared Autonomous Mobility-on-Demand (SAMoD) systems and propose a deep reinforcement learning based method for the ride-sharing dispatch problem. In particular, this work includes: (1) an Offline Policy Evaluation (OPE) based method to learn a value function that indicates the expected reward of a vehicle reaching a particular state; (2) an online learning procedure to update the value function to capture the real-time dynamics during the operation; (3) an efficient online dispatch method that optimizes the matching policy by considering both past and future influences. Extensive simulations are conducted based on the New York City taxi data, and show that the proposed method further increases the service rate compared to the method presented in Chapter 3. Additionally, it outperforms the state-of-the-art far-sighted ride-sharing dispatch approach.

5.1 Introduction

In SAMoD systems, a key goal is to assign and route vehicles to serve as many requests as possible. The most common approach is batch assignment, which uses dispatch windows to match multiple requests at the same time and is well suited for real-world ride-sharing production systems. Recent research has enabled "optimal" optimization in each dispatch window, but is myopic in nature as the window length has to be small to provide a good passenger experience, and leaves room for further optimizations [23, 32]. Figure 5.1 shows an example where the optimal policy is assigning the blue schedule to the vehicle at present, although the red one has a more immediate reward, and adjusting the schedule according to the appearance of newly revealed requests to get a maximum total reward.

Meanwhile, the rapid development of online ride hailing services (e.g., Uber and DiDi) provides rich information on taxi mobility patterns, which has lead to research on the far-sighted vehicle dispatch problem [64, 88, 99]. To leverage the transit information to learn urban mobility patterns and improve transportation efficiency, the following technical challenges need to be addressed:

1. *Learning Efficiency.* Applying extensive learning exploration on real-world deployments is inefficient and may lead to unintended behaviour that leads to a bad user experience. Training in simulations requires an accurate simulator that is typically hard to build and the transfer of learned behaviour from simulation to reality is also a challenge. Thus, there is a desire to use historical vehicle trajectories from the real-world for efficient learning.

2. *Real-Time Fluctuations.* In the real-world, besides the systematic patterns of urban mobility (e.g., people are more likely to travel to residential areas during the evening peak hours), there are also the non-stationarity dynamics (e.g., people may go to

different restaurants on different days). A dispatcher that can continues to capture real-time dynamics over time is more preferable to deploy.

3. *Value Deviation.* The learned mobility pattern is an approximation of the values of real-world dynamics and bias exists. Thus, if a dispatcher can re-optimize its matching policy, it is expected that some deviations will be corrected and eventually a higher service rate is achieved.

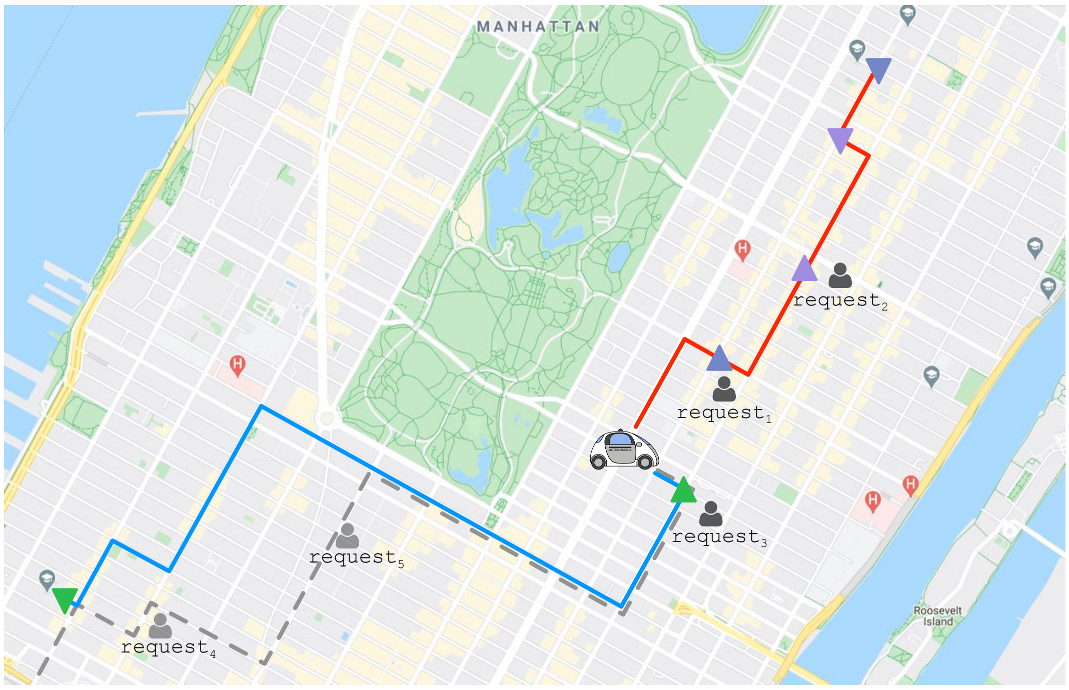


Figure 5.1: An example with one vehicle, three revealed requests and two requests that will arrive in the future but are not known at present. Triangles ($\triangle$) and inverted triangles ($\bigtriangledown$) represent the origins and destinations, respectively. When myopically optimizing over the current epoch, the vehicle will go towards the top right of the map to serve requests 1 and 2 and earn a reward of 2. However, if it chooses to serve request 3 and instead travels towards the bottom left of the map, it will eventually earn a reward of 3.

Most existing work on the far-sighted vehicle dispatch problem does not allow ride-sharing [64, 88, 99]. Some work studies double-occupancy fleets, but they formulate assignment decisions with no more than five candidate options [100, 102], which cannot be directly extended to higher occupancy ride-sharing. More relevant to this chapter is the Approximate Dynamic Programming (ADP) method proposed in [32], where a neural network based value function is learned through techniques from Deep Q-Network (DQN) [97]. However, the training process heavily relies on the use of simulators and real-time dynamics during operations are not considered.

In this chapter, we study the long-term optimization of the vehicle dispatch problem in SAMoD systems and present a learning-based method that is flexible in adjusting the dispatch policy based on real-time dynamics to maximize the service rate (i.e., percentage of requests served). The method adopts large-scale Offline Policy Evaluation (OPE) and fast online learning to capture the spatial-temporal dynamics of ride-sharing trips. It also employs the re-optimization strategy proposed in Chapter 3 to mitigate the deviation between the learned value and the actual value. To summarize the work in this chapter, we:

1. Develop an Offline Policy Evaluation (OPE) based method to capture the systematic patterns of ride-sharing mobility, using a neural network to learn a spatio-temporal value function from historical vehicle movement data. (Section 5.3.1.)

2. Adopt an online learning procedure to continuously update the value function to handle non-stationarity dynamics, using only the states of vehicles in the current operating epoch. (Section 5.3.2.)

3. Combine the learned value function with the re-optimization dispatch method in Chapter 3 to optimize the objective over a long horizon, including both recent and upcoming epochs. (Section 5.3.3.)

4. Conduct simulation experiments with taxi trips from Manhattan to evaluate our proposed approach and analyze the impact of long-term optimization. (Section 5.4.)

The remainder of the chapter is arranged as follows. In Section 5.2, we model the movements of a vehicle as a Markov Decision Process (MDP), introduce the problem statement and give an overview of the value-based dispatch framework. In Section 5.3, we present the technical details of the proposed dispatch method. In Section 5.4, we evaluate the proposed approach using real world taxi data. In Section 5.5, we draw our conclusions.

5.2 Preliminaries

In the vehicle dispatch problem, the rolling horizon framework is naturally a sequential decision-making process, as the dispatcher needs to compute matching decisions in consecutive time slots and aims to maximize the cumulative number of served requests. This suggests modelling the problem as a Markov Decision Process (MDP), especially considering that the goal in RL (i.e., maximizing the cumulative reward) is consistent with the objective in the dispatch problem. RL has great promise for enhancing the performance of a dispatcher, because it can not only learn value functions from historical trajectories to guide matching decision making, but can also have the potential to use policy iteration to improve the dispatcher over time. In addition, RL can complete the main training procedure offline and has no impact on the complexity of the candidate match search procedure for the dispatch problem, so it has very little impact on the computational efficiency of online dispatching, as shown in the experimental study later. Note that the candidate search procedure is the main computational burden of the dispatch problem.

Similar to the approaches presented in the previous two chapters, the dispatcher proposed in this chapter solves a static problem repeatedly in a rolling-horizon framework, i.e., batch planning every ΔT. We also follow the notions of SAMoD systems defined in Chapter 3. However, to be consistent with the notation in the Reinforcement Learning

(RL) community, some symbols used in this chapter may have a different meaning than in Chapters 3 and 4, e.g., V stands for the value function and no longer refers to the vehicle fleet, R stands for the reward function, s stands for the state, l stands for the schedule, α stands for the learning rate and γ stands for the discount factor.

5.2.1 Definitions

Following the definitions in Section 3.2 of Chapter 3, a dispatcher operates a fleet of m vehicles. At each planning epoch, it receives a set of n new requests and computes all possible vehicle-trip matches to produce the optimal assignment. A trip Γ denotes that a group of requests are merged with ride-sharing. The order in which a vehicle picks up and drops off requests in a trip is called a schedule $l = \{o_1, \dots, d_1, \dots, o_2, \dots, d_{n_\Gamma}\}$, where q is the current location of the vehicle, o_i and d_i are the origins and destinations of requests. A schedule must satisfy the capacity (κ), waiting time (Ω), total delay (Λ) and precedence constraints. In this chapter, we assume that all possible vehicle-trip matches are known and focus on the allocation problem, i.e., finding the maximum matching policy. We also assume that each vehicle-trip match comes with a schedule that has the lowest expected travel delays for passengers. In addition, the set of candidate vehicle-trip matches for an individual vehicle is denoted by a schedule pool $\mathcal{F} = \{l_1, l_2, \dots\}$.

We model the vehicle dispatch problem as an MDP, with an agent representing an individual vehicle. In this framework, a vehicle interacts with an environment over a sequence of discrete time steps $t \in \{0, 1, 2, \dots, n_{end}\}$, where n_{end} is the terminal time (e.g., end of an operating hour or day). At each time step t, the vehicle receives a state s_t, based on which it selects an action a_t. It then receives a scalar reward r_{t+1} and transitions to s_{t+1}, according to environmental dynamics. The goal of a vehicle is to maximize the expected discounted return $G_t = \sum_t^{n_{end}} \gamma \cdot r_{t+1}$ (i.e., the long-term total reward from time step t), where $0 < \gamma \leq 1$

is the discount rate. The specific definitions of the ride-sharing dispatch problem are given as follows:

State. The state of a vehicle contains the spatial status l_t, the associated temporal status ζ_t, the real world time stamp u_t and the exogenous information w_t (i.e., supply-demand contextual features). Formally, it is defined as $s_t = (l_t, \zeta_t, u_t, w_t)$. The temporal status ζ_t represents the remaining delay time when the vehicle visits each location in schedule l_t, reflecting how much more deviation from the current path the vehicle can take to pick up a new passenger. For example, assuming a vehicle is traveling to location d_1 to drop off a passenger, the deadline to arrive is 400 sec and the travel time is 200 sec, then the vehicle has 200 sec (i.e., the remaining delay time) to make a detour to pick up a new passenger. Passengers that require more than 200 sec to pick up will not be feasible for the vehicle. The time stamp u_t indicates the time scale in the entire operating horizon of the SAMoD system and is independent of the algorithmic time t. The exogenous information w_t consists of two scalars: the number of new requests at the current dispatch epoch and the number of nearby vehicles at the vehicle's current location.

Action. The eligible actions for a vehicle include all possible candidate schedules $\mathcal{F}_{i,t}$ (i stands for the vehicle id), where each one indicates a candidate vehicle-trip match that assigns the requests included in the trip to the vehicle. Formally, it is defined as $a_t = l_{t+1}$. Note that a vehicle has $|\mathcal{F}_{i,t}|$ actions. There are basically two types of actions. The first type is to add (or remove) one or more requests to (from) the vehicle and replace its schedule with the action schedule. The other type is "remaining the same" which leaves the vehicle to follow its current schedule. Executing action a_t means replacing the schedule of the vehicle with l'_{t+1} and transitioning to l_{t+1}, where l'_{t+1} is the status of l_{t+1} at time t.

Reward function. The reward is the number of requests added to the vehicle when it takes an action a_t. Formally, it is defined as $R(s_t, a_t) = (|l'_{t+1}| - |l_t|)/2$. The output value of $R(s_t, a_t)$ is denoted by r_{t+1}. The "remaining the same" action would produce a reward of 0 (i.e., $r_{t+1} = 0$). Note that r_{t+1} could be negative if requests that are assigned to the vehicle at $t-1$ are re-assigned to other vehicles at t.

State transition. When a vehicle is in state s_t and takes an action a_t, its transition to state s_{t+1} is denoted by $s_{t+1} = f(s_t, a_t, w_{t+1})$. However, in SAMoD systems, the decisions have to be made at t before the realization of the exogenous information w_{t+1} (arriving between t and $t+1$), so as not to affect the passenger experience. Therefore, the post-decision state s_t^a [124] (also called afterstate [125]) is introduced to split the transition function into two components: $s_t^a = f^{(1)}(s_t, a_t)$ and $s_{t+1} = f^{(2)}(s_t^a, w_{t+1})$. The post-decision state is a segregation of deterministic and stochastic information about the future. It explicitly captures the most recent system state without actually moving forward in time [126].

In the above MDP model, the interactions between a vehicle and the environment would produce a historical trajectory $(s_0, a_0, r_1, s_0^a, w_1, s_1, a_1, r_2, s_1^a, \ldots, s_t, a_t, r_{t+1}, s_t^a, \ldots)$. To solve an MDP, a common way is to find a value function that measures potential future rewards to guide the interaction of the agent. As the full dynamics of an SAMoD system are not known to the dispatcher when deciding the matching between vehicles and requests at time step t, we choose to learn a post-state value function that incorporates the most recent information we have. This would produce a more efficient learning method than learning an action-value function [125, 127].

5.2.2 Problem Statement

Considering the long-term impact of vehicle-trip matches, we allow vehicles to forgo the maximization of immediate reward in order to maximize their service rate over the entire operating horizon. We define the objective of value-based far-sighted dispatch as a sum over the action-value of each vehicle (denoting the vehicle id by i):

$$O_{value} = \sum_{i=1}^{m} Q(s_i, a_i) \tag{5.1}$$

where $Q(s_i, a_i)$ is the expected cumulative future rewards, i.e., the expected number of requests that a vehicle may serve considering its state s_i and its action a_i. $Q(s_i, a_i)$ is derived from a learned post-state value function, which will be discussed in the next section.

Problem 4 (Far-sighted dispatch) *Given a set of requests and a set of vehicles at a dispatch epoch with a length of ΔT, the problem of far-sighted dispatch is to assign vehicles particular schedules to maximize the service rate over a long time horizon, so that the objective (i.e., Equation (5.1)) is maximized, subject to the constraint that each request must be served by exactly one vehicle.*

5.2.3 System Framework

Figure 5.2 illustrates our proposed framework for solving Problem 4. We build on prior work on ride-hailing [64, 88, 99], which generalizes offline policy iteration and on-policy value iteration [125] to vehicle dispatch, but with a model strictly limited to single-occupancy vehicles. The framework, which chooses to optimize the objective of ride-sharing over a long horizon while maintaining an efficient online matching procedure that yields a short response time, consists of three components: Online Dispatch, Online Learning, and Offline Policy Evaluation (OPE). The online dispatch part is similar to the multi-phase scheme discussed in Section 4.2.3 of Chapter 4, where the scoring step is now processed by using a value function to get the expected number of served requests for each candidate match, instead of the on-time arrival probability. The two learning parts, online learning and offline evaluation, are working backends to support the online dispatch part and getting data from it to jointly train a value function.

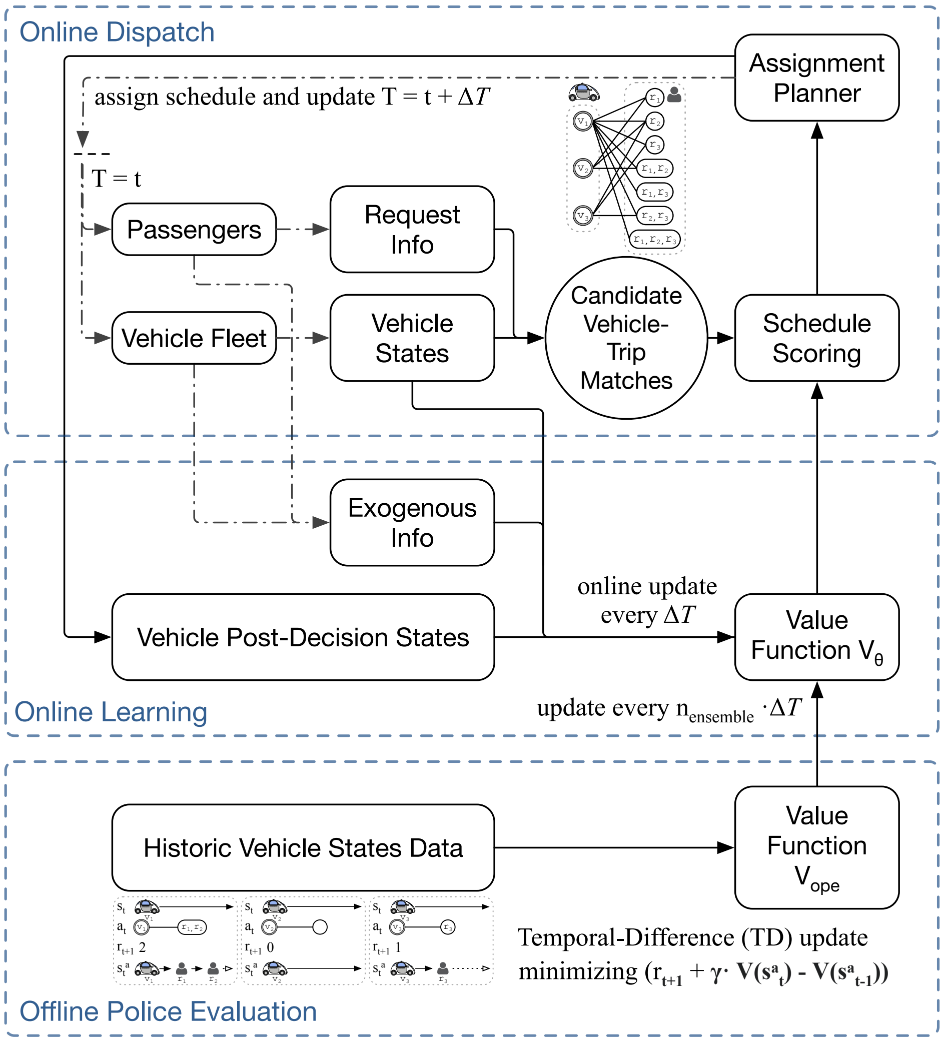

Online Dispatch
Assignment Planner
assign schedule and update T = t + ΔT
T = t
Passengers
Request Info
Vehicle Fleet
Vehicle States
Candidate Vehicle-Trip Matches
Schedule Scoring
Exogenous Info
online update every ΔT
Vehicle Post-Decision States
Value Function Vθ
Online Learning
update every n_ensemble ·ΔT
Historic Vehicle States Data
Value Function V_ope
Temporal-Difference (TD) update
minimizing (r_t+1 + γ· V(s^a_t) - V(s^a_t-1))
Offline Police Evaluation

Figure 5.2: Framework of the long-term optimization logic.

OPE. Given a set of historical state transitions collected from different vehicles, denoted by $\{(s^a_{i,t-1}, r_{i,t+1}, s^a_{i,t})\}$ (i stands for vehicle id), the dispatcher performs a policy evaluation with a neural network to learn a post-decision state value function $V_{ope}(s^a)$, by applying the Bellman squared error and Temporal Difference (TD) update.

Online Dispatch. With the learned value function, the dispatcher is able to match vehicles and requests in a far-sighted way, by scoring candidate matches (i.e., candidate schedules) with the sum of instant rewards and long-term expectations. The dispatcher also considers re-optimization by adjusting the matching between vehicles and previously received requests with the most recent knowledge about the system, so as to extend the matching horizon for a more "global" optimization and correct the bias of the value function.

Online Learning. During the online dispatch procedure, a new set of state transitions is generated at each epoch, which reflects the real-time dynamics of the current operational state. The dispatch leverages these real-time transitions to continuously update the value function, so that it can quickly adapt to any fluctuations in the system.

5.3 Value-based Vehicle Dispatch Scheme

We assume the dispatcher is equipped with a matching policy that always maximizes the objective function at each dispatch epoch (i.e., maximizing Equation (5.1) at time step t) and remains unchanged over the training period. When the objective function has no knowledge about the future (i.e., $Q(s,a) = R(s,a)$), the dispatcher is called a myopic one. Given the MDP formulation and historical trajectories generated by the myopic dispatcher, we first use a neural network to approximate a post-state value function $V_{ope}(s^a)$ for the underlying policy. We then present an online learning procedure that adds an adaptive capability to the value network. Finally, with the learned value function in place, we discuss how to employ it in the scoring and allocating steps of online ride-sharing dispatch, so as to turn the myopic dispatcher into a far-sighted one.

The value function is trained for an individual vehicle, under the assumption that all vehicles are homogeneous and have a common goal of maximizing the global service rate of the SAMoD system. Learning a shared value function, not a separate network for each vehicle, utilizes the historical data in a more efficient way and requires a simpler training pipeline [64, 89]. The matching step during dispatch maximizes Equation 5.1 in a coordinated way to achieve global optimization.

5.3.1 Offline Policy Evaluation With Neural Networks

As we do not distinguish individual vehicles, a complete historical transition tuple can be presented as $(s^a_{t-1}, w_t, s_t, a_t, r_{t+1}, s^a_t, w_{t+1}, s_{t+1})$. If $t = 0$, we set $s^a_{t-1} = s_0$. Let $V(s)$ denote the state-value function of a vehicle, the Bellman equation can be written as:

$$\begin{aligned} V(s_t) &= \mathbb{E}\{G_t | s = s_t\} \\ &= \mathbb{E}\{r_{t+1} + \gamma \cdot G_{t+1} | s = s_t\} \\ &= \mathbb{E}\{r_{t+1} + \gamma \cdot V(s_{t+1})\} \end{aligned} \tag{5.2}$$

Considering the transition from s_t to s_{t+1} and applying the Temporal Difference (TD) learning method [125], the state-value function is updated as:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma \cdot V(s_{t+1}) - V(s_t)] \tag{5.3}$$

where $\alpha > 0$ is the learning rate, s_t is the current state of the vehicle and s_{t+1} is the next state following the schedule indicated by a_t.

Using the post-decision state and let $V_{ope}(s^a)$ denote the post-state value function learned by OPE, Equation (5.2) is decomposed into two steps:

$$V(s_t) = \mathbb{E}\{r_{t+1} + \gamma \cdot V_{ope}(s^a_t)\} \tag{5.4}$$

$$V_{ope}(s^a_t) = \mathbb{E}\{V(s_{t+1}) | s^a_t\} \tag{5.5}$$

The advantage of this decomposition is that the right hand side of Equation (5.4) is deterministic, which makes it easier to use deterministic algorithms [128]. Similar to Equation (5.3), the post-decision state value function is updated as:

$$V_{ope}(s^a_{t-1}) \leftarrow V_{ope}(s^a_t) + \alpha[r_{t+1} + \gamma \cdot V_{ope}(s^a_t) - V_{ope}(s^a_{t-1})] \tag{5.6}$$

which is almost the same as Equation (5.3), except that we are now updating the value function around the previous post-decision state.

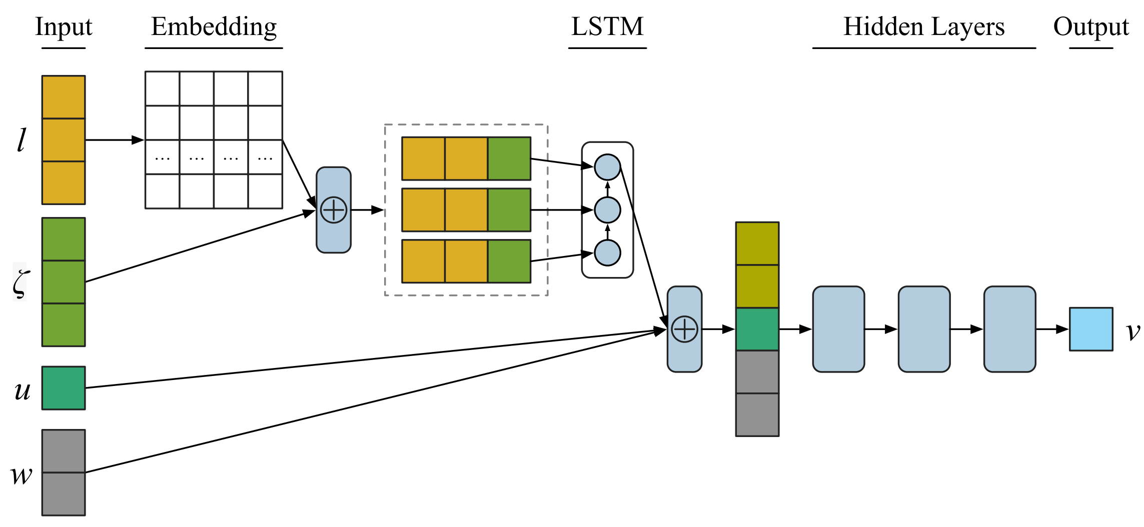


Figure 5.3: Structure of the value network.

In SAMoD systems, both the state space and action space grow exponentially, especially with regard to the number of state variables. To overcome the curse of dimensionality, a neural network is used to approximate the post-decision state value function. Its structure is depicted in Figure 5.3. The network takes a post-decision state $s_t^a = f^{(1)}(s_t, a_t) = (l_{t+1}, \zeta_{t+1}, u_{t+1}, w_t)$ as input and outputs a scalar as the estimation of the value. It uses an embedding layer to learn the geographical feature, a Long Short-Term Memory (LSTM) layer to learn the sequence-level temporal feature, and a hidden layer to combine the spatial-temporal features with the exogenous information.

We collect a set of historical transition tuples and store them in a replay buffer $D_{offline}$. Each tuple $(s_{t-1}^a, r_{t+1}, s_t^a)$ represents that a vehicle transitions from s_{t-1}^a to s_t^a within one time step and receives a reward of r_{t+1}. We use Double Q-Learning [101] and TD error to train the network $V_{ope}(s^a)$ offline with the following objective:

$$\underset{\theta_{ope}}{\operatorname{argmin}}\ \mathcal{L} = \mathbb{E}_{(s_{t-1}^a, r_{t+1}, s_t^a) \sim D_{offline}}[(r_{t+1} + \gamma \cdot \hat{V}_{ope}(s_t^a) - V_{ope}(s_{t-1}^a))^2] \tag{5.7}$$

where θ_{ope} is the weights of $V_{ope}(s^a)$, and $\hat{V}_{ope}(s^a_t)$ is the target network that is designed to reduce overestimation [97].

Algorithm 7 Offline Policy Evaluation (OPE) with Neural Network

Input : A prioritized replay buffer filled with historical state transitions $D_{offline} = \{(s^a_{t-1}, r_{t+1}, s^a_t)\}$, the number of iterations n_{iter}, the discount factor γ and the target smoothing factor ρ.

Output: The post-decision state value function $V_{ope}(s^a)$.

```
1: $V_{ope}, \theta_{ope} \leftarrow InitializeValueNetwork();$
2: $\hat{V}_{ope}, \hat{\theta}_{ope} \leftarrow InitializeValueNetwork();$
3: for $i \leftarrow 1$ to $n_{iter}$ do
4:     $B \leftarrow SampleMiniBatch(D_{offline});$
5:     $\mathcal{L} = 1/|B| \cdot \sum_{j=1}^{|B|}(r_{j,t+1} + \gamma \cdot \hat{V}_{ope}(s^a_{j,t}) - V_{ope}(s^a_{j,t-1}))^2;$
6:     $\theta_{ope} \leftarrow GradientDescent(\mathcal{L}, \theta_{ope});$
7:     $\hat{\theta}_{ope} \leftarrow \rho \cdot \theta_{ope} + (1 - \rho) \cdot \hat{\theta}_{ope};$
```

The main procedure for offline training is presented in Algorithm 7. The algorithm uses prioritized experience replay [129] to make the training more efficient by sampling important transitions more frequently, where the importance of a transition is measured by the magnitude of its TD error. It also uses mini-batch gradient descent to train $V_{ope}(s^a)$. Function *InitializeValueNetwork* returns a network using the structure in Figure 5.3 with random weights. Function *SampleMiniBatch* returns a small set of transitions using prioritized sampling, which is then used to compute the mean-squared loss $\mathcal{L}$ and perform a gradient descent step to update the network weights θ_{ope} (line 6). In the last line of the algorithm, a soft update [130] is applied to the target network weights $\hat{\theta}_{ope}$. Soft update tries to improve the stability of learning by changing the target network very slowly.

The resultant value function $V_{ope}(s^a)$ captures the general systematic patterns of the supply-demand conditions from historical data, and will serve as the basis for the online value function during operation (i.e., online dispatching).

5.3.2 Online Learning With Value Ensemble

To capture the non-stationary dynamics in real-time, we present a new value network $V_{ol}(s^a)$ that uses the transitions of vehicles in the current dispatching epoch. The main difference between $V_{ol}(s^a)$ and $V_{ope}(s^a)$ is that the online value function $V_{ol}(s^a)$ cares more about the spatial dynamics of the supply-demand conditions, and fixes time at t_{net} to utilize the sparse online transition data more efficiently.

Considering that at time step t, after the matching policy has been made, we will have an online state transition $(s_{i,t}, a_{i,t}, r_{i,t+1}, s^a_{i,t})$ for each vehicle $i \in [1, 2, \dots, m]$. We allow the dispatcher to make a backup of the transitions at time step $t-1$, and use it to generate an online buffer $D_{online} = \{(s^a_{i,t-1}, r_{i,t+1}, s^a_{i,t})\}$. Note that $|D_{online}| = m$. The one-step TD update for each vehicle's transition is slightly different from Equation 5.6, given by:

$$V_{ol}(\Psi(s^a_{i,t-1}, t_{net})) \leftarrow V_{ol}(\Psi(s^a_{i,t}, t_{net})) + \alpha[r_{t+1} + \gamma \cdot V_{ol}(\Psi(s^a_{i,t}, t_{net})) - V_{ol}(\Psi(s^a_{i,t-1}, t_{net}))] \tag{5.8}$$

where a new function $\Psi(\cdot)$ is introduced to fix the time of states. As an example, for $s^a_{t-1} = (l_t, \zeta_t, u_t, w_{t-1})$ and $s^a_t = (l_{t+1}, \zeta_{t+1}, u_{t+1}, w_t)$, we have $\Psi(s^a_{t-1}, t_{net}) = (l_t, \zeta_t, u_{t_{net}}, w_{t-1})$ and $\Psi(s^a_t, t_{net}) = (l_{t+1}, \zeta_{t+1}, u_{t_{net}}, w_t)$.

Using all transitions in D_{online}, Double Q-Learning and TD error to train the network $V_{ol}(s^a)$ online, the objective is given by:

$$\underset{\theta_{ol}}{\operatorname{argmin}}\ \mathcal{L} = \sum_{i=1}^{m} (r_{i,t+1} + \gamma \cdot \hat{V}_{ol}(\Psi(s^a_{i,t}, t_{net})) - V_{ol}(\Psi(s^a_{i,t-1}, t_{net})))^2 \tag{5.9}$$

where θ_{ol} is the weights of $V_{ol}(s^a)$, and $\hat{V}_{ol}(s^a_t)$ is the target network.

The terms t_{net} and $\Psi(\cdot)$ are introduced to build the capability of effective online learning from sparse data. All transitions occurring between t_{net} and $t_{net,next}$ are treated as data at t_{net}, which in turn allows more effective learning of the spatial dynamics of the supply-demand conditions, under the assumption that the temporal dynamics do not change significantly in a short period of time, e.g., 10 min. To give an example, supposing $t_{net} = 0$ sec and $t_{net,next} = 600$ sec, function $\Psi(\cdot)$ will modify the time parameter of the input to the neural network during this interval $[t_{net}, t_{net,next})$ to $u_t = 0$ sec. The downside of this is that it will make the update of the neural network less smooth, so the online trained network is periodically combined with the offline trained one, as shown in Algorithm 8.

Algorithm 8 Online Learning With Value Ensemble

```
Input : The offline learned value network $V_{ope}$ with its weights $\theta_{ope}$, the discount factor
    $\gamma$, the value ensemble step size $n_{ensemble}$, the value ensemble factor $\psi$ and the target
    smoothing factor $\rho$.
 1: $V_{ol}, \theta_{ol} \leftarrow InitializeValueNetwork();$
 2: $\hat{V}_{ol}, \hat{\theta}_{ol} \leftarrow InitializeValueNetwork();$
 3: $t_{net} \leftarrow 0;$
 4: for $t \leftarrow 0$ to $n_{end}$ do
 5:     if $t \bmod n_{ensemble} = 0$ then
 6:         $\theta_{ol} \leftarrow \psi \cdot \theta_{ope} + (1 - \psi) \cdot \theta_{ol};$
 7:         $t_{net} \leftarrow t;$
 8:     $D_{online} \leftarrow SolveMatchingProblem(V_{ol});$
 9:     $\mathcal{L} = 1/m \cdot \sum_{i=1}^{m} (r_{i,t+1} + \gamma \cdot \hat{V}_{ol}(\Psi(s^a_{i,t}, t_{net})) - V_{ol}(\Psi(s^a_{i,t-1}, t_{net})))^2;$
10:     $\theta_{ol} \leftarrow GradientDescent(\mathcal{L}, \theta_{ol});$
11:     $\hat{\theta}_{ol} \leftarrow \rho \cdot \theta_{ol} + (1 - \rho) \cdot \hat{\theta}_{ol};$
```

The main procedure for online training is presented in Algorithm 8, which is a joint procedure for online dispatching. The algorithm just needs a very simple memory buffer, as it only has a limited number of transitions and uses batch gradient descent to train $V_{ol}(s^a)$.

The algorithm initializes $V_{ol}(s^a)$ with random weights θ_{ol} at the beginning of operation and updates it in two processes: (1) periodically update its weights θ_{ol} using a weighted ensemble with a snapshot of the offline learned network $V_{ope}(s^a)$ (line 6); (2) update its weights θ_{ol} using real-time transitions in D_{online} at each dispatch epoch (line 8-10). The value ensemble factor $\psi > 0$ is a hyperparameter similar to the target smoothing factor ρ. It determines how much importance we place on historical and immediate experience, by adjusting the weighting between the offline trained network $V_{ope}(s^a)$ and the online learned network $V_{ol}(s^a)$ [99].

The resultant online value function $V_{ol}(s^a)$ is actually an augmentation of $V_{ope}(s^a)$, by continuously updating the value function using the most recent information to capture the real-time variations of the moment. It relies on the ensemble with $V_{ope}(s^a)$, as only partial vehicle trajectories are available for online training, which makes it hard to learn a global value function. Also, solely using online learning suffers from sample-inefficiency.

5.3.3 Re-Optimization and Value-Based Allocating

At each epoch, function *SolveMatchingProblem* in Algorithm 8 solves a maximum matching problem to determine the assignment policy. Considering a fleet of m vehicles, a set of previously reveived yet not served requests REQ_{prev}, a set of newly submitted requests REQ_{new}, and a set of candidate action pools $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_i\}$ that have been computed for each vehicle, the goal is to find the best action for each vehicle to maximize the long-term value (i.e., the service rate over a long horizon). The action pool $\mathcal{F}_i$ for each vehicle is generated by using Algorithm 3 and Algorithm 4 discussed in Chapter 3. It contains all possible actions the vehicle can take at the current dispatch epoch, meaning that the action with the maximum value must be included otherwise it is an infeasible one (i.e., violating one or more constraints on capacity, waiting time, total delay and precedence). This maximum matching problem can be formulated as a Integer Linear Program (ILP):

$$\operatorname*{argmax}_{x_{i,j},\epsilon_k} \quad \sum_{i=1}^{m}\sum_{j=1}^{|F_i|} x_{i,j} \cdot Q(s_i, a_j) \tag{5.10}$$

$$s.t. \quad \sum_{j=1}^{|F_i|} x_{i,j} = 1, \quad \forall i \in [1, 2, \ldots, m] \tag{5.11}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{|F_i|} x_{i,j} \cdot \Theta_{a_j}(k) + \epsilon_k = 1, \quad \forall k \in REQ_{prev} \cup REQ_{new} \tag{5.12}$$

$$\epsilon_k = 0, \quad \forall k \in REQ_{prev} \tag{5.13}$$

where $Q(s_i, a_j) = R(s_i, a_j) + \gamma \cdot V_{ol}(s_i^a)$, and $\Theta_{a_j}(k)$ is an indicator function (i.e., $\Theta_{a_j}(k) = 0$ if a_j does not serve k and $\Theta_{a_j}(k) = 1$ otherwise).

Binary variable $x_{i,j} \in \{0, 1\}$ indicates whether an action a_j (i.e., a candidate schedule) is selected ($x_{i,j} = 1$) or not ($x_{i,j} = 0$) for vehicle i. Binary variable ϵ_k indicates whether a request k is matched ($\epsilon_k = 0$) or not ($\epsilon_k = 1$). Constraint (5.11) ensures that each vehicle selects exactly one action, constraint (5.12) ensures that each request is matched to at most one vehicle. While assigning vehicles to new incoming requests, the dispatcher also considers adjusting the matches of previous requests based on the latest knowledge of the supply-demand conditions (including both revealed and estimated). At the same time, constraint (5.13) ensures the passenger experience, by guaranteeing that no previously matched requests will be rejected even if rejecting them yields a higher objective value at the current epoch.

The re-optimization procedure adjusts the previous matching, both by considering new requests and by updating the weights of actions with the latest value function $V_{ol}(s^a)$. When producing the matching policy at time step t, each candidate vehicle-trip match (i.e., the action) is scored with the expected future reward (i.e., $R(s_t, a_t) + \gamma \cdot V_{ol}(s_t^a)$) that might have potential prediction bias. Then, at time step $t + 1$, besides using TD error to update the value function, the dispatcher also implicitly uses a Bellman-style update of the form

$R(s_t, a_t) + R(s_{t+1}, a_{t+1}) + \gamma \cdot V_{ol}(s^a_{t+1})$, to adjust the score of each candidate match, to alter the matching policy. Moreover, the latest value function $V_{ol}(s^a)$ guides the dispatcher to alter the matching policy in a more far-sighted way. For example, a vehicle is previously dispatched to a working area where it is expected to receive requests from off-duty passengers, but if a large musical concert has just ended and creates a large number of requests, the vehicle will be re-directed to that area for a better future expectation.

5.4 Experimental Study

In this section, our proposed method is evaluated and analyzed using a real-world taxi dataset from New York City [111]. The performance of our method is compared to the state-of-the-art far-sighted dispatch approach and our previous approach introduced in Chapter 3. All algorithms are implemented and run on the same machine for a fair comparison. The performance measures include the service rate (i.e., percentage of requests served) and the additional computation time yielded by introducing RL methods (i.e., mean scoring time per dispatch epoch).

Experiments are designed to evaluate the following hypotheses: (H5.1) our Long-Term Optimization (LTO) dispatcher can outperform the state-of-the-art far-sighted dispatcher (i.e., [32]), by combining a long-term value function and re-optimization; (H5.2) updating the value function online to capture traffic fluctuations has the potential to further improve the service rate; and (H5.3) the additional computational time yielded by employing the value function does not affect the real-time deployment of the LTO dispatcher.

5.4.1 Simulation Details

The experimental environment and assumptions in this chapter are the same as in Chapter 3, where the travel times are deterministic, as our focus is on considering the distribution requests to improve the service rate. The experiments are conducted using data from two days: 25th and 26th May 2016. The value network $V_{ope}(s^a)$ is trained using data from nine days (3rd-5th, 10th-12th and 17th-19th May 2016) and validated on 24th May 2016. The selected days are Tuesday, Wednesday and Thursday and have similar request patterns [99]. The vehicle state transitions used for training are generated by running OSP, the dispatcher in Chapter 3. Simulations are run for the hour with peak demand (19:00-20:00), where the average number of requests is 18,789. The road map and travel times of Manhattan are produced using the method proposed in [7]. The simulation environment used in this chapter is the same as the one in Chapter 3.

Table 5.1: Parameter settings (defaults in bold).

Parameters	Settings
Vehicle Capacity κ	2, 4, **6**, 8
Fleet Size m	1200, **1500**, 1800
Maximum Waiting Time Ω (sec)	180, **300**, 420
Batch Period ΔT (sec)	10, **30**, 60, 120

The main experimental parameters are summarized in Table 5.1, where we vary the supply and demand conditions to evaluate the algorithms. Besides, the maximum travel delay is set to be twice the maximum wait time $\Lambda = 2\Omega$. In the previous Chapters 3 and 4, the evaluations are conducted on datasets synthesized from taxi data over three days to test the ability of handling large-scale scenarios (400k–800k requests) and the scalability of algorithms. In this chapter, to avoid the dependency between datasets affecting the training effect, the raw taxi trip data is used, which has an average number of daily requests of 308k.

So the fleet size here is different. In addition, the results of Chapters 3 and 4 show that whether the dispatcher can handle large scale problem scenarios basically depends only on the method of generating candidate schedule pool, so we are not testing the performance for varying instance scales here any more. The hyperparameters of the value network are tuned by running the validation simulation and then fixed when running the evaluation simulation. Specifically, we set the discount factor to $\gamma = 0.95$, the learning rate to $\alpha = 0.01$, the target smoothing factor to $\rho = 0.05$, the value ensemble step size to $n_{ensemble} = 600\ sec/\Delta T$ and the value ensemble factor to $\psi = 0.9$. All experiments were run on a machine with 56 Intel(R) Xeon(R) E5 2.60GHz processors and 512GB RAM in Python 3.9. The results are averaged over five experiment runs.

5.4.2 Algorithm Comparison

We examine and compare the following algorithms.

- LTO (Long-Term Optimization): The method proposed in this chapter that optimizes the objective for all available requests over a long horizon.
- LTO-Basic: A method that emulates the one presented in [32]. It only optimizes the objective for newly received requests, with a value function estimating the expected reward in the future. It is obtained by removing the online learning and re-optimization components of LTO.
- OSP (Optimal Schedule Pool): The method from Chapter 3 that maximizes the objective for all available requests, but does not have any knowledge about the future.
- Baseline: A method that only optimizes the current dispatch epoch by maximizing the objective for newly received requests. It is a simplified version of OSP that does not allow re-optimization.

The key difference between them is how many batch windows are optimized for the assignment policy, as shown in Figure 5.4. The Baseline algorithm is used to demonstrate the performance of the dispatcher without any advanced algorithms. In addition, the difference between the improvement of LTO over OSP and the improvement of LTO-Basic over Baseline can help demonstrate the role of re-optimization and online learning.

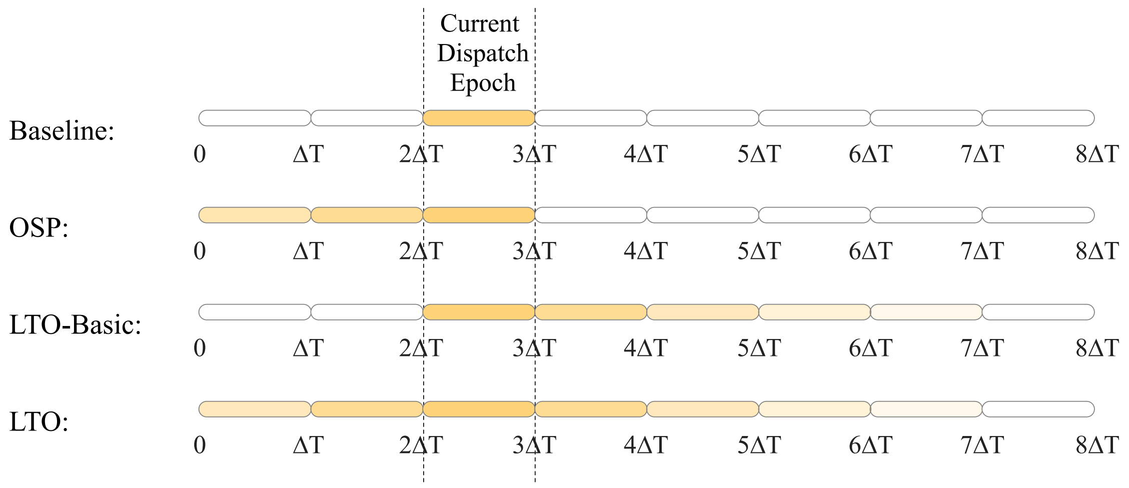


Figure 5.4: A schematic comparison of how many dispatch epochs are considered for optimization by different algorithms. The epochs considered are marked in light orange, and the intensity of the color indicates how much the information within that epoch contributes to the optimization.

5.4.3 Results

(H5.1) Figure 5.5 shows the results of varying vehicle capacities on two different days. The mobility patterns of the two days are similar and all the algorithms have similar performance. With higher vehicle capacity, the SAMoD system would have more seats and thereby serve more requests. But the new increased seats are bundled with existing trips, rather than working independently and flexibly to transport passengers, and require the dispatcher to capture a more complex mobility pattern. When the capacity is 2 and 4, the average service

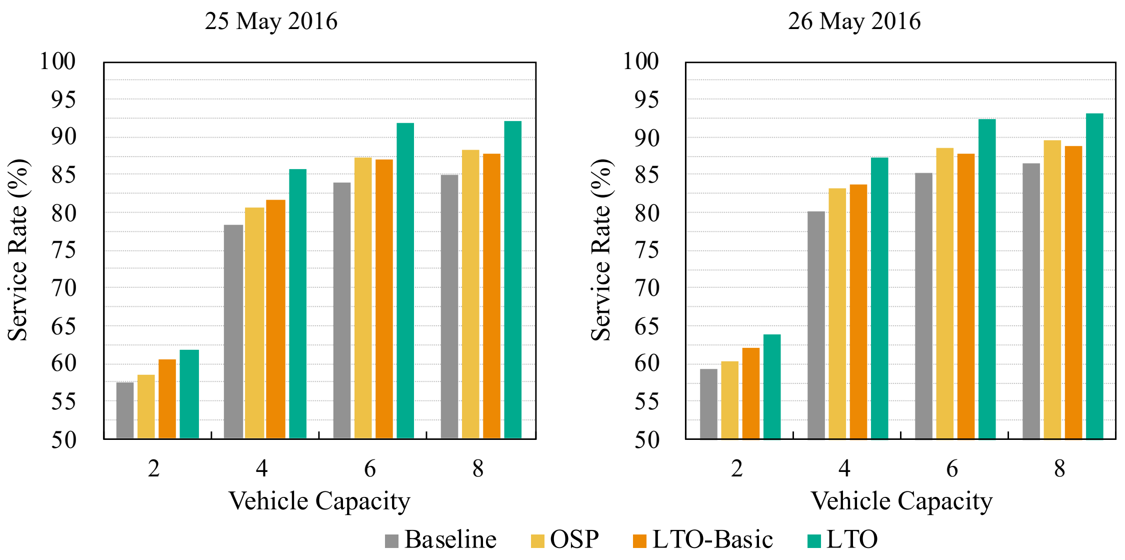


Figure 5.5: A comparison of service rate (%) during the peak hour for varying vehicle capacities ($m = 1500,\ \Omega = 300\,\mathrm{sec},\ \Delta T = 30\,\mathrm{sec}$).

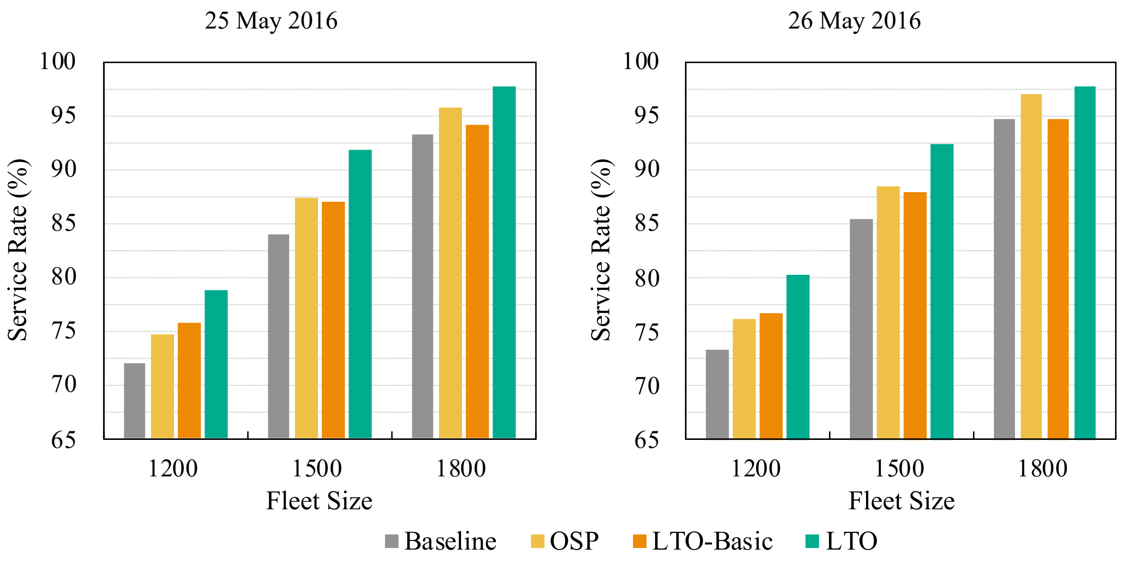


Figure 5.6: A comparison of service rate (%) during the peak hour for varying fleet sizes ($\kappa = 6,\ \Omega = 300\,\mathrm{sec},\ \Delta T = 30\,\mathrm{sec}$).

rate of LTO-Basic is 1.82% and 0.76% higher than that of OSP; but when the capacity is 6 and 8, the average service rate of LTO-Basic is 0.45% and 0.66% lower than that of OSP, respectively. This indicates the importance of making accurate estimates of the future, and the robustness of the re-optimization strategy that always explicitly uses all precisely revealed information. Our proposed method LTO also suffers from a decrease in accuracy that comes with the increase in the complexity of the supply-demand condition. When the capacity increases from 4 to 8, LTO's improvement relative to OSP in the average service rate drops by 0.82%, from 4.61% to 3.79%, while the drop for LTO-Basic is 1.41%. This indicates that, by employing online learning and re-optimization, we can learn a better value function and correct for prediction bias. When exploring the patterning of different days, the results show that when the capacity is 4, LTO-Basic serves 3.3% and 3.66% more passengers than Baseline on the 25th and 26th, respectively, indicating that LTO-Basic performs better on the 26th. When the capacity is 8, LTO-Basic serves 2.76% and 2.18% more passengers than Baseline on the 25th and 26th, respectively, indicating that LTO-Basic performs better on the 25th. This demonstrates that mobility patterns can vary on different days. Specifically, when the capacity is 6, the service rates yielded by LTO are 4.48% higher than OSP and 4.73% higher than LTO-Basic on the 25th, and 3.88% higher than OSP and 4.52% higher than LTO-Basic on the 26th.

(H5.1) Figure 5.6 shows the results of varying fleet sizes on the two different days. When the fleet size increases, the system has more independent seats, which gives more flexibility to the dispatcher and also reduces the dispatcher's reliance on sophisticated algorithms. We can tell that as the number of vehicles increases from 1200 to 1800, the lift in LTO-Basic relative to Baseline decreases, which is consistent with the results in [32]. The reason for this may be that a far-sighted dispatcher will tend to ignore immediate rewards and instead save empty seats for future orders. This allows LTO-Basic to achieve an average of 3.62% higher improvement over Baseline when there are not enough vehicles (i.e., $m = 1200$). However,

when there are a sufficient number of vehicles (i.e., $m = 1800$), most of the future requests are quickly taken by vehicles and there are not many left, so LTO-Basic only achieves an average of 0.48% improvement over Baseline. It is also possible that, because the value network is trained on 1500 vehicles, when the number of vehicles increases, which actually increases the supply and decreases the average reward of vehicles, the value becomes over-estimated and inaccurate. OSP, on the other hand, only considers real-time supply-demand conditions and can quickly adapt to changes in fleet size. It has an average improvement of 2.8% and 2.45% versus Baseline when using 1200 and 1800 vehicles, respectively. LTO, combining the value function and re-optimization to make better match decisions, achieves a two-day average improvement of 1.26% over OSP with 1800 vehicles.

(H5.1) Figure 5.7 shows the results of varying values of the maximum waiting time constraint on two different days. The increase in travel delay constraint allows for more detours and therefore makes it easier to carry more passenger. Thus, all algorithms achieve higher service rates as the delay tolerance increases. When the constraint increases from 180 sec to 420 sec, the improvement in the two-day average service rate of LTO-Basic relative to Baseline over the two days increases from 2.65% to 3.17%. Enhanced by online learning and re-optimization, LTO further improves on the basis of LTO-Basic by 2.03%, 4.63% and 5.17%, when the time constraints are 180 sec, 300 sec and 420 sec. Overall, the trend of the change in different algorithms' performance, as the constraint becomes looser, is similar to what happens when changing vehicle capacities. When the constraint is tight (i.e., $\Omega = 180\,\text{sec}$), LTO-Basic performs better than OSP. But when the constraint becomes looser, LTO-Basic has a significant drop in service rate compared to OSP and becomes worse than OSP. The performance of LTO shows that online learning can slow the drop. This is because passengers are more likely to be pooled together and produce a more complex scheduling problem that requires the dispatcher to be more sophisticated. Considering the mobility patterns on the two different days, when the maximum waiting time constraint is 420 sec, the service rates

yielded by LTO are 3.34% higher than OSP and 5.47% higher than LTO-Basic on the 25th, and 2.55% higher than OSP and 4.87% higher than LTO-Basic on the 26th.

(H5.1) Figure 5.8 shows the results of varying lengths of batch period on two different days. As the window length increases, more requests are received per epoch, but the response time of passengers also grows. Requests waste more time waiting for the end of the dispatch window, which in turn leads to an actually tighter maximum waiting time constraint. Thus we can see a significant decrease in the two-day average service rate for both OSP and LTO, with a 2.93% decrease in OSP (from 88.92% to 85.99%) and a 4.22% decrease in LTO (from 93.17% to 88.95%), when the window length is increased from 10 sec to 120 sec. This is because, through re-optimization, they are actually using a batch period as large as possible to perform matching by considering all available requests. For them, a larger batch window would only reduce the time available for detours and produce a worse performance. But for LTO-Basic and Baseline, because they optimize only the current batch window, receiving more new orders offsets the drop in detour time and makes their service rate to drop by less than 1%. In the real world, however, the smaller the window length, the better. When the window length is 10 sec, the service rates yielded by LTO are 4.86% higher than OSP and 4.88% higher than LTO-Basic on the 25th, and 4.25% higher than OSP and 5.15% higher than LTO-Basic on the 26th.

Table 5.2: A comparison of service rate (%) during the peak hour for using online learning ($m = 1500,\ \kappa = 6,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Date of Experimental Data	LTO	LTO Without Online Learning
25 May 2016	91.80	91.32
26 May 2016	92.34	92.08

(H5.2) In the above results, we find that LTO sometimes yields a greater improvement than LTO-Basic on the 25th and sometimes on the 26th, showing that mobility patterns can

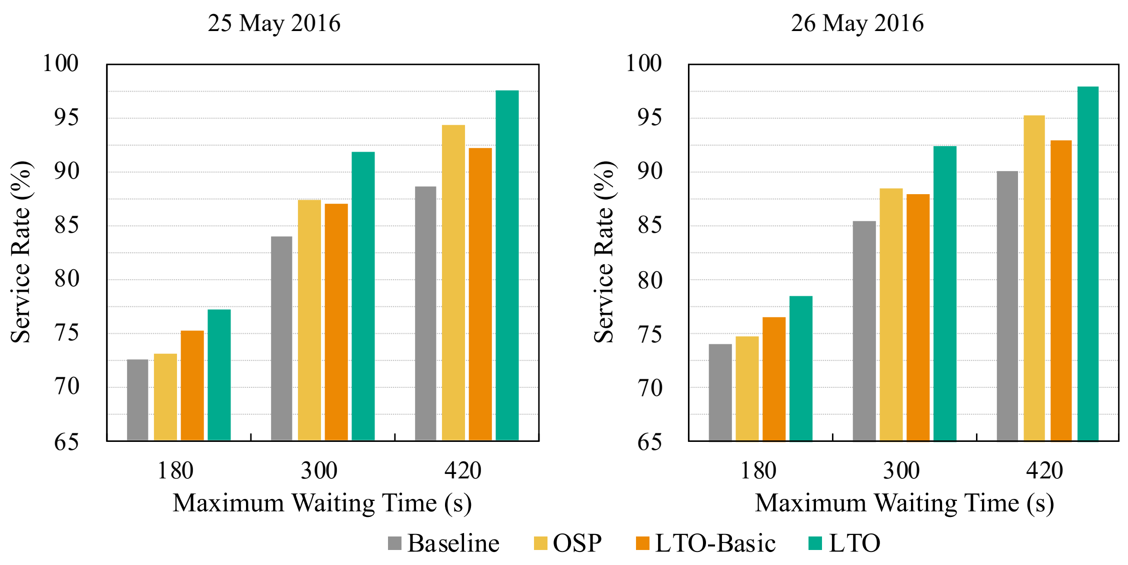


Figure 5.7: A comparison of service rate (%) during the peak hour for varying values of maximum waiting time constraint ($m = 1500,\ \kappa = 6,\ \Delta T = 30\,\text{sec}$).

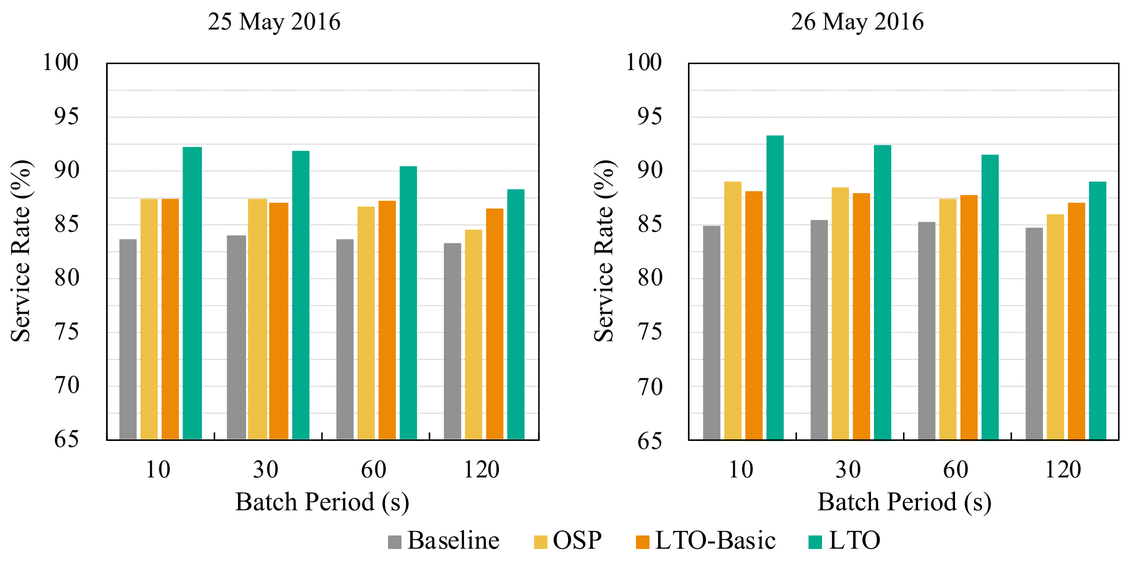


Figure 5.8: A comparison of service rate (%) during the peak hour for varying lengths of batch period ($m = 1500,\ \kappa = 6,\ \Omega = 300\,\text{sec}$).

vary not only on different days but also across different settings. To figure out the role of online learning, we remove the online learning component of LTO to see the difference. Table 5.2 shows the service rates over the two days with the default parameter set. It is shown that online learning yields more improvements on the 25th, which is consistent with the results that LTO yields a greater improvement than LTO-Basic on the 25th under the default experimental setting. The results in Table 5.2 show that online learning only contributes 0.48% and 0.26% to the performance of LTO on the 25th and 26th, suggesting that the supply-demand conditions may vary very little during the peak hour. We then examine the role of online learning during 00:00 - 08:00, where the change in the number of requests is greater than during the peak hour (19:00-20:00), and find that the contributions of online learning to the service rates increase to 1.57% and 1.21% on the 25th and 26th, respectively. This suggests that greater improvements can be achieved through online learning when real-time fluctuations are high.

Table 5.3: A comparison of service rate (%) during the peak hour for varying discount factors ($m = 1500,\ \kappa = 6,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Date of Experimental Data	Discount Factor γ			
	0.90	0.95	0.97	0.99
25 May 2016	90.65	91.80	91.44	90.25
26 May 2016	91.53	92.34	91.97	90.87

We further investigate the impact of varying discount factors. Table 5.3 shows the service rates over the two days. In general, the discount factor implicitly determines the number of future epochs we take into consideration when compute the matching policy. A small discount factor results in a short-sighted strategy and a large one provides a long-sighted goal. We can tell that the service rate rises when γ is increased from 0.9 to 0.95, for being more far-sighted. But the service rate falls if we continue to increase γ. This might be

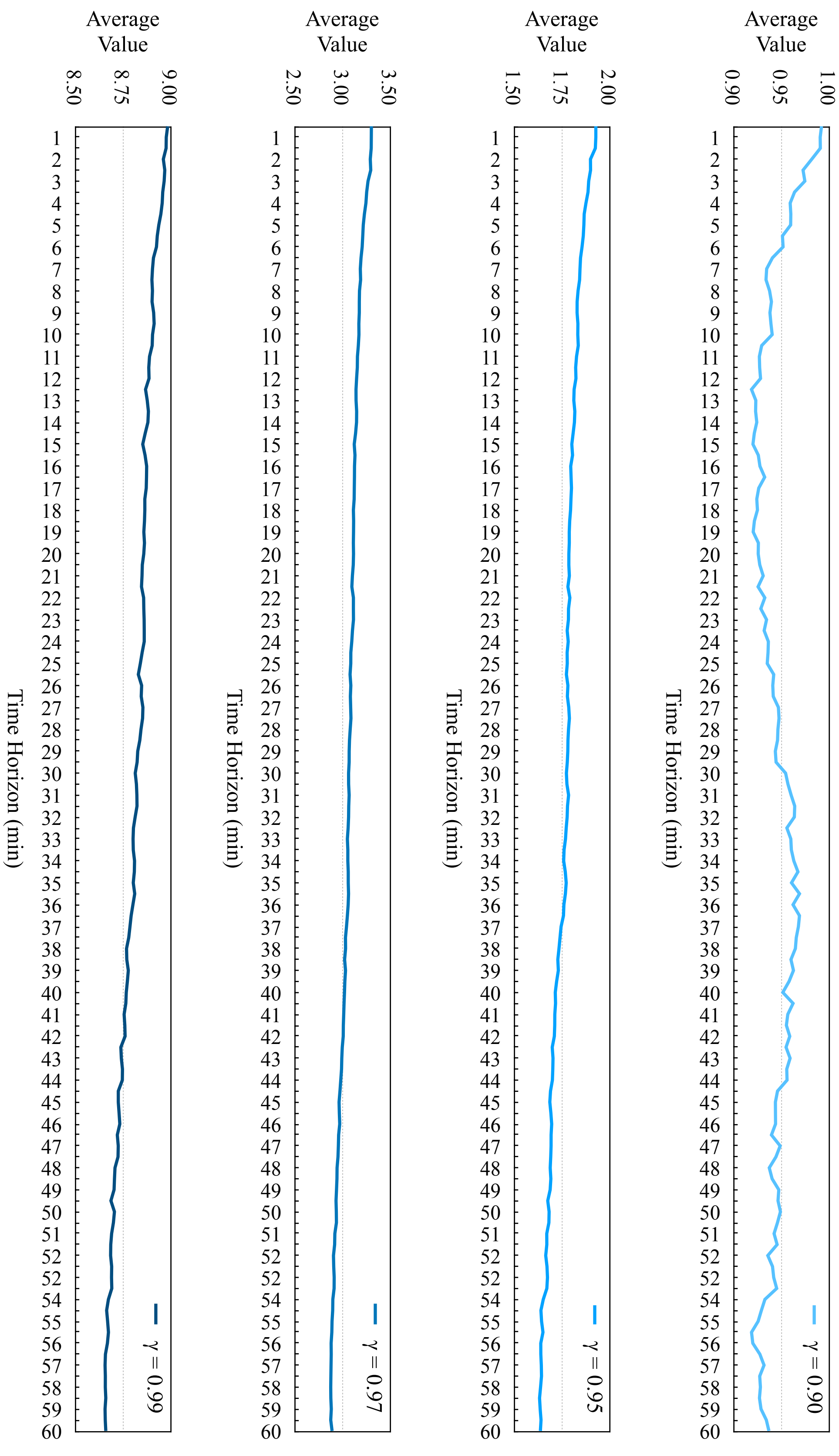


Figure 5.9: Temporal patterns in the learned values for varying time discount factors.

that a very large discount factor induces the dispatcher to optimize the objective over the next few hours, which is longer than the length of our experiments. Figure 5.9 shows the average value of all vehicle states over the time horizon for varying discount factors. The values are around 0.95, 1.75, 3.05 and 8.8 for $\gamma = 0.9$, $\gamma = 0.95$, $\gamma = 0.97$ and $\gamma = 0.99$, respectively. The value indicates the discounted expected number of requests that will be served in the future. We can tell that the larger the discount factor, the further the dispatcher looks ahead and the larger the value. When $\gamma = 0.9$, the curve fluctuates more than the others, because the dispatcher only captures the rewards over a short time (several minutes). When sufficiently far-sighted, the value decreases with the passage of time, because the total number of future passengers decreases.

Table 5.4: A comparison of computation time (sec) of different procedures during the peak hour for varying vehicle capacities ($m = 1500,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Procedures	Vehicle Capacity κ			
	2	4	6	8
Scoring Process in OSP	0.05	0.07	0.07	0.07
Scoring Process in LTO	2.71	2.71	2.75	2.77
Online Training in LTO	4.12	4.18	4.21	4.26

(H5.3) Finally, we investigate the computation times of the scoring process using value network and the online training process. Table 5.4 shows the comparison for varying capacities. The time consumed by the scoring process in OSP is shorter than that in Chapter 4, because of the decrease in the number of candidates due to the decrease in the number of requests (from 400k to 308k). The time consumed by the scoring process in LTO increases only very little when the capacity increases, indicating that its sensitivity to problem size is low, most likely because most of its computation time is spent on data format processing. Therefore, LTO is able to run in real-time. As for the time consumed by the online train-

ing process, since it can run separately from dispatching, it can meet the requirements of real-time deployment as long as the computation time is shorter than the dispatch window.

5.4.4 Discussion

Considering longer horizons when computing a matching policy results in better performance than Baseline. Solely optimizing over all possible past epochs, OSP produces an increase in service rate of up to 5.58% at peak hour on the 25th when $\Omega = 420\,\text{sec}$. LTO-Basic, solely optimizing over the future epochs, yields an increase of up to 3.69% at peak hour on the 25th when $\Delta T = 10\,\text{sec}$. While optimizing over both the future and the past epochs, LTO achieves an improvement over Baseline by 8.92% and 8.57% at peak hour on the 25th when $\Omega = 420\,\text{sec}$ and $\Delta T = 10\,\text{sec}$, respectively.

Learning a good state value function has the potential to significantly improve the performance of a dispatcher. However, due to the highly dynamic nature of urban mobility and the unavoidable prediction bias, the ability to continuously alter the matching policy online is also very important. Moreover, updating the value function during online dispatch has the opportunity to further improve performance. By employing online learning and re-optimization, LTO (our proposed method) captures real-time supply-demand conditions and corrects for deviations with newly revealed requests. Compared to OSP, LTO improves the service rate by up to 5.03% at peak hour on the 25th when $\kappa = 4$, while compared to LTO-Basic, it improves the service rate by up to 5.47% when $\Omega = 420\,\text{sec}$.

5.5 Chapter Summary

In this chapter, we have proposed a deep reinforcement learning based method for far-sighted vehicle dispatch in SAMoD systems. Our work aims to investigate the role of optimization over a long horizon, including both the past and the future. An offline learning method is developed to learn a value function to capture general mobility patterns from historical data. An online updating procedure is then developed to quickly adapt the offline learned value function to real-time dynamics of the system during operation. The value function is embedded into an efficient online dispatcher that uses re-optimization for better performance. Numerical experiments on real large-scale datasets show that the proposed method further improves the dispatcher proposed in Chapter 3 in terms of service rate (up to 5.03% at peak hour). It also yields an improvement over the state-of-the-art (up to 5.47% at peak hour).

Chapter Six

Conclusions

This thesis addresses the problem of improving the efficiency and quality of vehicle dispatch in high-capacity SAMoD systems, by means of (1) an optimal online batch assignment method, (2) a travel time uncertainty aware vehicle dispatch method and (3) a long-term reward optimization method. Regarding efficiency, we work on making explicit use of the revealed deterministic information about the system in real-time. Regarding quality, we work on tackling two types of uncertainty information, i.e., stochastic travel times and dynamic passenger demands. In this chapter, we summarize the results presented in the thesis and discuss the directions of future work.

6.1 Summary

In Chapter 3, we propose a batch assignment method that produces an optimal matching policy for each dispatch window in real-time. It uses an incremental search algorithm which reduces the global search for vehicle scheduling problems to local search through heuristics while ensuring optimality. This means that it can explore all possible vehicle-trip matches by quickly pruning the futile portion of the search space, and thereby compute the optimal matching policy. It also uses a re-optimization strategy to dynamically alter the matching policy to keep it optimal at any given time. Case studies using real-world taxi trip data demonstrate that the proposed method outperforms the leading online dispatch algorithm in terms of service rate and scalability.

In Chapter 4, we propose a multi-phase method that leverages stochastic travel time models to optimize the on-time arrival probability of passengers and the profit of the platform. It consists of three steps: computing candidate matches (which could be done using the method presented in Chapter 3), scoring the candidates with the on-time arrival probabilities, and computing the matching policy by solving a maximum weight matching problem. It also takes into account compensation for late arrivals and scores the candidates based on expected profits. Simulations using real-world taxi trip data demonstrate that, by taking travel time uncertainty into account, the proposed method achieves improvements in service rate, reliability and profit.

In Chapter 5, we propose a learning-based method that captures spatio-temporal mobility patterns to optimize long-term service objectives. It combines offline evaluation and online learning to obtain a value function that captures both the general systematic patterns and the real-time dynamics of the system. The value function is embedded into an online dispatch scheme to optimize the matching policy over a long horizon. The online dispatch scheme is similar to the one in Chapter 4, but using a different scoring process. It also employs the re-optimization strategy in Chapter 3 to extend the matching horizon and correct for the prediction bias. Numerical experiments demonstrate that, by optimizing over a longer horizon, the proposed method yields a higher service rate than the state-of-the-art far-sighted dispatch approach.

In general, Chapter 3 develops an optimal online dispatch method, in which only deterministic information is involved. Chapters 4 and 5 extend and enhance the approach in Chapter 3 by handling different kinds of uncertainty, respectively; they share some similarities in the solution framework, but are optimizing for quite different objectives.

6.2 Future Directions

In this section, we discuss several directions that may extend and improve the methods presented in this work.

Appropriate Fleet Size. The method developed in Chapter 3 focuses on producing the optimal matching policy for a predetermined number of vehicles. However, a fundamental unsolved problem of finding the appropriate number of vehicles to serve all requests has been ignored. It is a central issue for service providers, as deploying too few vehicles can lead to poor passenger experience and loss of customers, while deploying too many vehicles can lead to a waste of resources. Some work has been done on determining the minimum fleet size [131, 132]. It is of interest to investigate other optimization objectives such as how to change the fleet size according to the changes in demand distributions and requirements.

Congestion-Aware Routing. In Chapter 4, the dispatcher assumes that the routing of the controlled vehicles does not affect traffic conditions. With the deployment of large-scale SAMoD systems, we expect the number of vehicles under control to be very large and the matching and routing policy may affect travel times. Congestion-aware routing would be an important problem [33, 114], especially when allowing ride-shairng, as delay time is a very important consideration.

Vehicle Rebalancing. The dispatcher presented in Chapter 5 tries to produce a far-sighted matching policy by considering the supply-demand conditions. There is another active research topic on routing idle vehicles to potentially high demand areas, known as rebalancing [103, 104]. Rebalancing has the potential to influence the supply-demand conditions to serve more requests. It would be an interesting idea to consider the combination of rebalancing and dispatching in ride-sharing, such as learning a unified value function [99].

Public Transit Interaction. Despite the various benefits of the SAMoD systems, public transportation is an integral component of urban mobility. It is important to examine the impact of the SAMoD systems on the public transportation system, as well as build a fusion system [133, 134]. Using a fleet of autonomous vehicles to provide transportation services jointly with public transit could potentially result in lower travel costs and emissions.

References

[1] Marco Pavone. “Autonomous mobility-on-demand systems for future urban mobility”. In: *Autonomes Fahren*. Springer, 2015, pp. 399–416.

[2] Rick Zhang. “Models and Large-scale Coordination Algorithms for Autonomous Mobility-on-demand”. PhD thesis. Stanford University, 2016.

[3] Katarzyna Anna Marczuk. “Modeling and analysis of an autonomous mobility on demand system”. PhD thesis. National University of Singapore, 2017.

[4] Federico Rossi. “On the interaction between Autonomous Mobility-on-Demand systems and the built environment: Models and large scale coordination algorithms”. PhD thesis. Stanford University, 2018.

[5] Daniel James Fagnant. “The future of fully automated vehicles: opportunities for vehicle-and ride-sharing, with cost and emissions savings”. PhD thesis. The University of Texas at Austin, 2014.

[6] Xing Wang. “Optimizing ride matches for dynamic ride-sharing systems”. PhD thesis. Georgia Institute of Technology, 2013.

[7] Paolo Santi et al. “Quantifying the benefits of vehicle pooling with shareability networks”. In: *Proceedings of the National Academy of Sciences* 111.37 (2014), pp. 13290–13294.

[8] Hadi Hosni et al. “The shared-taxi problem: Formulation and solution methods”. In: *Transportation Research Part B: Methodological* 70 (2014), pp. 303–318.

[9] Wen Shen et al. “Toward understanding the impact of user participation in autonomous ridesharing systems”. In: *2018 Winter Simulation Conference (WSC)*. IEEE. 2018, pp. 845–856.

[10] economist.com, ed. *The hidden cost of congestion.* https://www.economist.com/blogs/graphicdetail/2018/02/daily-chart-20. Accessed: 2021-11-18. Feb. 2018.

[11] Regina R Clewlow et al. “Disruptive transportation: The adoption, utilization, and impacts of ride-hailing in the United States”. In: *University of California, Davis, Institute of Transportation Studies, Davis, CA, Research Report UCD-ITS-RR-17-07* (2017).

[12] Alejandro Henao et al. “The impact of ride-hailing on vehicle miles traveled”. In: *Transportation* 46.6 (2019), pp. 2173–2194.

[13] Alejandro Tirachini. “Ride-hailing, travel behaviour and sustainable mobility: an international review”. In: *Transportation* 47.4 (2020), pp. 2011–2047.

[14] Gregory D Erhardt et al. “Do transportation network companies decrease or increase congestion?” In: *Science advances* 5.5 (2019), eaau2670.

[15] Long T Truong et al. “Estimating the trip generation impacts of autonomous vehicles on car travel in Victoria, Australia”. In: *Transportation* 44.6 (2017), pp. 1279–1292.

[16] Remi Tachet et al. “Scaling law of urban ride sharing”. In: *Scientific reports* 7.1 (2017), pp. 1–6.

[17] Patrick M Bösch et al. “Cost-based analysis of autonomous mobility services”. In: *Transport Policy* 64 (2018), pp. 76–91.

[18] Michael Hyland et al. “Operational benefits and challenges of shared-ride automated mobility-on-demand services”. In: *Transportation Research Part A: Policy and Practice* 134 (2020), pp. 251–270.

[19] Javier Alonso-Mora et al. “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment”. In: *Proceedings of the National Academy of Sciences* 114.3 (2017), pp. 462–467.

[20] Matthew Tsao et al. “Model predictive control of ride-sharing autonomous mobility-on-demand systems”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 6665–6671.

[21] Kangjia Zhao et al. “Online Vehicle Dispatch: from Assignment to Scheduling”. In: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2018, pp. 608–611.

[22] Chiwei Yan et al. “Dynamic pricing and matching in ride-hailing platforms”. In: *Naval Research Logistics (NRL)* 67.8 (2020), pp. 705–724.

[23] Zhiwei Qin et al. “Ride-hailing order dispatching at DiDi via reinforcement learning”. In: *INFORMS Journal on Applied Analytics* 50.5 (2020), pp. 272–286.

[24] Shuo Ma et al. “T-share: A large-scale dynamic taxi ridesharing service”. In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE. 2013, pp. 410–421.

[25] Ming Zhu et al. “An online ride-sharing path-planning strategy for public vehicle systems”. In: *IEEE Transactions on Intelligent Transportation Systems* 20.2 (2018), pp. 616–627.

[26] Yongxin Tong et al. “A unified approach to route planning for shared mobility”. In: *Proceedings of the VLDB Endowment* 11.11 (2018), p. 1633.

[27] Andrea Simonetto et al. “Real-time city-scale ridesharing via linear assignment problems”. In: *Transportation Research Part C: Emerging Technologies* 101 (2019), pp. 208–232.

[28] Meghna Lowalekar et al. “ZAC: A zone path construction approach for effective real-time ridesharing”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 528–538.

[29] Connor Riley et al. “Column generation for real-time ride-sharing operations”. In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2019, pp. 472–487.

[30] Andres Fielbaum et al. “Anticipatory routing methods for an on-demand ridepooling mobility system”. In: *Transportation* (2021), pp. 1–42.

[31] Alex Wallar et al. “Vehicle rebalancing for mobility-on-demand systems with ridesharing”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4539–4546.

[32] Sanket Shah et al. “Neural approximate dynamic programming for on-demand ridepooling”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 01. 2020, pp. 507–515.

[33] Federico Rossi et al. “Routing autonomous vehicles in congested transportation networks: Structural properties and coordination algorithms”. In: *Autonomous Robots* 42.7 (2018), pp. 1427–1442.

[34] Soheil Sadeghi Eshkevari et al. “Reinforcement Learning in the Wild: Scalable RL Dispatching Algorithm Deployed in Ridehailing Marketplace”. In: *arXiv preprint arXiv:2202.05118* (2022).

[35] Lingyu Zhang et al. “A taxi order dispatch model based on combinatorial optimization”. In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining.* 2017, pp. 2151–2159.

[36] James Munkres. “Algorithms for the assignment and transportation problems”. In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38.

[37] Ldc Martins et al. “Optimizing Ride-Sharing Operations in Smart Sustainable Cities: Challenges and the Need for Agile Algorithms”. In: *Computers & Industrial Engineering* 153 (2020), p. 107080.

[38] Mitja Stiglic et al. “Making dynamic ride-sharing work: The impact of driver and rider flexibility”. In: *Transportation Research Part E: Logistics and Transportation Review* 91 (2016), pp. 190–207.

[39] Erhun Özkan et al. “Dynamic matching for real-time ride sharing”. In: *Stochastic Systems* 10.1 (2020), pp. 29–70.

[40] Niels Agatz et al. “Optimization for dynamic ride-sharing: A review”. In: *European Journal of Operational Research* 223.2 (2012), pp. 295–303.

[41] Hipólito Hernández-Pérez et al. “The multi-commodity one-to-one pickup-and-delivery traveling salesman problem”. In: *European Journal of Operational Research* 196.3 (2009), pp. 987–995.

[42] Thierry Garaix et al. “Optimization of occupancy rate in dial-a-ride problems via linear fractional column generation”. In: *Computers & Operations Research* 38.10 (2011), pp. 1435–1442.

[43] Sophie N Parragh et al. “A heuristic two-phase solution approach for the multi-objective dial-a-ride problem”. In: *Networks: An International Journal* 54.4 (2009), pp. 227–242.

[44] Gerardo Berbeglia et al. “A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem”. In: *INFORMS Journal on Computing* 24.3 (2012), pp. 343–355.

[45] Gerardo Berbeglia et al. “Dynamic pickup and delivery problems”. In: *European journal of operational research* 202.1 (2010), pp. 8–15.

[46] Karl F Doerner et al. “Chapter 7: Pickup-and-delivery problems for people transportation”. In: *Vehicle Routing: Problems, Methods, and Applications, Second Edition.* SIAM, 2014, pp. 193–212.

[47] Neda Masoud et al. “A decomposition algorithm to solve the multi-hop peer-to-peer ride-matching problem”. In: *Transportation Research Part B: Methodological* 99 (2017), pp. 1–29.

[48] Amirmahdi Tafreshian et al. “Trip-based graph partitioning in dynamic ridesharing”. In: *Transportation Research Part C: Emerging Technologies* 114 (2020), pp. 532–553.

[49] Amirmahdi Tafreshian et al. “Frontiers in service science: Ride matching for peer-to-peer ride sharing: A review and future directions”. In: *Service Science* 12.2-3 (2020), pp. 44–60.

[50] Ali Najmi et al. “Novel dynamic formulations for real-time ride-sharing systems”. In: *Transportation research part E: logistics and transportation review* 108 (2017), pp. 122–140.

[51] Roozbeh Ketabi et al. “Playing with matches: vehicular mobility through analysis of trip similarity and matching”. In: *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* 2018, pp. 544–547.

[52] Wesam Herbawi et al. "The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm". In: *2012 IEEE Congress on Evolutionary Computation*. IEEE. 2012, pp. 1–8.

[53] Marius M Solomon. "Algorithms for the vehicle routing and scheduling problems with time window constraints". In: *Operations research* 35.2 (1987), pp. 254–265.

[54] Jizhe Xia et al. "A new model for a carpool matching service". In: *PloS one* 10.6 (2015), e0129257.

[55] Alp M Arslan et al. "Crowdsourced delivery—A dynamic pickup and delivery problem with ad hoc drivers". In: *Transportation Science* 53.1 (2019), pp. 222–235.

[56] Ruimin Ma et al. "A novel algorithm for peer-to-peer ridesharing match problem". In: *Neural Computing and Applications* 31.1 (2019), pp. 247–258.

[57] P Gruebele. "Interactive system for real time dynamic multi-hop carpooling". In: *Global Transport Knowledge Partnership* (2008), p. 28.

[58] Wesam Herbawi et al. "Evolutionary multiobjective route planning in dynamic multi-hop ridesharing". In: *European conference on evolutionary computation in combinatorial optimization*. Springer. 2011, pp. 84–95.

[59] Neda Masoud et al. "A real-time algorithm to solve the peer-to-peer ride-matching problem in a flexible ridesharing system". In: *Transportation Research Part B: Methodological* 106 (2017), pp. 218–236.

[60] Libin Zheng et al. "Order dispatch in price-aware ridesharing". In: *Proceedings of the VLDB Endowment* 11.8 (2018), pp. 853–865.

[61] Der-Horng Lee et al. "Taxi dispatch system based on current demands and real-time traffic conditions". In: *Transportation Research Record* 1882.1 (2004), pp. 193–200.

[62] Kiam Tian Seow et al. "A collaborative multiagent taxi-dispatch system". In: *IEEE Transactions on Automation science and engineering* 7.3 (2009), pp. 607–616.

[63] Minne Li et al. “Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning”. In: *The World Wide Web Conference*. 2019, pp. 983–994.

[64] Zhe Xu et al. “Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 905–913.

[65] Dominic Widdows et al. “Grabshare: The construction of a realtime ridesharing service”. In: *2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*. IEEE. 2017, pp. 138–143.

[66] Peng Cheng et al. “Utility-aware ridesharing on road networks”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 1197–1210.

[67] Douglas Oliveira Santos et al. “Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem”. In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013.

[68] Jaeyoung Jung et al. “Dynamic shared-taxi dispatch algorithm with hybrid-simulated annealing”. In: *Computer-Aided Civil and Infrastructure Engineering* 31.4 (2016), pp. 275–291.

[69] Yinglei Li et al. “Ride-sharing under travel time uncertainty: Robust optimization and clustering approaches”. In: *Computers & Industrial Engineering* 149 (2020), p. 106601.

[70] Dawn Woodard et al. “Predicting travel time reliability using mobile phone GPS data”. In: *Transportation Research Part C: Emerging Technologies* 75 (2017), pp. 30–44.

[71] Sejoon Lim et al. “Stochastic motion planning and applications to traffic”. In: *The International Journal of Robotics Research* 30.6 (2011), pp. 699–712.

[72] Evdokia Nikolova et al. “Stochastic shortest paths via quasi-convex maximization”. In: *European Symposium on Algorithms*. Springer. 2006, pp. 552–563.

[73] Evdokia Nikolova. “Approximation algorithms for reliable stochastic combinatorial optimization”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2010, pp. 338–351.

[74] Sejoon Lim et al. “Practical route planning under delay uncertainty: Stochastic shortest path queries”. In: *Robotics: Science and Systems*. Vol. 8. 32. 2013, pp. 249–256.

[75] Zhiguang Cao et al. “Finding the shortest path in stochastic vehicle routing: A cardinality minimization approach”. In: *IEEE Transactions on Intelligent Transportation Systems* 17.6 (2016), pp. 1688–1702.

[76] Sejoon Lim et al. “Stochastic motion planning with path constraints and application to optimal agent, resource, and route planning”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 4814–4821.

[77] Astrid S Kenyon et al. “Stochastic vehicle routing with random travel times”. In: *Transportation Science* 37.1 (2003), pp. 69–82.

[78] Xiangyong Li et al. “Vehicle routing problems with time windows and stochastic travel and service times: Models and algorithm”. In: *International Journal of Production Economics* 125.1 (2010), pp. 137–145.

[79] Duygu Taş et al. “Vehicle routing problem with stochastic travel times including soft time windows and service costs”. In: *Computers & Operations Research* 40.1 (2013), pp. 214–224.

[80] D Taş et al. “Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach”. In: *European Journal of Operational Research* 236.3 (2014), pp. 789–799.

[81] Baoxiang Li et al. “The share-a-ride problem with stochastic travel times and stochastic delivery locations”. In: *Transportation Research Part C: Emerging Technologies* 67 (2016), pp. 95–108.

[82] Shangyao Yan et al. “A car pooling model and solution method with stochastic vehicle travel times”. In: *IEEE transactions on intelligent transportation systems* 15.1 (2014), pp. 47–61.

[83] Jiancheng Long et al. “Ride-sharing with travel time uncertainty”. In: *Transportation Research Part B: Methodological* 118 (2018), pp. 143–171.

[84] Xiaoming Li et al. “Ride-Sharing Matching under Travel Time Uncertainty through A Data-Driven Robust Optimization Approach”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 3420–3425.

[85] Zhiguang Liu et al. “Dynamic shared autonomous taxi system considering on-time arrival reliability”. In: *Transportation Research Part C: Emerging Technologies* 103 (2019), pp. 281–297.

[86] Jin Y Yen. “Finding the k shortest loopless paths in a network”. In: *management Science* 17.11 (1971), pp. 712–716.

[87] Jun Gao et al. “Fast top-k simple shortest paths discovery in graphs”. In: *Proceedings of the 19th ACM international conference on Information and knowledge management*. 2010, pp. 509–518.

[88] Xiaocheng Tang et al. “A deep value-network based approach for multi-driver order dispatching”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 1780–1790.

[89] Yan Jiao et al. “Real-world Ride-hailing Vehicle Repositioning using Deep Reinforcement Learning”. In: *arXiv preprint arXiv:2103.04555* (2021).

[90] Javier Alonso-Mora et al. “Predictive routing for autonomous mobility-on-demand systems with ride-sharing”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3583–3590.

[91] Xianan Huang et al. “Efficient mobility-on-demand system with ride-sharing”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 3633–3638.

[92] Meghna Lowalekar et al. “Zone pAth Construction (ZAC) based Approaches for Effective Real-Time Ridesharing”. In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 119–167.

[93] Yang Liu et al. “Proactive rebalancing and speed-up techniques for on-demand high capacity ridesourcing services”. In: *IEEE Transactions on Intelligent Transportation Systems* (2020).

[94] Connor Riley et al. “Real-time dispatching of large-scale ride-sharing systems: Integrating optimization, machine learning, and model predictive control”. In: *arXiv preprint arXiv:2003.10942* (2020).

[95] Ramon Iglesias et al. “Data-driven model predictive control of autonomous mobility-on-demand systems”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 6019–6025.

[96] Zhaodong Wang et al. “Deep reinforcement learning with knowledge transfer for online rides order dispatching”. In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 617–626.

[97] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[98] James S Albus. “A theory of cerebellar function”. In: *Mathematical biosciences* 10.1-2 (1971), pp. 25–61.

[99] Xiaocheng Tang et al. “Value Function is All You Need: A Unified Learning Framework for Ride Hailing Platforms”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 3605–3615.

[100] Ishan Jindal et al. “Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining”. In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 1417–1426.

[101] Hado Van Hasselt et al. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[102] Xian Yu et al. “An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.9 (2019), pp. 3811–3820.

[103] Jian Wen et al. “Rebalancing shared mobility-on-demand systems: A reinforcement learning approach”. In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. Ieee. 2017, pp. 220–225.

[104] Daniele Gammelli et al. “Graph Neural Network Reinforcement Learning for Autonomous Mobility-on-Demand Systems”. In: *arXiv preprint arXiv:2104.11434* (2021).

[105] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[106] Rick Zhang et al. “Control of robotic mobility-on-demand systems: a queueing-theoretical perspective”. In: *The International Journal of Robotics Research* 35.1-3 (2016), pp. 186–203.

[107] James J Pan et al. “Ridesharing: simulator, benchmark, and evaluation”. In: *Proceedings of the VLDB Endowment* 12.10 (2019), pp. 1085–1098.

[108] Bilong Shen et al. “Dynamic ridesharing”. In: *Sigspatial Special* 7.3 (2016), pp. 3–10.

[109] Michal Cáp et al. “Multi-Objective Analysis of Ridesharing in Automated Mobility-on-Demand”. In: *Robotics: Science and Systems*. 2018.

[110] Hongjian Wang et al. “A simple baseline for travel time estimation using large-scale trip data”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–22.

[111] NYC Taxi et al. *TLC Trip Record Data*. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page. Accessed: 2021-11-19. July 2021.

[112] Dimitris Bertsimas et al. “Travel time estimation in the age of big data”. In: *Operations Research* 67.2 (2019), pp. 498–515.

[113] Transport forLondon. *Travel in London Reports*. https://tfl.gov.uk/corporate/publications-and-reports/travel-in-london-reports. Accessed: 2022-07-20. July 2022.

[114] Mauro Salazar et al. “A congestion-aware routing scheme for autonomous mobility-on-demand systems”. In: *2019 18th European Control Conference (ECC)*. IEEE. 2019, pp. 3040–3046.

[115] Tao Xing et al. “Finding the most reliable path with and without link travel time correlation: A Lagrangian substitution based approach”. In: *Transportation Research Part B: Methodological* 45.10 (2011), pp. 1660–1679.

[116] Bi Yu Chen et al. “Finding reliable shortest paths in road networks under uncertainty”. In: *Networks and spatial economics* 13.2 (2013), pp. 123–148.

[117] Hailong Huang et al. “Reliable path planning for drone delivery using a stochastic time-dependent public transportation network”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.8 (2020), pp. 4941–4950.

[118] Bin Yang et al. “PACE: a PAth-CEntric paradigm for stochastic path finding”. In: *The VLDB Journal* 27.2 (2018), pp. 153–178.

[119] Simon Aagaard Pedersen et al. “Fast stochastic routing under time-varying uncertainty”. In: *The VLDB Journal* 29.4 (2020), pp. 819–839.

[120] Patricia June Carstensen. "The complexity of some problems in parametric linear and combinatorial programming". PhD thesis. University of Michigan, 1983.

[121] Philippe Artzner et al. "Coherent measures of risk". In: *Mathematical finance* 9.3 (1999), pp. 203–228.

[122] Matthew Norton et al. "Calculating CVaR and bPOE for common probability distributions with application to portfolio optimization and density estimation". In: *Annals of Operations Research* (2019), pp. 1–35.

[123] Xiucheng Li et al. "Learning travel time distributions with deep generative model". In: *The World Wide Web Conference*. 2019, pp. 1017–1027.

[124] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Vol. 703. John Wiley & Sons, 2007.

[125] Richard S Sutton et al. *Reinforcement learning: An introduction*. MIT press, 2018.

[126] Wouter van Heeswijk. *What Are Post-Decision States and What Do They Want From Us?* https://towardsdatascience.com/what-are-post-decision-states-and-what-do-they-want-from-us-9e02105b7f40. Accessed: 2021-12-21. May 2021.

[127] Marcin Szubert et al. "Temporal difference learning of n-tuple networks for the game 2048". In: *2014 IEEE Conference on Computational Intelligence and Games*. IEEE. 2014, pp. 1–8.

[128] Warren B Powell. "What you should know about approximate dynamic programming". In: *Naval Research Logistics (NRL)* 56.3 (2009), pp. 239–249.

[129] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[130] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[131] Mohammad M Vazifeh et al. “Addressing the minimum fleet problem in on-demand urban mobility”. In: *Nature* 557.7706 (2018), pp. 534–538.

[132] Alex Wallar et al. “Optimizing multi-class fleet compositions for shared mobility-as-a-service”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 2998–3005.

[133] Mauro Salazar et al. “On the interaction between autonomous mobility-on-demand and public transportation systems”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 2262–2269.

[134] Jian Wen et al. “Transit-oriented autonomous vehicle operation with integrated demand-supply interaction”. In: *Transportation Research Part C: Emerging Technologies* 97 (2018), pp. 216–234.

[135] Michal Kümmel et al. “Taxi dispatching and stable marriage”. In: *Procedia Computer Science* 83 (2016), pp. 163–170.

[136] A. Prokhorchuk et al. “Estimating Travel Time Distributions by Bayesian Network Inference”. In: *IEEE Transactions on Intelligent Transportation Systems* (2019), pp. 1–10.