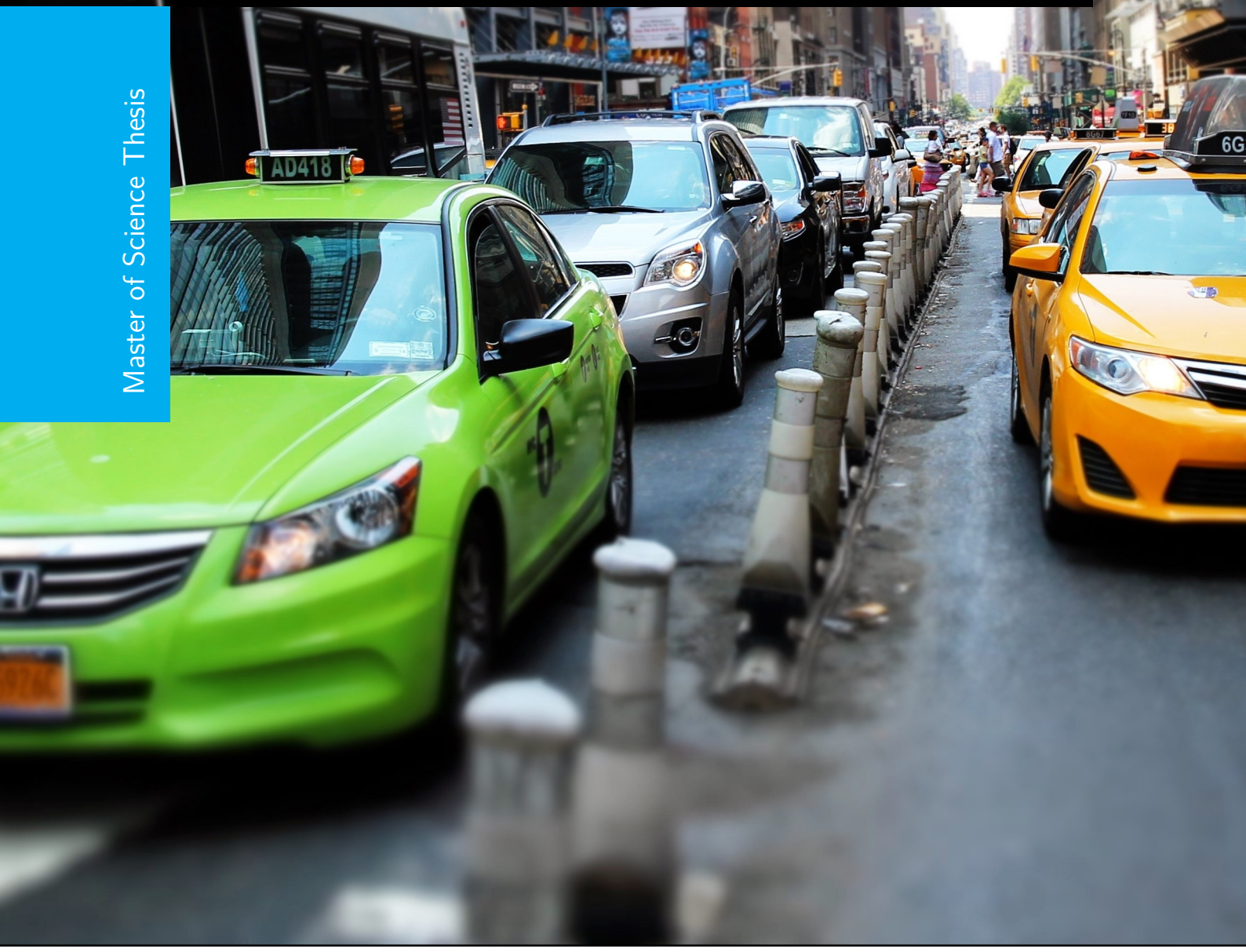
Repositioning in shared mobility systems

Combining model predictive control and approximate dynamic programming.

M.G. Vinks

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Repositioning in shared mobility systems

Combining model predictive control and approximate dynamic programming.

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

M.G. Vinks

August 17, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Repositioning in shared mobility systems

by

M.G. Vinks

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: August 17, 2022

Supervisor(s):

Dr.ir. A. Dabiri Supervisor

Dr.ir. F. Schulte Second Super

Reader(s):

Dr.ir. A. Dabiri. First Reader

Dr.ir. F. Schult Reader-two

Dr.ir. B.Beirigo. Th. Reader-three

Abstract

The global market for personal mobility has transformed over the last decade. Traditional taxi services have to compete with the emergence of ride-hailing services such as Uber and Lyft. Rapid developments in available algorithms and real-time inter-connectivity of travellers and vehicles offer mobility-on-demand (MOD) services new possibilities to maximise the efficiency of the ride-hailing process. It is well established that rebalancing can enable the true potential of a ride-hailing fleet. In this context, rebalancing is defined as the redistribution of idle vehicles over the service area of a ride-hailing operator.

In the search for the optimal rebalancing strategy, model predictive control (MPC) and approximate dynamic programming (ADP) methodologies are active fields of research. However, the computational burden of MPC limits the length of the predictive horizon in large MOD applications. This thesis proposes a novel algorithm that combines ADP and MPC. More specifically, we integrated a value function into the MPC framework as a terminal cost. We shorten the horizon of MPC and propose to use the value function as a long-term planner.

For the terminal cost, we continue research into piece-wise linear value function approximation. We implement a novel multi-period approximation of the value function, where we use the maximum repositioning length as the horizon. In our case study, the value-based repositioning strategy offers comparable service quality to conventional MPC at 5.2% of the computational burden.

The hybrid ADP-MPC algorithm offers flexible horizon partitions between the multi-period value function and MPC. It addresses the shortcomings of both ADP and MPC algorithms in repositioning problems. At peak performance, it offers an increase of 4.5% in service rate and a decrease of 5.2% in the average waiting time over conventional MPC, at roughly half the computational burden.

Keywords: Ride-hailing, Mobility on demand, Model predictive control, Approximate Dynamic programming

Table of Contents

Acknowledgements

When I started working on my thesis in November of 2020, I was oblivious to the challenges I still to come. Over this past year and half, I have learned that overcoming habits is one the key challenges of life. Luckily, I had proper guidance along the way. I would like to thank all my supervisors, A. Dabiri, F. Schulte and B. Beirigo for their assistance and guidance during this thesis.

Delft, University of Technology M.G. Vinks
August 17, 2022

"In the future, car ownership will be the exception, not the rule. Shared mobility will give the streets to the people and the personal automobile will be remembered as the product of capitalistic consumerism."

— *Ties Vinks*

Chapter 1

Introduction

Urbanisation of the world over the last century has put tremendous pressure on our transportation networks. Existing transportation infrastructure often operates at or near full capacity, whilst demand keeps increasing [1]. Congestion and low air quality have become a characteristic of fast-growing cities [2]. Spatial and economic incentives often prohibited the expansion of car infrastructure in city centres. Alternative modes of travel, such as metro and tram, deal with overcrowding during peak hours. Furthermore, public transport is subject to the last mile problem, which refers to the availability of transportation services from the nearest public transport hub to a home or office. These issues trouble the access to mobility of citizens. Innovations in the last decades have led to the emergence of shared mobility services as a new type of last-mile transportation. Shared mobility—the shared use of a vehicle, bicycle, or other mode—is an innovative transportation strategy that enables users to gain short-term access to transportation modes on an "as-needed" basis [3]. By a shift towards collaborative use of service through the advancements in location-based services and cloud technologies, shared mobility-on-demand (MOD) services rise to complement the existing services [4].

MOD is a transportation concept led by innovation. It capitalises on the data exchange among travellers and other parts of the transportation infrastructure, such as train and bus networks, to generate integrated and multi-modal options for travelling. Furthermore, it encourages inter-connectivity between transportation modes and promotes choice in personal mobility. According to MOD, transportation is a commodity where each mode of travel has economic values that are distinguishable in terms of cost, journey time, wait time, the number of connections, convenience, and other attributes [3]. Through MOD, travellers have access to a network of safe, affordable and reliable transport options.

In the search for efficient last-mile transportation, ridesharing has become an ambiguous term. According to Furuhata et al.[5]: "Ridesharing refers to a mode of transportation in which individual travellers share a vehicle for a trip and split travel costs". However, ridesharing is frequently used as a collective term for all types of on-demand transportation companies [6, 7, 8, 9]. In this work, we share the second perspective and use ride-pooling for the mode

of transportation where a single vehicle is shared among multiple individual travellers and ridesharing as the term for shared mobility services.

In the ridesharing market, ride-hailing is currently the dominant service type [10]. Traditional ride-hailing operators provide door-to-door transport in an urban environment. Customers order a car from a mobile application, where after the available driver pool receives the origin and destination of the ride request. Once a driver accepts the request, the customer receives the current location of the driver and an estimated pick-up time. The ride-hailing driver picks up the client and transports him to their destination.

Ride-hailing companies, such as Uber, operate a network of independent drivers that provide transportation to customers in privately owned vehicles. Traditionally, ride-hailing platforms just offered a constant influx of trips in exchange for a share of the ride fare. The focus was to provide a convenient alternative mode for last mile trips, while generating maximum revenue for the operator. However, MOD services focus on a traveller centric ride-hailing service, where the goal is to provide the highest quality journey from a travellers perspective. In ride-hailing, quality can be expressed as journey time. In the achievement of minimal journey time in ride-hailing, fleet management plays an integral role.

In this thesis, we consider ridesharing modes from a travellers perspective. The appeal of a mode is measured primarily in terms of availability and price [11]. The former is the focal point of active fleet management strategies. The availability is expressed in terms of total journey time, divided into waiting and in-vehicle time. The average waiting time is a metric of the balance between the ridesharing supply and demand. Fleet management strategies relocate idle vehicles to provide maximum availability for potential customers. In ridesharing, this type of predictive positioning problem is known as a rebalancing problem. Proper rebalancing can provide additional revenue for ridesharing operators and drivers. Ultimately, it optimises the waiting time of customers.

1-1 Problem statement

A key operational challenge in shared mobility is the problem of asymmetric demand. In morning rush hours, most demand originates from residential areas to business districts. Without proper fleet management, asymmetric demand inevitably depletes some areas of vehicles while in other areas vehicles accumulate. In this work, we aim to improve the data and computational efficiency of fleet management in ridesharing.

Ridesharing fleet management deals with the allocation of resources, i.e., vehicles, to tasks, i.e., ride-request. In this type of resource allocation problem, we seek to find the optimal allocation of a fixed amount of resources to activities to minimise the cost incurred by the allocation [12]. In a rebalancing problem, the goal is to allocate resources in real-time in response to stochastic demand. For idle vehicles, we find optimal locations that minimise the response time to future demand. A rebalancing strategy finds a minimal cost routing solution for idle drivers and ensures optimal availability of the ridesharing service. Devising efficient operating strategies for ride-hailing fleet management is an active area of research.

Model predictive control (MPC) is a well-established model-based control method that first gained popularity in the control of industrial chemical processes at the end of the last century [13]. MPC combines three main elements: a predictive model, feedback correction, and

optimisation over a receding horizon. The framework is very flexible in design and within MOD research MPC optimises the availability of the fleet within a prediction horizon [8, 9, 14, 15, 16]. It produces a fleet management strategy by repeatedly solving an optimisation problem using predictions of future demand and the movement of vehicles. The complexity of the optimisation problem is mainly dependent on model complexity and size. As most ride-hailing companies operate regional services, the MOD models that capture the door-to-door dynamics are extensive and highly complex [17]. Therefore, the computational burden of successful use of MPC in repositioning problems is high.

In contrast to predictive fleet management, rebalancing strategies can be learned iteratively. Approximate dynamic programming (ADP) uses a value function, an efficient method to determine the value of being in a state, to solve sequential decision-making problems. A fundamental property of value functions is that they satisfy recursive relationships. During the iteratively learning process, the value function is continually updated and exploited. The process is terminated once the value function no longer changes. This optimal value function is then used to efficiently solve sequential decision-making problems. Due to the complexity of the fleet management problem, an exact version of the value function cannot be computed but has to be approximated. In fleet management, ADP methodologies are an active field of research [18, 19, 20]. Due to the iterative approximation of the value function, large-scale problems suffer from what is known as the curse of dimensionality [21] . It refers to the exponential increase in learning computations required as we increase problem complexity and size. Since the approximation of value function can be done preemptively, a hybrid predictive and learning strategy can potentially reduce the computational burden of conventional predictive strategies.

1-2 Research questions

In this work, we explore the benefit of rebalancing strategies for better availability in on-demand transportation. The main goal is to reduce the computational complexity of MPC methods by reducing the prediction horizon while maintaining it's performance. To this extent, we aim to answer the following main research question:

> Can MPC and ADP be combined for computationally efficient rebalancing of MOD ride-hailing services?

The main challenge of the thesis will be to design a value function approximation (VFA) that can be implemented within an MPC framework. The following sub-questions will be used to answer the main research question:

1. *Can we capture the dynamics of ride-hailing into a MOD model?* We design a MOD model that closely mimics the real-life interactions of a ride-hailing service. The simulation of door-to-door ride-hailing is essential for the assessment of repositioning strategies.
2. *Can we learn a value function for rebalancing operations MOD in networks?* ADP offers a plenty of different VFA methods. We focus on linear and piece-wise linear value approximation.

3. *Can value function approximation be utilised as a terminal cost within the MPC framework to provide real-time realisable rebalancing of MOD in networks?* We design a hybrid framework for optimal control of MOD systems. A value function provides the value of states beyond the horizon of the MPC.

1-3 Thesis outline

The outline of this work is as follows. Chapter 2 offers an introduction to ride-hailing and introduces a general MOD model. Chapter 3 shortly recaps the background knowledge for ADP and presents two value-based learning algorithms. Chapter 4 introduces model predictive control and presents a novel hybrid learning and planning repositioning algorithm. The proposed strategies of Chapter 3 and Chapter 4 are evaluated in the case studies in Chapter 5. Finally, the main conclusions are discussed in Chapter 6.

Chapter 2

On-demand transportation

This chapter gives a general introduction in on-demand transportation. We offer insight into the process flow of a typical ride-hailing service and shortly introduce the general rebalancing problem. Lastly, we introduce a ride-hailing **mod!** (**mod!**) model.

2-1 Shared mobility

The rise of shared mobility services in the recent years have made the last mile trip more convenient. Fig. 2-1 showcases the evolution of on-demand transportation enabled by innovations in location-based services and connectivity.

Currently, ride-hailing services such as Lyft and Uber complement the public transport services. Ride-hailing companies enable travellers to match up with independent drivers at short notice. The popularity of Uber and Lyft show the appeal of ride-hailing apps. Additionally, (electric) micro-mobility services have risen as a new form of last-mile trip transportation. This section gives a short introduction to ride-hailing and micro-mobility.

2-1-1 Ride-hailing

Ride-hailing is a modern taxi service where you hail a ride from a mobile application rather than a street corner. A typical ride-hailing application integrates three core modules: pricing, matching and repositioning. Fig. 2-2 highlights these decision modules in the ride-hailing process. Supplementary routing modules find the shortest travel time between an origin and a destination. They provide turn-by-turn guidance to the drivers on the road network. The *pricing* module computes the costs of transportation. First, each ride request is priced and represented to the customer. Upon acceptance by the customer, the request is added to the open order list. Afterwhich, the *matching* module assigns orders to drivers. Each order is matched to a driver from the idle driver pool. When a match is found, the driver picks up the passenger and fulfils the inquiry. Upon drop-off, the driver receives a fare and is reintroduced

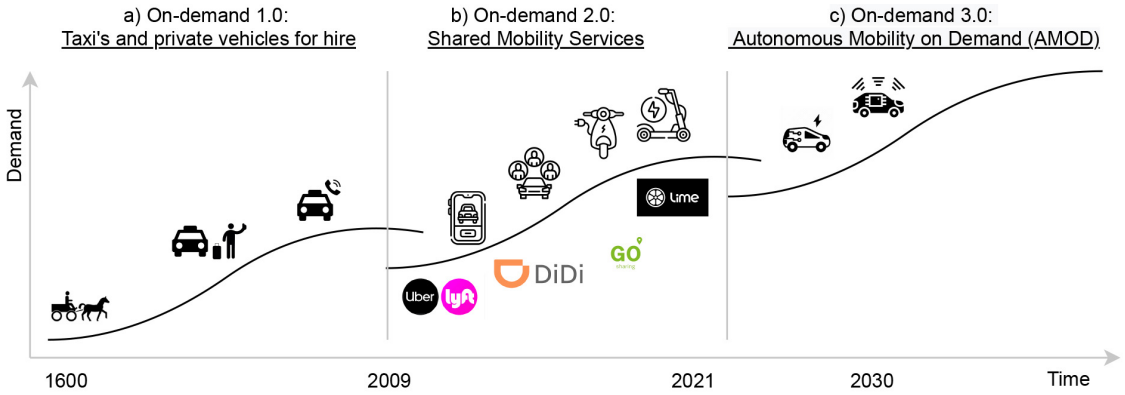


Figure 2-1: The evolution of on-demand transportation in three phases(inspired by [22]). (a) On-demand 1.0: A reflection of a market dominated by dail-a-ride and traditional taxi services. (b) On-demand 2.0: Status quo, the rise of shared mobility services (c) On-demand 3.0: Prospect of a future with fully autonomous mobility services.

into the available driver pool. For the customer, the ridesharing process has ended. The driver can remain idle at the drop off location until a new ride request is presented or he can reposition. The *repositioning* module shows idle drivers promising locations in the prospect of future ride inquiries.

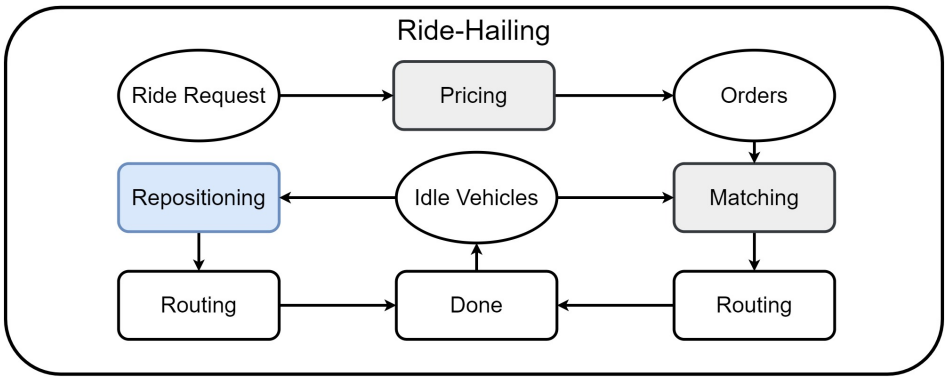


Figure 2-2: The process flow of a ridesharing service [6]. The coloured boxes represent different decision modules. In this work, we evaluate repositioning strategy by simulation of a ride-hailing service.

Ride-hailing drivers only accept one ride request at a time. In *ride-pooling* mode, a driver can accept multiple requests at a time. To enable ride-pooling, the process flow of ride-hailing service has to be reworked and extended, which is beyond the scope of this work. Ride-pooling is more related to dynamic pickup and delivery problems, where spatiotemporally distributed demand must be picked up in and delivered in specific time windows [23, 24]. In literature, ridesharing can refer to both ride-pooling and ride-hailing. To avoid ambiguity, in this work we consider a rebalancing problem to be of the ride-hailing type.

2-1-2 Micro-mobility

Vehicle rental services have been around for decades, however, advancements in location-based services have lifted shared mobility rental schemes to a new level. Classic station-based ridesharing rental schemes require a customer to return the leased vehicle to a fixed set of stations. For example, in the city of London the public bicycle scheme, commonly known as Boris bikes, has been operating for over ten years [25]. The stations are strategically spread over the city, located near other transport hubs. However, these station-based services suffered from several flaws that have been tackled by the new micro-mobility services.

Since building and maintaining physical stations is expensive, micro-mobility services operate a fleet of ridesharing vehicles that freely roam the service area. The type of vehicles ranges from (electric) mopeds to bicycles and scooters. Location-based services within each vehicle place restrictions on vehicle parking. At transportation hubs designated areas are reserved for the parking of micro-mobility vehicles [26].

In micro-mobility, customers drive to their destination by short-time lease of a rental vehicle. The vehicles can be unlocked through a ridesharing app and are often paid by the minute. The station-based vehicles had a longer rental time and first have to be returned to the rental station before another customer could use them [25]. A customer pays in advance for a rental period, e.g., 24 hours, and has to return it within this given period. However, micro-mobility vehicles, once parked, are ready to be used by another customer.

From a MOD perspective, micro-mobility services largely draw from the same demand pool as ride-hailing services. Differences in convenience, e.g. driving yourself versus being driven, make that the customer bases do not entirely overlap. However, micro-mobility is another option for efficient last-mile transportation. Thus, the demand pattern of both modes is comparable. Therefore, reposition of idle vehicles is relevant to micro-mobility. For the current generation of micro-mobility vehicles, additional personal is required for repositioning operations. Developments in autonomous routing reduce the need for human interference. As we move toward autonomous mobility-on-demand (AMOD) services in the near foreseeable future, ride-hailing and micro-mobility services will merge into a universal on-demand service.

2-1-3 Shared mobility problems

In this section, we give an introduction to the core operational problems in ride-hailing. We shortly cover pricing and matching problems and then introduce the general rebalancing problem.

In ride-hailing process, each new request first visits the pricing module, as seen in Fig. 2-2 The customer receives a quota of the estimated trip fare and can choose to either reject or accept the quota. The trip fare denotes the cost for passengers and the income of the drivers. Pricing is a macro-level lever to influence the supply and demand of ride-hailing [27, 6]. Both are sensitive to the price of trips. Pricing problems explore the sensitivity of customers and drivers to pricing decisions. A prime example of price sensitivity is price surging, increasing prices in times of higher demand. Dynamic pricing policies respond in real-time to fluctuations in supply and demand.

The matching of riders and drivers is a bipartite matching problem where both supply and demand are dynamic [28]. Ridesharing matching falls into the broader class of dynamic

matching problems for on-demand markets [29]. Challenges in matching problems come from the uncertainty in travel times, supply-demand (SD) distribution and exit-entrance behaviours of drivers and passengers. Matching can be done continuously or in batches at timed intervals. A distinct feature for ridesharing matching is the spatiotemporal nature. Trips times vary in length and can be influenced by the presence of congestion. Each trip changes the spatial state of a driver and their eligibility to accept new requests is dependent on their proximity to the request. Within the scope of ridesharing, rebalancing is a tool to influence the supply distribution of a matching problem.

In this work, we consider the operators perspective to a fleet rebalancing problem. An operator needs to construct sets of routes for idle ridesharing vehicles. The proposed routes aim to improve the overall availability of ridesharing vehicles over a given service area. The system-wide rebalancing problem, also known as fleet management, vehicle allocation/repositioning or driver allocation problem in literature, is, in essence, a resource allocation problem. This problem seeks to find an optimal allocation of a fixed amount of resources to activities so as to minimise the cost incurred by the allocation [12]. We differentiate a rebalancing problem in that repositioning is done in real-time to deal with non-stationary global SD distributions.

The aim is to proactively dispatch idle vehicles to new locations to better align the vehicles and customers distribution. Alignment of vehicle and customer distribution is beneficial for all parties in ridesharing services. The operator's primary objectives are to maximise revenue and offer the best customer experience. In shared mobility services, only occupied vehicles generate revenue. Ride-hailing drivers revenue is fare-based and thus a higher occupancy rate will increase his overall income. A driver might incur additional costs for moving empty, however, the additional revenue generated from potential customers will compensate for these costs. To maximise revenue, both the operator and driver want to increase the overall occupancy rate.

The occupancy rate is improved through better availability of the service. From a MOD perspective, customers opt for the most efficient mode of transport. In general, travellers will opt for the ridesharing mode with the shortest queue time. The queue time, also known as waiting time, denotes the time spent waiting between transportation modes. The goal is to optimise the average waiting time for all potential customers.

From the view of spatio-temporal movement, order matching can be viewed as a special case of vehicle repositioning trough trip fulfilment [30]. From this perspective, a single problem can solve the matching and rebalancing tasks simultaneously. In this case, the challenges of the matching problem are inherited. We deal with uncertainty in travel times, SD distributions and exit-entrance behaviour of ride-hailing drivers and passengers. The general MOD ride-hailing problem consists of the joint optimisation of pricing, matching and vehicle rebalancing. In this work, we emphasise the rebalancing aspect of the ride-hailing problem.

2-2 Model description and problem formulation

In this section, we propose a fluid-based network optimisation problem for vehicle repositioning and demand matching. We formulate the vehicle repositioning and demand matching optimisation problem as a mixed-integer linear program (MILP) We present our basic model as a deterministic Markov decision process, which is applicable to many other types of resource

allocation applications [20, 31]. For national convenience and clarity, our initial formulation assumes that all decisions take a single time period to complete. In Section 2-2-3, we dismiss this assumption and present its implications on our solution methodology.

2-2-1 Routing network

For the spatial-temporal representation of the service area of a shared mobility service, we follow the language of dynamic resource management [19, 20, 32]. In this framework, vehicles are resources and trips are loaded movements. We want to service trip requests over a maximum period of time T for a given service area:

$\mathcal{T}$ = Set of time periods over which we want to full-fill rideh-hailing demand, $\{0, 1, \ldots, T\}$.

We update the model at discrete intervals Δt. Each vehicle is described by a single attribute vector a, which solely captures the current location of the car:

$$a = \begin{pmatrix} a_1 \end{pmatrix} = \begin{pmatrix} \text{location} \end{pmatrix}$$

$\mathcal{A}$ = Set of all possible car attribute vector a.

The location a_1 corresponds to a node in the street network graph $G(\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is the set of nodes (locations) and $\mathcal{E}$ is a set of directed edges (streets). Modern ride-hailing applications are connected via the cellular network which provide precise GPS coordinates of drivers and passengers in real-time. We work with a high-resolution street network, obtained from OpenStreetMap [33] using OSMnx [34]. As in Beirigo et al. [17], we discretize real-world street coordinates into a network of nodes, in which nodes cannot be further than thirty seconds away from another, assuming an average speed of 20km/h. The real-world street coordinates are replaced by the closest node in the street network. In this fine grain street network, we simulate door-to-door ride-hailing trips.

We similarly describe each trip with a three attributes vector b:

$$b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} n_o \\ n_d \\ w \end{pmatrix} = \begin{pmatrix} \text{origin} \\ \text{destination} \\ \text{delay} \end{pmatrix}$$

$\mathcal{B}$ = Set of trip attribute vectors b.

Where b_1 and b_2 describe the origin and destination nodes of a trip respectively. Each trip request has a built-in delay attribute w. Upon arrival of a ride-request, we initialise $w = 0$. Individual trip orders arrive in continuous time, the trips arriving between $t-1$ and t are therefore collected and serviced at time step t. Trips that cannot be serviced at the current time step t are backlogged and added to the next trip batch at time $t+1$, with their delay b_3 incremented. Upon reaching their delay threshold w_{max}, a trip is rejected.

2-2-2 Optimal rebalancing

In this work, we simultaneously consider the trip matching and vehicle repositioning problems. However, vehicle move on the street-level network, while matching and repositioning decisions are made based on an aggregate network.

Aggregate network

To define the aggregate network, the service area is commonly split in to zones either using a grid of geometric shapes [35, 19], district borders (see Fig. 2-3) [36, 37] or a data driven-approach, based on the origin and destination of trips [38, 15, 8]. To identify the balance between ride-hailing supply and demand, we simplify the street network graph into a network of virtual stations $\mathcal{I}$. We aggregate the demand and supply of each zone at a virtual station $i \in \mathcal{I}$, as in Fig. 2-3(c). We represented each zone with a node i and construct an aggregate network for the demand matching and repositioning problems.

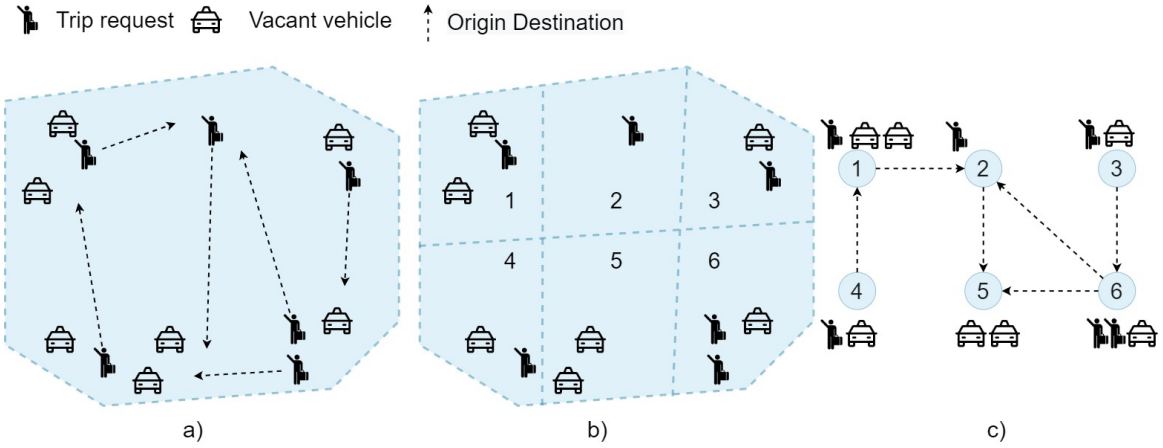


Figure 2-3: Steps in network aggregation. a) Locate all current trip requests and vehicles. b) Divide the service area in zones. c) Aggregate all demand and vehicle at the virtual station for each zone and identify the inter-zonal demand

In this work, we employ regional centres that serve as virtual stations, adapted from Beirigo et al. [17]. The network of regional centres is the result of a variant of the facility location problem as proposed by Toregas et al. [39]. The goal of this problem is to determine the minimum set of facilities in a street network graph $G(\mathcal{N}, \mathcal{E})$ that can be reached by all others within a set time limit of s time units.

System State

We gather relevant ride-hailing information in our state vector S_t, which we define as:

$$S_t = (R_t, D_t) = \text{The current state of the system at time } t.$$

The state vector details the number of vehicles R_t and the current ride-hailing demand D_t present within each zone at time t. For each node i, we define a resource vector R_{it} as:

$$\begin{aligned} R_{it} &= \text{The number of resources with at location } i \in \mathcal{I} \text{ at time } t \in \mathcal{T} \text{ .} \\ R_t &= \text{The resource state vector at time } t. \\ &= (R_{it})_{i \in \mathcal{I}} \end{aligned}$$

and a demand vector D as:

$$\begin{aligned} D_{ijt} &= \text{Number of trips from origin } i \in \mathcal{I} \text{ to destination } j \in \mathcal{I} \text{ at time period } t \in \mathcal{T} \\ D_t &= \text{The demand state vector at time } t. \\ &= (D_{ijt})_{i,j \in \mathcal{I}} \end{aligned}$$

Decisions

Given the current state S_t of the network, we define the different movements within the network. The movements are decision variables of the problem. We formulate the following types of decision variables:

x^p_{ijt} = Number of vehicles moving loaded from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$.

x^r_{ijt} = Number of vehicles moving empty from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$.

x^d_{ijt} = Number of trip requests with origin $i \in \mathcal{I}$ and destination to $j \in \mathcal{I}$ not serviced at time period $t \in \mathcal{T}$.

In this formulation, staying in place is seen as an empty movement x^r with $i = j$. Our decision variables are limited by the following constraints:

$$\sum (x^p_{ijt} + x^d_{ijt}) = D_{ijt} \quad \forall i\,, j \in \mathcal{I}\,, \forall t \in \mathcal{T} \tag{2-1a}$$

$$\sum_{j \in \mathcal{I}} \left(x^p_{ijt} + x^r_{ijt} \right) = R_{it} \quad \forall i\,, \in \mathcal{I}\,, \forall t \in \mathcal{T} \tag{2-1b}$$

$$x^r_{ijt}\,, x^p_{ijt}\,, x^d_{ijt} \in Z_+ \quad \forall i\,, j \in \mathcal{I}\,, \forall t \in \mathcal{T} \tag{2-1c}$$

where Eq. (2-1a) denotes the conservation of passengers, Eq. (2-1b) denotes the conservation of vehicles and Eq. (2-1c) ensures integer movements.

Transition Function

In our network, we describe how the position of our vehicles changes over the simulation period. Since each decision only takes a single period Δt, the resource state at the next time period can be described for a node i as:

$$R_{j,t+1} = \sum_{i \in \mathcal{I}} (x^r_{ijt} + x^p_{ijt}) \quad \forall j \in \mathcal{I}, \forall t \in \mathcal{T} \tag{2-2}$$

To compute the new demand vector D_{t+1}, we first backlog all trips that have not been serviced at the current time step. For each route between nodes i and j, we first increment the delay of each individual trip b in x^d_{ijt}. We consequently remove the trips where $w > w_{max}$ and denote the residual demand as $d_{ij,t+1}$.

We assume the arrival of new trips to be exogenous information. This is new information that arrives from an exogenous source between $t-1$ and t. We collect new demand for ride-hailing arriving between time t and $t+1$ into $\hat{D}_{t+1}$. For each origin destination pair, we describe the new demand vector $D_{ij,t+1}$ at time $t+1$ as:

$$D_{ij,t+1} = d_{ij,t+1} + \hat{D}_{ij,t+1} \quad \forall i \in \mathcal{I}\, j \in \mathcal{I}\,, \forall t \in \mathcal{T} \tag{2-3}$$

Cost function

In this work, all decisions have an associated contribution or cost. For clarity, we include the travel time τ_{ij} between node i and j, which we assume is equal to Δt for now. For the trip decisions, we can denotes the contribution as:

$$c^p_{ijt} = p_{\text{base}} + (p_{\text{time}} - c_{\text{time}}) \cdot \tau_{ij} - c_{\text{delay}} \cdot w_d \tag{2-4}$$

Where c_{time} represents the operational cost of a vehicle. p_{base} and p_{time} denote the base and time depended fare respectively. In this work, we do not focus on the effect of pricing schemes on service quality. We therefore adopt the pricing schemes of [18], where the base fare and time depended fare are derived from TLC standardised metered fare [40]. We penalise for the current delay w with the cost parameter c_{delay}. We acknowledge that this formulation depreciates delayed trips, however we cover this during the decision execution, Section 2-2-3.

For the repositioning tasks, we formulate the cost as follows:

$$c^r_{ijt} = \begin{cases} c_{\text{stay}}, & i = j \\ c_{\text{time}} \cdot \tau_{ij} & \text{otherwise} \end{cases} \tag{2-5}$$

where the c_{time} represents the operational cost of a vehicle and τ_{ij} the travel time between node i and j. c_{stay} represent the operational cost of staying idle i.e., parking cost

For the backlogged trips, we denote the cost as follow:

$$c^d_{ijt} = \begin{cases} c_{\text{reject}}, & w = w_{max} \\ c_{\text{delay}} \cdot w & \text{otherwise} \end{cases} \tag{2-6}$$

where c_{reject} denotes the cost of rejecting a trip.

We summarise the contribution of all decisions at time t in our general contribution as:

$$C(x_t) = \sum_{i,j \in \mathcal{I}} (c^p_{ijt} x^p_{ijt} - c^r_{ijt} x^r_{ijt} - c^d_{ijt} x^d_{ijt}) \tag{2-7}$$

where the first term denotes the gain from servicing trips, the second the cost resulting from the repositioning operations and the third a penalty for backlogging passengers.

For any initial state S_0 and a known demand pattern D, we can formulate the deterministic objective function F as:

$$F(S_0) = \max_x \sum_{t \in \mathcal{T}} C(x_t) = \sum_{t \in \mathcal{T}} \sum_{i,j \in \mathcal{I}} (c^p_{ijt} x^p_{ijt} - c^r_{ijt} x^r_{ijt} - c^d_{ijt} x^d_{ijt}) \tag{2-8}$$

Fleet configuration

MOD promotes new businesses models that enhance the traveller experience [41]. Recent advancements in autonomous routing have sparked interest in the potential of fully autonomous ride-hailing services. As such, AMOD systems could rise as the next big step in personal mobility.

In this work, we opt for the employment of fully autonomous vehicles. Whilst, current autonomous vehicles still have limited capability when it comes to handling complex road situations, in mobility on demand literature it is commonly assumed that fully autonomous vehicles are available [42, 6]. The assumption merges the ridesharing drivers and operators into one entity, removing one human element from the system. It removes uncertainty in the behaviour of ridesharing drivers. To elaborate, in ridesharing services the revenue of drivers is generally fare-based. Each driver will want to maximise their revenue. Competition among drivers will make drivers act selfishly. Human selfish drivers can choose to decline short trips or ignore rebalancing requests. Furthermore, human drivers can choose to enter and exit the

system at any time. Implementing the entrance and exit behaviour of drivers is complex and introduces stochasticity to the fleet size [32]. Autonomous vehicles are fully compliant and will always follow the instructions given, removing internal competition.

We consider a fixed fleet size $|R|$. For any given demand scenario, the optimal fleet-size can be determined by the optimal rebalancing strategy of Iglesias et al. [8]. This formulation introduces the starting positioning and size of the fleet as a decision variable R_0 at the first time step. For a known deterministic demand pattern D_T, we formulate the following optimal rebalancing optimisation problem:

$$\max_{x,R_0} \sum_{t\in\mathcal{T}} \sum_{i,j\in\mathcal{I}} (c^p_{ijt} x^p_{ijt} - c^r_{ijt} x^r_{ijt}) \tag{2-9a}$$

subject to

$$R_{j,t+1} = \sum_{i\in\mathcal{I}} (x^r_{ijt} + x^p_{ijt}) \quad \forall j \in \mathcal{I}, \forall t \in \mathcal{T} \tag{2-9b}$$

$$\sum x^p_{ijt} = D_{ijt} \quad \forall i\,, j \in \mathcal{I}\,, \forall t \in \mathcal{T} \tag{2-9c}$$

$$R_{it} = 0\,, \forall i \in \mathcal{I}\,, \forall t \in \mathcal{T}\,, t > 1 \tag{2-9d}$$

$$x^r_{ijt}\,, x^p_{ijt}\,, R_{i0} \in Z_+ \quad \forall i\,, j \in \mathcal{I}\,, \forall t \in \mathcal{T}, \tag{2-9e}$$

Where Eq. (2-9b) denote the dynamics of the fleet as in Eq. (2-2), Eq. (2-9c) ensures trips are serviced instantly and Eq. (2-9e) ensures all variables are integers. Eq. (2-9d) ensures that cars only get injected at the first timestep.

The effects off different fleet-size configurations will be further discussed in Chapter 5

2-2-3 Multi period travel times

In this section, we introduce multi-period travel times to our model. We elaborate on the changes in vehicle dynamics and extend the model to multi-period travel times. In reality, factors such as congestion, weather and time of day influence the travel time between nodes. While these disturbances can be account in problem formulation (see [43, 44]), deterministic formulations are more conventional in MOD literature [8, 9, 42]. In our model, we assume that the flow between two nodes i and j is deterministic and therefore has a fixed length:

$$\tau^s_{ij} = \text{Travel time from location } i \in \mathcal{I} \text{ to } j \in \mathcal{I} \text{ in seconds}$$
$$\tau_{ij} = \text{Travel time from location } i \in \mathcal{I} \text{ to } j \in \mathcal{I} \text{ in update intervals } \Delta t$$

The travel time τ^s_{ij} is derived from the shortest path between the nodes i and j in the street level graph $G(\mathcal{N}, \mathcal{E})$ with the assumed average speed of 20 km/h. We round up the travel time to the closest discrete time interval to get τ_{ij}. In the conversion of the travel time from seconds to time steps, we account for worst-case scenarios.

To define the system dynamics of the network, we extend the resource vector R_t to account for vehicles inbound to node i:

$$R_{it't} = \text{Number of vehicles inbound to location } i \in \mathcal{I} \text{ at time period } t \in \mathcal{T}$$
$$\text{and will arrive at location } i \in \mathcal{I} \text{ at time period } t' \in \mathcal{T}$$

We define τ_{max} as the maximum travel time within the model. We can consequently describe the dynamics of the network as:

$$R_{jt',t+1} = \sum_{i \in \mathcal{I}} \left(\mathbf{1}_{\tau_{ij}} \left(t' - t \right) x^r_{ijt} + \mathbf{1}_{\tau_{ij}} \left(t' - t \right) x^p_{ijt} \right) + R_{jt't} \quad j \in \mathcal{I}, t' > t \tag{2-10}$$

where we use

$$\mathbf{1}_y(x) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{2-11}$$

The decisions in ride-hailing are integers and can take multiple periods to complete. In most repositioning problems, rebalancing operations are expected to only last a single period of time, i.e., at each time interval all vehicles are either servicing a customer or idle [37, 36, 19]. The single period assumption imposes a strict constraint on the granularity of the network. Furthermore, it is computationally expensive to find the optimal decision for all idle vehicles at every time interval. The lower the resolution of the underlying map, the fewer locations are available, and the more multi-period travel times can be expected between location pairs. Multi-period travel times can reduce the computational complexity of the problem by reducing the number of optimisation variables and states. We therefore set list a set of specific assumptions and rules within the next section for executing decisions.

Decision execution

Once we have found all decisions for the aggregate network, we match each decision to a vehicle on the street network. Since multiple vehicles and trips might be in the same region, we use priority rules for matching decisions to vehicles. We first cover the trip decisions then the repositioning and consider the idle decisions last.

To match vehicles and trip decisions, we use the following routine. For each virtual station i, we first rank the trip requests in descending order based on the current delay w. We find the vehicle closed to the first request and assign it to service this request. The vehicle drives to the pick-up location and transports the customer to their destination. We remove the trip request and vehicle from the available pool and repeat these steps until all trip decisions are covered.

Next, we consider the repositioning tasks. While repositioning, vehicles are not available for new decisions. Hence, we keep the repositioning tasks as short as possible.

Repositioning follows a similar logic to trip matching. We denote the positions of the remaining vehicles in their regional centre i. Simultaneously, we consider all the street-level nodes of the regional centre, j. We find the shortest path between the current vehicle locations and a node within the regional centre j. We assign the repositioning task to the vehicle with the shortest travel time from centre i to centre j. We repeat this routine until all repositioning tasks are assigned to a vehicle.

The execution of decisions is prone to induce errors between the street-level network and the regional centre network. First, the aggregate network does not account for the time between assignment and pick-up. In the aggregate network, trips get picked up instantly and the

travel time between centre i and j is fixed. Secondly, the trip destination node within j can be closer to or further than the regional centre node j. Lastly, repositioning tasks are kept as short as possible by moving to the closed node within the centre j.

The above-mentioned points all contribute to cars approximately finishing a task in the travel time between i and j. Since the travel time between nodes is deterministic on the street-level network, we can compute the accurate arrival information. We correct for errors between the street-level network and aggregate network. At the next time step, the aggregate problem is solved with updated arrival times.

2-3 Summary

This thesis focuses on the repositioning of shared mobility systems. The goal is to minimise the delay through combined matching and repositioning of a MOD fleet. In the near future, AMOD shared mobility services will provide door-to-door service to their customers.

To simulate the effect of different algorithmic rebalancing strategies in terms of quality of service, we present a deterministic AMOD model where all trips and actions have a static price and length. For vehicle routing, we employ a street level network based on real-world physical infrastructure. However, at street level, the demand matching and repositioning problems will be too computationally expensive to solve. Furthermore, repositioning in single increments on street level is tremendously inefficient. We therefore solve the combined demand matching and repositioning problems on an aggregate level. The aggregate model reduces the amount of optimisation variables of the problem. We account for errors between the aggregate model and street-level network upon execution decisions.

Chapter 3

Approximate dynamic programming for MOD

Over the past decades, ADP has become popular as methodology to solve challenging sequential decision-making problems. These methods can learn innovative strategies by iteratively solving sequential decision-making problems. The potential of ADP spans many domains such as robotics [45], self-driving cars [46], games [47] and many more. In this chapter, we elaborate on the basics of ADP. We introduce the ADP strategy known as approximate value iteration. Where after, we present two methods to approximate the value function, based on linear and piecewise-linear value functions.

3-1 Introduction to ADP

One of the cornerstones of ADP is to store the merit of past experiences into a value function. This value function is included in the decision-making process. For each state visited, we store the objective, i.e., Eq. (2-8), as the value associated with that state. We can write the decision making problem in terms of a recursion that relates the value of being in a particular state at the current time to the value of the states that we can visit at the next point in time. For a deterministic resource allocation problem, we can write this as:

$$V_t(S_t) = \max_{x_t} \{C_t(S_t, x_t) + V_{t+1}(S_{t+1})\} \tag{3-1}$$

where S_{t+1} is the state we transition to if we take decision x_t from S_t and C_t denotes the contribution, i.e., Eq. (2-7). Eq. (3-1) is known as the Bellman optimality equation [31], often referred to as just Bellman's equation.

Dynamic programming (DP) refers to a collection of algorithms that solve a sequential-decision making problem using Bellman's equation. Bellman's equation is used to solve a multi-step optimisation problem by recursively solving smaller sub-problems, given that the

problem adheres to the Markov property [48]. This property asserts that future states only depend on the present state. Thus, the next state only depends on the current state and action. A finite Markov decision process (MDP) is a sequential decision making-process restricted to a set of finite states, rewards and actions. At the current stage, DP methods are restricted to finite MDPs.

In DP, we always follows a policy π, a rule for making decisions. The policy π specifies what our decision x will be given state s. One of the key steps in DP is how we compute the value function v_π of an arbitrary policy π:

$$v_\pi(s) = \sum_x \pi(x \mid s) \left(C(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}\left(s' | (s,x)\right) v_\pi(s') \right) \quad \forall s \in \mathcal{S} \tag{3-2}$$

where $\pi(x|s)$ is the probability of taking action x in state s under policy π, $\mathbb{P}(s' \mid s_t, x_t)$ denotes the one-step transition probability matrix that $s_{t+1} = s'$. s' and γ denote the future state and discount factor respectively. This step is known as *policy evaluation*. We can use Eq. (3-2) to iteratively compute the values of states associated with a policy π. In DP, we are interested in the optimal policy π^*, the policy that returns the optimal value function v^*. To find this optimal policy, we use our knowledge of the system, the value function, to come up with a better policy. This strategy, knows as *policy improvement*, computes an improved policy given the value function of our current policy:

$$\pi'(s) = \arg\max_x \left(C(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}\left(s' | (s,x)\right) v_\pi(s') \right) \quad \forall s \in \mathcal{S} \tag{3-3}$$

Policy improvement provides a strictly better policy except when the original policy is already optimal [49].

Classic DP methods operate in sweeps through the state set, the value of one state is based on the values of all successor states and their probabilities of occurring. Two basic DP algorithms, policy iteration and value iteration work by alternating the policy evaluation and improvement steps. They obtain a sequence of monotonically improving policies and value functions through policy evaluation and policy improvement.

Value iteration combines policy evaluation and iteration into update operation of the value function:

$$v^n(s) = \max_x \left(C(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}\left(s' | (s,x)\right) v^{n-1}(s') \right) \quad \forall s \in \mathcal{S} \tag{3-4}$$

We start with an initial guess of the value function for all states and iteratively update the estimates of the value function by sampling the states. We iterate until the value function converges within some tolerance. The optimal policy can then be extracted from the optimal value function by a policy improvement step.

In policy iteration, the policy is fixed a number of policy evaluation steps. We then perform policy improvement to find a new policy. After which the process is repeated. We alternate

between policy evaluation and policy improvement until the optimal policy is found. Policy iteration typically converges to the optimal policy in fewer iterations than value iteration [49]. Policy iteration and value iteration can be used to compute the optimal policy for finite MDPs given complete knowledge of the MDP.

One of the problems within DP, is the need for full knowledge of $\mathbb{P}$. We hardly ever have complete knowledge of the MDP. In order to find the optimal value function, we would need to visit all states recursively and update their value functions. For some small problems, this can be feasible but tedious computation. The number of computations grows exponentially with regard to the number of states and decisions. This is the widely known "curse of dimensionality" of dynamic programming and is the most often-cited reason why dynamic programming cannot be used [21].

3-2 Approximate value iteration

Value iteration is a method for obtaining the optimal value function and consequently the optimal policy. However, as with most classic DP algorithms, it suffers from the curse of dimensionality. For large problems, it is not feasible to recursively visit all possible states and compute their values. Rather than compute the exact value function, we can resort to value function approximation strategies.

Approximate value iteration is a dynamic programming method that exploits the post-decision and subsequent pre-decision state [31]. It breaks the transition from one state to the next into two steps. The post-decision state is the state of the MDP after an action, but just before we arrive at the next state. For a problem with state S_t, this works as follows. At the deterministic post-decision state, no new information has arrived yet:

$$S_t^x = s^{S,x}(S_t, x_t) \tag{3-5}$$

where $s^{S,x}$ is a transition function, e.g. Eq. (2-2), that describes how the system evolves from state S_t to post-decision state S_t^x when taking action x_t.

From the post decision resource state, we can compute the subsequent pre-decision state:

$$S_{t+1} = s^{S^x,W}(S_t^x, \hat{W}_{t+1}) \tag{3-6}$$

where $s^{M,W}$ is a transition function describing how the system evolves from the arrival of new information $\hat{W}_{t+1}$ e.g., arrival of ride requests between timestep t and timestep $t+1$.

Approximate value iteration splits the Bellman equation, Eq. (3-1), into two parts:

$$V_t(S_t) = \max_{x_t} \{C_t(S_t, x_t) + V_t(S_t^x)\} \tag{3-7}$$

$$V_t(S_t^x) = \mathbb{E}\{V_{t+1}(S_{t+1}|S_t^x\} \tag{3-8}$$

Due to the curse of dimensionality, we can not compute the exact value function $V_t(S_t^x)$, but aim to find $\bar{V}_t(S_t^x)$, an approximation of the value function around the post-decision state S_t^x.

In fleet management problems, the deterministic resource transition function is exploited to compute a value function of the post-decision resource vector $V_t(S_t^x) = V_t(R_t^x)$ [18, 19, 20].

This strategy ignores post-decision demand state D_t^x, the backlogged trips. It assumes that trips are not backlogged and D_t^x is empty. We elaborate on the potential use of a non-empty post-decision demand state D_t^x in Section 3-2-4. In this work, we adhere to the common fleet management strategy and replace $V(S_t^x)$ with $V(R_t^x)$. At each time period t of iteration n, we compute the estimate value $\hat{V}(R_t, \hat{D}_t)$ of the current resource state R_t and demand realisation $\hat{D}_t$ as:

$$\hat{V}_t(R_t, \hat{D}_t) = \max_{x_t} C(x_t) + \bar{V}_t^{n-1}(R_t^x) \tag{3-9}$$

where $\bar{V}_t^{n-1}$ denotes the current estimate of expected contribution of the value associated with R^x after $n-1$ iterations. The strategy is to iteratively produce sample demand scenarios $\hat{D}$ that produces outcomes $(\hat{D}_1, \hat{D}_2, \hat{D}_3, \ldots, \hat{D}_T)$ and solve a subproblem for each time period t.

Given we found an estimate of the $\hat{V}_t$ for iteration n, we update our approximation $\bar{V}_t$ according to the update rule:

$$\bar{V}_t^n = (1-\alpha)\,\bar{V}_t^{n-1} + \alpha \hat{V}_t \tag{3-10}$$

where α denotes the learning rate, a parameter that specifies the worth of the current sample compared to our previous approximation. The learning rate takes a value between 0 and 1, however, it does not have to be constant. In the first few iterations, we might want a high learning rate while after hundreds of iterations the learning rate is preferably small.

The strategy works as follows, we initialise V^0 for every $t \in \mathcal{T}$, generally a value of 0 works well. Starting at an initial resource state R_0, we sequentially solve each sub-problem Eq. (3-9) for $t \in \mathcal{T}$ and update the value function according Eq. (3-10). We keep computing iterations till the value function now longer changes.

Construction of an approximation of the value function is one of the challenges of ADP [31]. In general, we can use our knowledge of the problem structure to help construct appropriate value functions. For example, in resource allocation, we have resource R (such as water or oil) available and want to construct the value $V(R)$ of this resource. Logically, water is more valuable during a drought than during the rainy season. For most resources, the general value goes up when the resource is sparse and demand is high while the value goes down if the resource is available in abundance. The value of a resource thus depends on the current demand and availability of this resource. We can exploit these characteristics of resources to find a proper value function. In this work, we focus on linear and piece-wise linear approximation and refer the reader to Powell et al. [31] for a comprehensive overview of value function approximation methods.

3-2-1 Gradients and value functions

One key insight for resource allocation problems is to use the derivatives to estimate value functions, rather than the value of being in a state [18, 19]. In principle, estimating the derivative of a function is not harder then estimating a function itself [31]. For decision making in resource allocation, the most important factor for allocation is the added marginal value of the allocated resource.

Estimating slopes has some practical advantages. For example, if the function is approximately linear, we can replace the estimates of being in each state or set of states with a single

parameter that estimates the slope of that function. Secondly, as common in ADP, you only estimate the value of a resource when it is in a particular state. However, when you estimate a gradient, you get an estimate of the derivative for each resource:

$$\nabla_{R_t} V_t(R_t) = \begin{pmatrix} \hat{v}_{1t} \\ \hat{v}_{2t} \\ \vdots \\ \hat{v}_{|\mathcal{I}|t} \end{pmatrix}, \tag{3-11}$$

where

$$\hat{v}_{it} = \frac{\partial V_t(R_t)}{\partial R_{it}} \tag{3-12}$$

Where the partial derivative $\hat{v}_{it}$ estimates the marginal value of additional resource at node i for time t. In typical ADP, we would only compute the gradient for $R_{it} > 0$, the states where we actually have resources available. To obtain each element of the gradient, additional work might be required. For linear programs, partial derivatives are naturally returned as part of the optimisation, also known as dual variables. For other problems, we might resort to numerical derivatives. However, solving small perturbations to a previously solved problem is relatively inexpensive, especially compared to constructing the value function itself [31].

3-2-2 Linear approximations

Linear functions are the simplest non-trivial value function approximation. Its ease of use is one of the key strengths. Linear approximations work well in two settings. In the first, the true value function is linear or approximately linear over the range of interest. Depending on problem structure linear approximation may or may not be an appropriate choice. For example, if we approximate the contribution of 5 additional ridesharing vehicles for a large city and a village. In the large city, the contribution of 5 additional vehicles to a large fleet will likely not influence demand for ridesharing and the contribution of each individual car will be constant. We could then approximate the value of each additional vehicle with a linear approximation. While in a small village, which may only operate 20 vehicles, each addition can spark additional demand or influence the average profit per vehicle. In this case, the contribution of each vehicle is not constant and would need to be approximated by a non-linear value function.

Likewise, linear approximations are useful in the management of complex resources, i.e., resources with a lot of different attributes. For example, in a fleet management problem, a car's attribute vector a can constitute more than just the location. It might include capacity, fuel load, fuel type or domicile and many more. Naturally, our attributes a is denoted by a vector and let R_{ta} be the number of resources with attribute a at time t. If a is complex, let us say a vector with ten different attributes, we rarely get two resources with the same attribute vector a. Thus the number of resources with attribute vector a is then typically 0 or 1.

In this case, a linear value function suffices. We simply need to construct an approximation of the slope between 0 and 1. It is very convenient that linear approximations only need a single parameter per attribute since complex settings already suffer from the curse of dimensionality.

A linear function results in a constant dual or slope variable v. Since our resource vector is limited by the conservation of resources, Eq. (2-1b), the dual variable v_{it} represents the marginal additional resource and is given by:

$$\hat{v}_{it} = \frac{\partial \hat{V}_t\left(R_t\right)}{\partial R_{it}} \tag{3-13}$$

where $\hat{V}_t\left(R_t, D_t\right)$ is our objective or value function given in Eq. (3-9). In our problem, we employ $\hat{v}_{it}$ as part of the gradient of our resource vector R_t and build a separable linear approximation of the value function:

$$V(R_t) = \hat{V}_t(R_t) = \sum_{i \in \mathcal{I}} \hat{v}_{it} R_{it} \tag{3-14}$$

We build an estimate for each dual variable $\bar{v}_{it} \in \mathcal{I}$. The update procedure is similar to that of a regular value function approximation $V(R)$, as Eq. (3-10). An estimated dual variable $\hat{v}_{it}^n$, computed at iteration n, can be obtained from Eq. (3-13). We use $\bar{v}_{it}^{n-1}$ to obtain the new value function $\bar{v}_{it}^n$ using the standard updating algorithm:

$$\bar{v}_{it}^n = \left(1 - \alpha_{n-1}\right) \bar{v}_{it}^{n-1} + \alpha_{n-1} \hat{v}_{it}^n \tag{3-15}$$

where α_{n-1} is the step-size or learning rate after $n-1$ iterations. $\bar{v}_{it}^n$ is viewed as the best estimate of the slope of node i at time t after iteration n. The slopes of all nodes are combined to form an estimate of the value function $\bar{V}^n(R_t)$, as in Eq. (3-14)

3-2-3 Piecewise-linear approximations

In many applications, we estimate the value of having a quantity R of a certain resource (where R is a scalar). We might want to know the value of R bananas in a warehouse or R machines in a factory. Generally, R can be continuous or discrete. In ride-hailing, we deal with integer resources, i.e., vehicles, and R is discrete. As common in fleet management applications, we assume $\hat{V}_t(R_t)$ is separable piecewise-linear concave in R [20, 50, 21]:

$$\hat{V}_t(R_t) = \sum_{i \in \mathcal{I}} \hat{v}_{it}(R_{it}) \tag{3-16}$$

Unlike with linear functions, we use sequence of numbers to describe the slopes of the value function, $\{\hat{v}_{it}(1), \hat{v}_{it}(2), \ldots, \hat{v}_{it}\left(|R|\right)\}$, where $\hat{v}_{it}(y)$ is the slope of $\hat{V}_{it}$ over $(y-1, y)$ and $|R|$ denotes the total fleet size. As y grows, the slope might become constant. We can limit the range of the piece-wise linear value function to R_{max} and use $v(R_{max})$ when $s > R_{max}$, rather than storing individual slopes for all possible quantities y. Since overcrowded regions are undesirable in our reposition problem, the chances are slim we would sample large quantities of y, i.e., $y = 0.5 \cdot |R|$. We can thus remove the unnecessary slopes and reduce the complexity of our value function.

The concavity of the value function dictates that the slopes should be monotonically decreasing in R [21], i.e., the marginal added value of an additional resource should decrease or remain constant. The bigger the quantity of a resource available, the less value one additional resource adds. We can denote this by:

$$\hat{v}(y) \geq \hat{v}(y+1) \quad \forall y = 1, \ldots, R_{max} \tag{3-17}$$

Given $\hat{v}^n(y)$, a sample realisation of the slope for a quantity of y resource at iteration n, we use the standard update rule:

$$\bar{v}^n(y) = (1 - \alpha_{n-1})\, \bar{v}^{n-1}(y) + \alpha_{n-1} \hat{v}^n(y) \tag{3-18}$$

After the update, our new estimate $\bar{v}(y)$ might violate the monotonicity property of Eq. (3-17). We present two strategies to overcome this issue: SPAR and CAVE

SPAR

The separable projective approximation routine (SPAR) [31, 50] works by averaging the slopes that violate the monoticity. We start with an initialised set of slopes:

$$\bar{v}^0_{it}(y) \quad y = 0, \ldots, R_{max} \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T} \tag{3-19}$$

At iteration n, we obtain a temporary values $\bar{z}^n(y)$ as follows:

$$\bar{z}^n(y) = \begin{cases} (1 - \alpha_{n-1})\, \bar{v}^{n-1}(y) + \alpha_{n-1} \hat{v}^n, & y = R^n_{it} \\ \bar{v}^{n-1}(y) & \text{otherwise} \end{cases} \tag{3-20}$$

We then check the monoticity of the temporary values. If $\bar{z}^n(y+1) \geq z^n(y)$ for all y then no additional measures are required. If not, either $\bar{z}^n(R_{it}) < \bar{z}^n(R_{it}+1)$ or $\bar{z}^n(R_{it}) < \bar{z}^n(R_{it}-1)$. Either violation is fixed by solving following projection problem:

$$\min_v \|v - \bar{z}^n\|^2 \tag{3-21a}$$

subject to

$$v(z+1) - v(z) \leq 0 \tag{3-21b}$$

This projection can be solved easily. We first check which side violates the monotonicity of the value function. We have to update this cell and check whether we now have a concave function. If not, we repeat the process until we reach the end or we get a concave function. For a violation on the left side $\bar{z}^n(R_{it}) < \bar{z}^n(R_{it} - 1)$, we want to find the largest $1 < s \leq R_{it}$ such that:

$$\bar{z}^n(s-1) \geq \frac{1}{R_{it} - s + 1} \sum_{y=s}^{R_{it}} \bar{z}^n(y) \tag{3-22}$$

If such s cannot be found, we set $s = 1$. Then, we calculate:

$$c_j = \frac{1}{R_{it} - j + 1} \sum_{y=j}^{R_{it}} \bar{z}(y)^n \quad j = s, \ldots, R_{it} \tag{3-23}$$

and set

$$v_j^n = \min(v_{max}, c_j) \tag{3-24}$$

where v_{max} is an optional parameter which is set as the upper bound of the slopes v.

For the right handside $\bar{z}^n(R_{it}) < \bar{z}^n(R_{it}+1)$, we flip Eq. (3-22) and want to find $R_{it} < s \leq R_{max}$ such that:

$$\bar{z}^n(s+1) \leq \frac{1}{s - R_{it} + 1} \sum_{y=R_{it}}^{s} \bar{z}^n(y) \tag{3-25}$$

and set:

$$v_j^n = \max(v_{min}, d_j), \quad j = s, \ldots, R_{max} \tag{3-26}$$

where d_j denotes the equivalent of c_j for (3-25). v_{min} poses as a lower bound for the slopes v. Setting bounds on the slopes with v_{max} and v_{min} is optional. In some cases, it can attenuate the effect of outliers on the value function.

CAVE

The CAVE algorithm is an alteration of SPAR meant to accelerate learning. Instead of updating the slopes of just $y = R_{it}$, we update over a larger interval:

$$\bar{z}^n(y) = \begin{cases} (1-\alpha_{n-1})\,\bar{v}^{n-1}(y) + \alpha_{n-1}\hat{v}^n, & R_{it}^n - \delta^n \leq y \geq R_{it}^n + \delta^n \\ \bar{v}^{n-1}(y) & \text{otherwise} \end{cases} \tag{3-27}$$

where δ^n is a parameter that determines the width of the update interval. Before each update, δ^n is rounded to the nearest integer. The strategy is to reduce δ as we improve our approximation of the value function. Typically, δ^0 takes a range between 0.2 and 0.5 of R_{max}.

We sample slope $\hat{v}^n$ and use the same logic to maintain the monotonicity as with SPAR, solving the projection problem Eq. (3-21). We start with an interval $R^n \pm \delta_0$, which we progressively reduce. Since no general rule exists for reducing δ, Powell [31] suggests tracking one of the performance indicators of the learning algorithm, i.e., the objective function:

$$\delta^n = \begin{cases} \delta^{n-1} & \text{if } F^n \geq (1-\epsilon)\frac{\sum_1^K F^{n-i}}{K} \wedge k \geq K \\ \max\{\delta_{min}, 0.75\delta^{n-1}\} & \text{otherwise.} \end{cases} \tag{3-28}$$

where

$$k = \begin{cases} k+1 & \text{if } \delta^n = \delta^{n-1} \\ 0 & \text{otherwise.} \end{cases} \tag{3-29}$$

F^n denotes the objective function after n iterations, as described in Eq. (2-8), ϵ is used to allow a small perturbation from the mean objective over the last K iterations. We limit the

rapid decrease of δ with a count variable k. Furthermore, we can keep a minimal interval δ_{min}. If $\delta_{min} < 0.5$, CAVE is just SPAR after a set number of iterations.

These approximation methods work well in all kinds of resource allocation problems, as long as the attribute vector is relatively simple. Both SPAR and CAVE can scale to pretty large problems with hundreds or thousands of nodes. Furthermore, we do not have to solve the exploration-exploitation dilemma [31]. The concavity of the value function approximations pushes sub-optimal solutions towards the correct solution. So we can use a pure exploitation strategy.

3-2-4 Artificial Duals

For piecewise-linear approximations, we need to perform additional work to obtain the slopes $\hat{v}_t$ of our approximation. While methods for obtaining marginal value in mixed integer programs exist [51, 52], these methods are not readily available. We, therefore, use our knowledge of the ride-hailing process to obtain artificial slopes and avoid computing numerical slopes through re-optimisation.

Backlogged trips

In the approximation of the value function $\hat{V}_t$, we utilize the post-decision deterministic resource vector and assumed the post-demand resource demand D_t^x to be empty. Generally, we do not compute the full gradient, Eq. (3-11), but only consider the partial derivatives v_{it} where $R_{it}^x > 0$, i.e., the post-decision states where we have resources available.

We propose to exploit the post-decision demand vector D_t^x to accelerate learning. The strategy is to additionally update the value function for the non-empty post-decision vector states D_{ijt}^x, which is equal to the backlogged trips x_{ijt}^d. We assume that an additional vehicle present at the origin of a backlogged trip would serve this trip. In the case of multiple backlogged trips in a single region i, we assume the additional car would serve the highest revenue customer. We denote these slopes as $\hat{v}_{it}^a$ and use the potential revenue of backlogged trips:

$$\hat{v}_{it}^a = \max c_{ijt}^p \quad \in \{j, x_{ijt}^d > 0, w \leq W\} \tag{3-30}$$

where $\max c_{ijt}^p$ denotes the maximum revenue from all backlogged trips x_{ijt}^d with origin i. We introduce an additional parameter W, which limits the number of artificial duals from a consecutively backlogged trip. We explore the effect of W on learning in Chapter 5.

Since the revenue of each trip is readily available, the artificial duals offer a computationally efficient alternative to re-optimisation of the model. Furthermore, they will accelerate the learning through additional slope updates of the nodes with backlogged trips. Rather than wait for the fleet to explore missed trip nodes, we force value function updates in regions with backlogged trips.

Value of staying

While AMOD and repositioning mean to improve the utilisation rate, i.e., the fraction of time that vehicles are transporting customers, the time spent idle in MOD networks is still

significant. Spieser et al. [38] show in four different potential AMOD deployments that the utilisation rate never goes over 50% for their minimum fleet size. The utilisation rate naturally drops further when considering larger fleet sizes. We exploit this idle behaviour to obtain more updates of the value function.

As previously stated, in ride-hailing, the additional value of a vehicle diminishes greatly once we reach a certain quantity, i.e., y. For example, if a region contains four idle vehicles after optimisation, what would be the most likely action of an additional vehicle in this region?

Additionally to artificial duals from missed trips, we plan to decrease the computational burden of the learning process by approximating the value of staying. We assume that an additional vehicle in a region with a quantity of Y idle vehicles will also remain idle. For each node i, we equate the number of staying decisions x^r_{iit} and the threshold parameter Y, if $x^r_{itt} > Y$, we assume that the slope $\hat{v}_{it} = \hat{v}^s_{it}$. We compute the value of staying $\hat{v}^s_{it}$ as:

$$\hat{v}^s_{it} = c^r_{iit} + \bar{v}^{n-1}_{it}(y+1) \tag{3-31}$$

c^r_{itt} denotes the cost of staying idle as per Eq. (2-5) and $\bar{v}^{n-1}_{it}(y+1)$ denotes the value of having one additional resource at this node at the post-decision resource vector R^x_t after $n-1$ iterations

We explore the effect of both artificial dual strategies in Section 5-2.

3-3 A multi-period value function

Multi-period travel times have a significant impact on the structure of our value function approximation. We define τ_{max} as the maximum travel time within our network. Since each decision now produces a post-decision state up to $t + \tau_{max}$, far-sighted decisions become a problem. We highlight the issue of far sight in the illustrated example of Fig. 3-1.

Fig. 3-1(a) presents the current status of a fleet spread over nodes 1 and 4 at time step t and the current value of the slopes for the next four consecutive time steps. For this example network, we attempt to solve Eq. (3-14). Both vehicles can either stay at the current node or reposition to one of the other three nodes. Fig. 3-1(b) maps the decisions and their respective post-decision values. Based on the value functions, the best decision would be let the cars switch positions. This logic is what we call far sight. Since it takes 4 time steps for the car at node 1 to reach node 4, we can sample the post-decision value of node 4 at time step $t+4$ from node 1. While from node 4, we can only sample the value of time step $t+1$. This characteristic is undesirable since it promotes redundant repositioning. From an operational perspective, it is more logical to command both vehicles to stay at their current node.

To avoid far-sight, the literature has proposed several solutions. In [20, 53], an additional attribute is added to account for the arrival time. The value gets an additional dimension t' for each time within t and $t + \tau_{max}$ similar to the resource state $R_{t',t}$. We formulate this value function as:

$$V_t(R_t) = \sum_{i \in \mathcal{I}} \sum_{t'=t}^{t+\tau_{max}-1} \bar{V}_{it't}(R_{it't}) \tag{3-32}$$

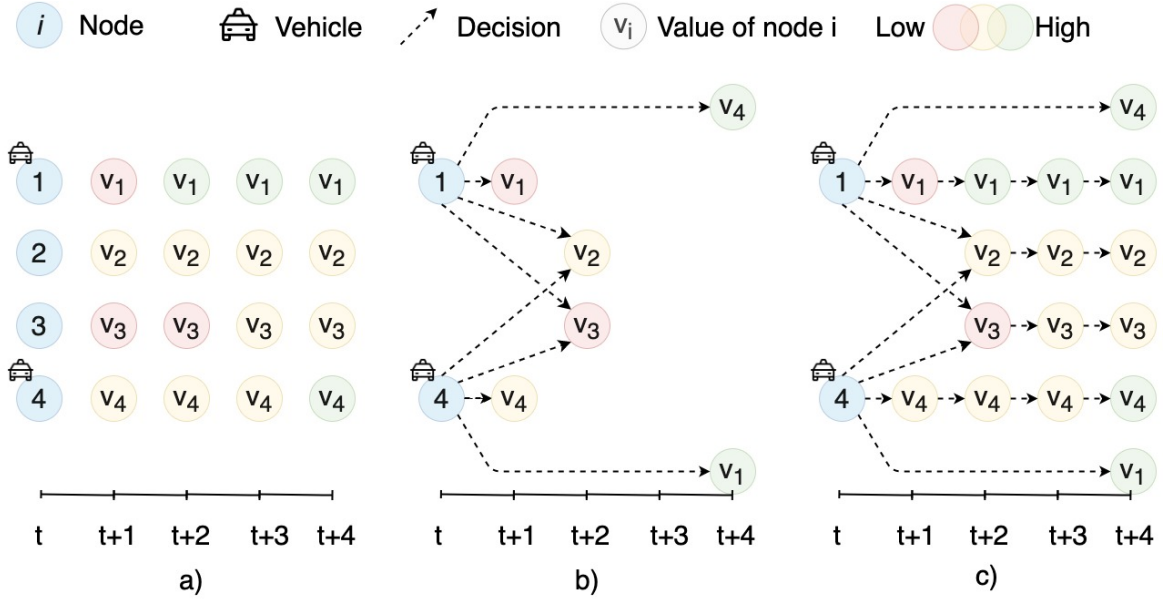


Figure 3-1: An illustrative example of the far-sight issue. a) Gives a sketch of the state at time t and the value function between $t = 1$ and $t+4$. b) The conventional method to find the slopes for the post-decision state, as Eq. (3-9). c) Updated multi-period value function with the additional samples within the horizon, as in Eq. (3-34)

Due to the additional attribute of the value function updates will become more sparse and thus learning rate will generally decrease. To avoid having to add an attribute in their value function, Beirigo et al. [18] address far sight by damping the value of repositioning by the opportunity cost of staying still. In the example of Fig. 3-1(b), for repositioning from node 1 to node 4, we get:

$$\hat{v}_{4,t+4} = \bar{v}_{4,t+4} - \sum_{t''=t+1}^{t+3} \bar{v}_{1,t''} \tag{3-33}$$

Inspired by the above methods, we propose to define a horizon parameter H^V which limits the length of the repositioning tasks. The motivation for the horizon H^V is specific to repositioning problems. Our value function approximation is primarily a tool for repositioning, i.e., trip decisions should always be prioritized over repositioning decisions. Once all trip decisions are fulfilled, we consider repositioning for the remainder of the fleet. We limit the maximum length of repositioning trips to H^V and sample the value functions of post-decision states that fall within this horizon H^V. Let define t^x as the post-decision time of post-decision sate R_t^x. We formulate a multi-period implementation of the value function as:

$$V_t(R_t) = \sum_{i \in \mathcal{I}} \sum_{t''=t^x}^{t+H^V} \gamma^{(t''-t)} \bar{v}_{it^x}(R_t^x) \tag{3-34}$$

where γ denotes the discount factor.

The approximation of Eq. (3-34) comes with one key assumption, to sample the slopes v_{it^x} within the horizon, we assume that all vehicles remain idle after finishing their tasks.

In Fig. 3-1(c), we set $H^V = 4$ and sample all slopes from the post-decision state up to the horizon H^V. The example highlights a key characteristic of the multi-period value function. Staying idle now samples a total of four slopes instead of just one, as in Fig. 3-1(b). While the longest repositioning task still samples a single value in Fig. 3-1(b-c). If most slopes v are positive, short decisions are favoured over long ones.

To further dampen the slopes at the end of the horizon, we introduce a discount factor γ. It incrementally discounts the values of vehicles relative to their respective post-decision step and steps hereafter within the horizon. For example, with $\gamma = 0.9$ the repositioning action from node 1 to 4 in Fig. 3-1(c) is discounted $0.9^4 = 0.6561$. While the decision to stay at node 1 has a discount of 0.9 for the first slope and $0.9^2 = 0.81$ for the second slope. The discount factor is a lever for the value function to find the right balance between the emphasis on the number of slopes versus the emphasis on the estimates of the slopes. We evaluate the influence of the horizon H^V and discount factor γ on repositioning in Chapter 5

3-4 Summary

In this chapter, we gave a brief introduction to ADP. We introduced the concept of value functions and how they can be useful for decision-making. We presented two methods for value function approximation based on linear and piecewise-linear functions. A key insight for resource allocation was to estimate the derivate of the value function instead of the value function itself. We give a brief summary of two algorithms for solving sequential decision-making problems in Algorithm 1 and Algorithm 2 based on linear and piecewise-linear approximation respectively.

Lastly, we elaborated on the implication of multi-period travel times for the value function. We switch from an approximation based on the post-decision state to an multi-period value function with a value horizon. Within the horizon, cars are assumed to remain idle upon finishing their respective task.

Algorithm 1: Linear approximate value iteration (LAVI)

Require: $\bar{V}^0, \alpha, N, T$

Initialise: Value function $\bar{V}_t^0 \,, \forall t \in \mathcal{T} = 0, 1, \ldots, T$
for $n = 1, \ldots, N$ **do**
  Sample demand $\hat{D}^n$ and initialize resources R_0
  for $t = 0, 1, \ldots, T$ **do**
    Solve optimisation problem Eq. (3-9) and obtain x_t
    for $R_{it}^n > 0 \in R_t$ **do**
      Extract the dual variable $\hat{v}_{it}^n$ corresponding to the resource conservation constraint Eq. (2-1b).
      Update the value function $\bar{v}_t^n$ using Eq. (3-15)
    end for
    Execute x_t into Eq. (2-2) and Eq. (2-3) to obtain R_{t+1} and D_{t+1}
  end for
end for
Return the value functions $\{\bar{v}_{it}^n | \forall t \in \mathcal{T} \,, \forall i \in \mathcal{I}\}$

Algorithm 2: Piecewise-linear approximate value iteration (PLAVI)

Require: $\bar{V^0}, \alpha, N, T, (W), (v_{max}), (v_{min}), (Y), (R_{max}), (K), (\delta_0)$

Initialise: Value function $\{\bar{V}(y)_t^0, \forall t \in \mathcal{T} = \{0, 1, \ldots, T\}, y = \{0, \ldots, R_{max}\}$
for $n = 1, \ldots, N$ **do**
  Sample demand $\hat{D}^n$ and initialise resources R_0
  for $t = 0, 1, \ldots, T$ **do**
    Solve optimisation problem Eq. (3-9) and obtain x_t
    if using Y: **then**
      for $R_{it}^n > 0 \in R_t$ **do**
        if $R_{it}^n \geq Y$ **then**
          assume $\hat{v}_{it}^n = \hat{v}_{it}^s$ using Eq. (3-31)
          Update the value function $\bar{v}_{it}^n$ using Eq. (3-18)
          if Eq. (3-17) is violated **then**
            SPAR or CAVE
          end if
        end if
      end if
    end for
    if using W: **then**
      for for trip b in D_t^x **do**
        if $w \leq W$ **then**
          get regional centre i of trip origin b_1
          get v_{it}^a from Eq. (3-30)
          Update the value function $\bar{v}_{it}^n$ using Eq. (3-18)
          if Eq. (3-17) is violated **then**
            SPAR or CAVE
          end if
        end if
      end for
    end if
    Execute x_t into Eq. (2-2) and Eq. (2-3) to obtain R_{t+1} and D_{t+1}
  end for
end for
Return the value functions $\{\bar{v}_{it}^n | \forall t \in \mathcal{T}, \forall i \in \mathcal{I}\}$

Chapter 4

Learning-based MPC

MPC or receding horizon control is a state-of-the-art model-based control method. It combines three main elements: a predictive model, optimisation over a receding horizon and feedback correction. It has been studied extensively over the years and widely applied in different industrial control applications [54]. We first give a brief introduction to MPC and present an MPC algorithm for the MOD problem formulated in Chapter 2. Lastly, we present a hybrid framework that incorporates a value function in the MPC framework to ease its computational demand.

4-1 Model predictive control

The concept behind MPC is quite straightforward, given a sequential decision-making problem, i.e., the optimal rebalancing problem of Eq. (2-9), we solve an MPC problem over a predetermined horizon. From a control perspective, we formulate an MPC problem as:

$$\min_{U,X} \sum_{k}^{H} J(\mathbf{x}(k), \mathbf{u}(k)) \tag{4-1}$$

s.t.

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k)) \tag{4-2}$$

$$U \in \mathcal{U} \tag{4-3}$$

$$X \in \mathcal{X} \tag{4-4}$$

$$\tag{4-5}$$

where J denotes the cost function, $\mathbf{x}(k+1) = f(\mathbf{x}, u)$ is the equivalent of a transition function, which describes the system dynamics over time, i.e., Eqs. (2-2) and (2-3). H denotes the horizon, $U = \{u_k, \dots, u_H\}$ denotes the control sequence and $X = \{\mathbf{x}_k, \dots, \mathbf{x}_H\}$ denotes the evolution of states. We acknowledge the resemblance to the language of dynamic resource management for the state $\mathbf{x}_k$ and S_t, and input $\mathbf{u}_k^*$ and decision x_t. For the rest of this section, we use the notation of control theory to expand on the characteristics of the MPC framework.

The control community focuses on steering the system dynamics to an equilibrium and have it operate in steady state. MPC works in real-time. At time step k, the initial state $\mathbf{x}_k$ is observed. Using the discrete system model Eq. (4-3), we predict how the state evolves given any set of inputs U. Optimisation of the cost function J results in an optimal control sequence U^* of which only the first control input $\mathbf{u}_k$ will be applied to the system. After which, the optimisation procedure is repeated at the next time step $k+1$. The process can be repeated indefinitely and is very flexible in design. Limitations on the system can be implemented by the use of constraints.

Constraints are one of the main features that differentiates MPC from other optimal control methods [55]. The control U and state X sequences should be contained within the admissible in the feasible sets $\mathcal{U}$ and $\mathcal{X}$. We can use additional input and state constraints to account for other limitations of the system. For example, if the decision variable is the number of vehicles, only non-negative integer inputs can be considered, i.e., $\mathbf{u} \in \mathbb{Z}^+$. These input constraints have to be considered for all steps within the horizon.

One of the features of MPC is the forecasting horizon. The real-time forecasting horizon is dependent on the horizon parameter H and sampling time Δt, the time between discrete states k and $k+1$. A good estimate for the sampling time is related to the bandwidth of the dynamical system [56]. The horizon parameter H determines the range of the forecast. As we select a longer horizon, the prediction is more prone to model errors. Each step we move further along the horizon errors accumulate.

The goal of the controller is expressed as the objective function. It is directly dependent on the input and state sequence and indirectly through the constraints. A regulation MPC uses a cost function to keep the state around the desired equilibrium point. A classic example of MPC uses a quadratic cost function $J(X,U)$:

$$J(X,U) = \sum^{H} X^T Q X + U^T R U \tag{4-6}$$

with $Q \succeq 0$ and $R \succ 0$. The matrices Q and R are the tuning parameters of the controller. The first term of the quadratic cost function emphasises deviations of the state from the equilibrium and we can emphasise the cost of each state in the horizon using the weight matrix Q. To clarify, using an identity matrix for Q gives the same cost for all states within the horizon. Large values of Q in comparison to R drive the state to the equilibrium quickly at the expense of large control action [55]. Choosing appropriate values for weight matrices is not always obvious and it is one of the challenges in the design of MPC.

The MPC framework works in real-time and is repeated at every time step. This type of online optimisation has limitations. Naturally, the solving time of the problem should be much smaller than the sampling time Δt. Otherwise, the controller is not realisable in real-time. The complexity of the problem increases as we consider large-scale applications resulting in more states, inputs and constraints. These challenges in online optimisation provide an incentive to use aggregated models for large-scale applications.

4-1-1 MPC in MOD

Within MOD literature,MPC has been studied extensively in recent years [8, 9, 15, 16, 43, 57, 58, 59]. This section introduce an MPC for MOD.

For an arbitrary state $S_t = (R_t, D_t)$, we can formulate an MPC optimisation problem for a single period travel MOD model as:

$$\max_x \sum_t^{t+H} C(x_t) = \sum_t^{t+H} \sum_{i,j \in \mathcal{I}} \left(c^p_{ijt} x^p_{ijt} - c^d_{ijt} x^d_{ijt} - c^r_{ijt} x^r_{ijt} \right) \tag{4-7a}$$

subject to

$$\sum_{i \in \mathcal{I}} \left(x^p_{ijt} + x^r_{ijt} \right) = R_{it} \quad , \forall i \in \mathcal{I} \tag{4-7b}$$

$$\sum_{j \in \mathcal{I}} \left(x^p_{ji,t-1} + x^r_{ji,t-1} \right) + \sum_{i \in \mathcal{I}} \left(x^p_{ijt} + x^r_{ijt} \right) = 0 \quad \forall i \in I \, , t \in \{t+1, \ldots, t+H\} \tag{4-7c}$$

$$\sum x^p_{ijt} + x^d_{ijt} = D_{ijt} \quad \forall i \in \mathcal{I} \, , \forall j \in \mathcal{I} \, , \tag{4-7d}$$

$$\sum x^p_{ijt} + x^d_{ijt} = \hat{D}_{ij,t+1} + d_{ij,t+1} \quad \forall i \in \mathcal{I} \, , \forall j \in \mathcal{I} \, , t \in \{t+1, \ldots, t+H\} \tag{4-7e}$$

$$x^r_{ijt} \, , x^p_{ijt} \, x^d_{ijt} \quad \in Z_+ \quad \forall i \in \mathcal{I} \, , \forall j \in \mathcal{I} \, , \forall t \in \mathcal{T} , \tag{4-7f}$$

where Eq. (4-7b) denotes the conservation of resources for the first step. Eq. (4-7c) is derived from the transition function of Eq. (2-2) and denotes the dynamics of resources within the horizon. Eq. (4-7d) ensures the current demand is serviced or backlogged. Eq. (4-7e) denotes the dynamics of trip request within the horizon. Lastly, Eq. (4-7f) ensures that all decision variables are integer.

The optimisation requires the demand realisation of all future time steps within the horizon, i.e., $\{\hat{D}_{t+1}, \ldots, \hat{D}_{t+H}\}$. In ride-haling, this demand is normally not readily available and has to be forecasted. This is one of the severe limitations of deterministic MPC. A prediction is never perfect and can induce model errors. Naturally, it is easier to accurately predict what may happen in the near future. However, even small errors in the earlier states of the horizon can have a significant impact on decision-making and performance.

Within MPC, it can be good practice to discount states within the horizon of MPC. Near-term decisions are assigned with a higher value than decisions in the long term. However, within ride-hailing, there is no consensus on using a explicit discount parameter γ, as it is used in [43, 58] but not used in [8, 9]. One of the methods MOD implicitly incorporates discounts is within the cost functions. In Eq. (2-4), the revenue of a trip is dependent on the delay and thus to maximise profit should be serviced as soon as possible.

Literature has provided several methods on demand passenger prediction which range from simple probabilistic methods to artificial neural networks. A time-invariant Poisson process with a constant customer arrival rate is fairly common [44, 60, 57]. Oda and Joe-Wong [43] rely on a convolutional neural network to forecast ride requests, while Miao et al. [59] used probability functions based on historic and current demand to predict future ride-request. In [8, 9], the authors predicted the customer arrival rate as a spatial and time variant Poisson process with a long short-term memory (LSTM) neural network.

Within the literature, the common strategy to reduce computational complexity is to have different matching and repositioning frequencies. MPC is then purely used as a vehicle repositioning strategy [9, 8, 58, 16]. The MPC issues repositioning tasks over a long horizon. The repositioning tasks are assigned to vehicles that become idle within the repositioning interval. At the new interval, the unused tasks are discarded and new repositioning tasks are computed for the next interval.

4-2 Combining ADP and MPC

MPC is a powerful tool for decision-making. It approximates the solution of an infinite-horizon optimal control problem by solving a finite-horizon optimal control problem in a receding horizon fashion [61]. The potential of combined learning and prediction is an active field of research [62].

To facilitate the combination of ADP and MPC, we take the approach of most ADP literature and consider repositioning at every sampling interval. In most repositioning problems, rebalancing operations are expected to only last a single period of time, i.e., at each time interval all vehicles are either servicing a customer or idle [37, 36, 19]. The single period assumption imposes a strict constraint on the granularity of the network. Furthermore, it is computationally expensive to find the optimal decision for all idle vehicles at every time interval. The lower the resolution of the underlying map, the fewer locations are available, and the more multi-period travel times can be expected between location pairs. Multi-period travel times can reduce the computational complexity of MPC by reducing the number of optimisation variables and states.

Another downside of multi-period travel times comes from modelling and forecasting errors. The decisions in ride-hailing are integers and take multiple periods to complete. A repositioning action can now prohibit a vehicle from servicing requests for multiple time steps. To overcome this issue, repositioning vehicles should ideally still be available for trip decisions. In this thesis, we have a perfect forecasting model and no model errors can occur. Therefore, vehicles cannot be interrupted while repositioning, which helps to further reduce the number of decision variables in the MPC.

A potential limitation of MPC, especially in high dimensional problems, is short-sightedness due to a limited prediction horizon. In rebalancing problems, the repositioning tasks can take multiple periods to complete. For MPC to have solid performance, all nodes should be reachable within the prediction horizon. If a node cannot be reached by any other node, the MPC is unable to reposition cars to and away from this node. Cars can consequently get stuck at such nodes. Unable to escape from this node except through ride requests. These nodes are undesirable and preemptive network analysis can help determine an adequate horizon. However, if we could obtain information beyond the horizon we could overcome this requirement. Adding information from beyond the horizon could help guide decision-making. A final or terminal cost approximating the infinite-horizon cost, or cost-to-go can mitigate this short-sightedness. If we draw a parallel between the Bellman equation and MPC framework we can see that we can use a final cost of MPC to approximate the infinite-horizon cost.

We consider a predictive horizon H and cast the Bellman Eq. (3-7) into a multi-period MPC framework:

$$\max_{x} \sum_{s=t\in\mathcal{T}}^{t+H} \sum_{i,j\in\mathcal{I}} \left(c^p_{ijs} x^p_{ijs} - c^d_{ijs} x^d_{ijs} - c^r_{ijs} x^r_{ijs} \right) + \bar{V}_{t+H} \left(R^x_{t+H} \right) \tag{4-8a}$$

subject to

$$\sum_{i\in\mathcal{I}} \left(x^p_{ijs} + x^r_{ijs} \right) = R_{iss} \quad \forall i \in \mathcal{I}, t \leq s \leq H \tag{4-8b}$$

$$\sum_{i\in\mathcal{I}} \left(\mathbf{1}_{\tau_{ij}} \left(t' - s \right) x^r_{ijs} + \mathbf{1}_{\tau_{ij}} \left(t' - s \right) x^p_{ijs} \right) + R_{jt's} = R_{jt',s+1} \tag{4-8c}$$

$$\forall i \in \mathcal{I}, t \leq s \leq t + H, s + 1 \leq t' \leq s + H^V$$

$$\sum x^p_{ijt} + x^d_{ijt} = D_{ijt} \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{I} \tag{4-8d}$$

$$\sum x^p_{ijs} + x^d_{ijs} = \hat{D}_{ij,s+1} + d_{ij,s+1} \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{I}, t + 1 \leq s \leq t + H \tag{4-8e}$$

$$x^r_{ijs}, x^p_{ijs} x^d_{ijs} \quad \in Z_+ \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{I}, t \leq s \leq t + H \tag{4-8f}$$

Where Eqs. (4-8b) and (4-8c) dictate the conservation of resources as per Eq. (2-10). Eqs. (4-8d) and (4-8e) denote the conservation of passengers as per Eq. (2-3). Finally, Eq. (4-8f) ensures all decision variables are integers.

The value function $\bar{V}_{t+H}$ is derived from Eq. (3-34). We use either algorithm 1 or 2 to find an approximation of the value function. In our hybrid algorithm, we use the the combined horizon of H and H^V as the upper bound of the length of repositioning tasks, i.e., $\tau^r_{max} = H + H^V$.

The integration of the terminal cost can benefit the performance since the MPC now includes information from beyond the predictive horizon in the optimisation. Furthermore, to reduce the computational burden of the MPC by shortening the prediction horizon and incorporating $\bar{V}$ to compensate the possible performance reduction cause by shortened predictive horizon. A reduction in complexity of the online optimisation would allow us to consider a more detailed environment or system model. The value function will be used as a long-term planner while the MPC captures the arrival of customers and the availability of drivers within the near future. We present possible uses for this type of hybrid algorithm in Chapter 5

4-3 Summary

In this chapter, we introduced the general framework of MPC from a control perspective. We shortly cover the use of MPC in MOD. Lastly, we introduced a hybrid MPC and ADP algorithm. The goal of this hybrid algorithm is to reduce the predictive horizon of MPC and use a value function as a long-term planner.

Chapter 5

Case study

In this chapter, we present three case study to evaluate the potential of combined ADP and MPC frameworks in MOD networks. Within the first two case studies, we validate our algorithms on a small benchmark network. The first case study focuses on approximation of the value function within our MOD model. In the second case study, we integrate a value function in to the MPC framework. We test and compare ADP,MPC and combined ADP-MPC strategies. In the last case study, we enlarge the network and explore how ADP-MPC strategies can reduce the computational burden of matching and repositioning in a large scale MOD network.

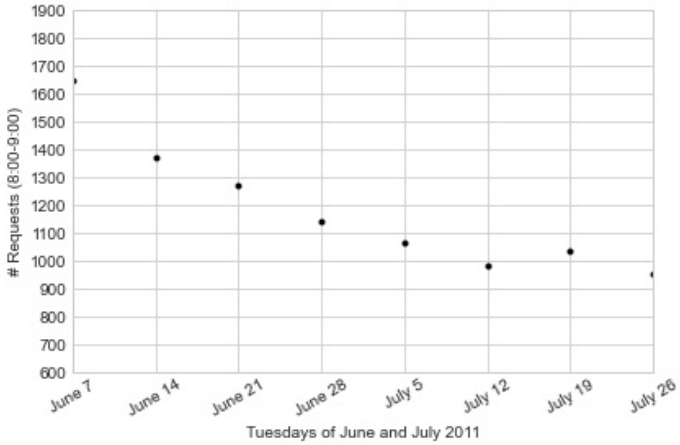
5-1 Set up

Before we showcase the algorithms, we provide some details on the network setup, the used data sets, fleet configuration, cost parameters and performance indicators.

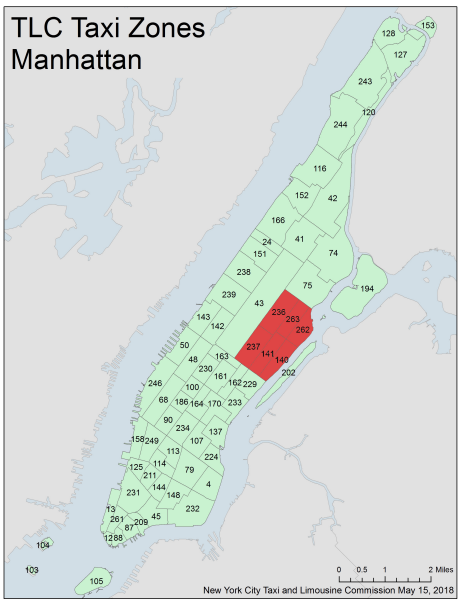
Network For case study A & B, we consider a subset of the taxi zones of Manhattan, labeled in red in Fig. 5-1b. The street network consist of the graph $G(\mathcal{N}, \mathcal{E})$, with $|\mathcal{N}| = 515$ nodes and $|\mathcal{E}| = 891$ edges. We computed the two minute, regional centres for this network, as described in Section 2-2. Our aggregate network consists of 14 virtual nodes. We compute the corresponding travel times with a sampling time of $\Delta t = 30s$ and summarised them into Table B-1. From this origin-destination (OD) matrix, we observer that a minimal horizon of 9 is required for the network to not have a repositioning sink, i.e., a node that cannot be reached by any other nodes through repositioning.

For case study C, we enlarge the network. It now covers the entirety of Manhattan Island. We include all taxi zones of Fig. 5-1b, excluding zones 153, 194, 202, 105, 104 and 103. The street network consist of the graph $G_c(\mathcal{N}_c, \mathcal{E}_c)$, with $|\mathcal{N}_c| = 6248$ nodes and $|\mathcal{E}_c| = 11375$ edges. We learn value functions for the two minute regional centres, consisting of 192 total nodes. On average, each region centre covers about 32 nodes.

Data sets
For training, we sample trips from June 28th 2011 with a resize factor of 0.2. We simulate



(a) The number of trips of each Tuesday in June and July 2011 for the benchmark network.



(b) TLC taxi zones of Manhattan. The zones of the benchmark simulation are marked in red.

Figure 5-1: Left to right, demand graph and the taxi zones of Manhattan.

over the period from 8:30 till 9:30 in the morning rush hour. For the benchmark network, we get approximately 110 to 160 trips per iteration, as seen in Fig. 5-1a. Individual trips are played back from the closest node in the street network graph G. We have a 5 minute repositioning period before trips start arriving and 10 minute termination period after the hour has passed. With a sampling rate of 30 seconds, we get a total simulation length of 150 steps.

In case study C, we enlarge the repositioning and termination offset to 10 and 15 min respectively. Resulting in a simulation length of 85 minutes and 170 steps. Furthermore, we reduce the resize factor of the demand to 0.1.

To validate our algorithm, we use the demand of the other Tuesdays in June and July. We evaluate the value function learned from a single day in other demand scenarios. We deliberately chose the 28th of June for training since it serves as a middle ground for the other demand scenarios, as seen in Fig. 5-1a.

Fleet configuration

To find an estimate of the appropriate fleet-size, we run the deterministic optimal rebalancing program as presented in Eq. (2-9a), as in [8]. This algorithm utilises a deterministic demand pattern to find the optimal fleet size given the given a network configuration. For our testing data and a $60s$ regional centre network, we find a range of 15 to 39 for the optimal fleet. We run additional tests on the two minute regional centre network to find an appropriate fleet size. The results are summarised in Fig. A-1. To fine tune our learning algorithm, we first consider a relatively small fleet size of 25, such that effect of the parameters tuning is more apparent. After which, we increase the fleet size to 40 to validate our findings.

For case study C, we again use the deterministic optimal rebalancing program. For our test data and the 120s regional centre network, we get a range from 280 to 409. To account for network aggregation during the optimal rebalancing program, we set the fleet size near the upper-bound, i.e., $|R| = 400$.

Cost parameters
We adapted the basic income for individual trips from the TLC data base [40] and adapted the cost parameters of Bosch et al. [63], which were consisted with previous cost analysis of autonomous shared mobility services. The parking cost are based on the annual cost of ownership presented within [63]. We present all the relevant cost parameters in Table 5-1.

Table 5-1: Table of scenario parameters for the case studies.

Parameter	Description	Value
p_{base}	Base fare	\$ 2.5
p_{time}	Time depended fare	\$ 1.56 / km
c_{time}	Operational cost of a vehicle	\$ 0.18 / km
c_{delay}	Penalty due to delay in pick-up time	\$ 0.1 / minute
c_{stay}	Cost of staying at the current position	\$ 0.0035 /min
$w_{\max}$	Maximum Delay	5 minutes
c_{reject}	Trip rejection cost	\$ 1
c_{delay}	Backlog cost	\$ 0.1 / min

Performance indicators

During the case studies, we measure the quality of the ride-hailing service through three main performance indicators

- The objective function F, as introduced in Section 2-2. This monetary indicator denotes the summed revenue of trips, the cost of backlogging and the cost of repositioning, as in Eq. (2-8).
- Service Rate, the percentage of total trips that receive service. After a delay of five minutes customers get rejected.
- Waiting time, the time customers have to wait before receiving service, measured in minutes.

Additionally, we are interested in the time spend on the different actions within the fleet:

- Servicing Passenger (S), the time spend transporting a passenger.
- Driving to pickup (D), the time spend in between trip request acceptance and pickup.
- Parked (P), the time spend idle.
- Repositioning (R), the time spend repositioning to a new regional centre.

Specifically, the correlation between the percentage of the total time spend on repositioning and the quality of service indicators, mentioned above, is of interest.

Finally, The goal of this work is to find a computationally efficient repositioning strategy. We, therefore take in account the time it takes to compute the iteration. This computation time is measured in seconds in Sections 5-2 and 5-3 and in minutes in Section 5-4.

5-2 Case study A: Linear and piece-wise linear approximations

In this first case study, we evaluate the approximation methods for learning a value function in MOD networks. We set our baseline parameters according to Table 5-2 and learn a value function for repositioning. We evaluate the problem in terms of the different learning parameters and approximation structures.

Table 5-2: Table of baseline learning parameters for case study A.

Parameter	Description	Baseline value
H^V	Value Horizon	9
γ	Discount factor	0.9
α	Learning rate	0.1
$\bar{V}_0$	Intial value function	0
v_{max}	Upperbound slope	5
v_{min}	Lowerbound slope	$-10 \cdot c_{stay}$
R_{max}	Maximum vehicles per node	5

5-2-1 Linear and piece-wise linear approximations

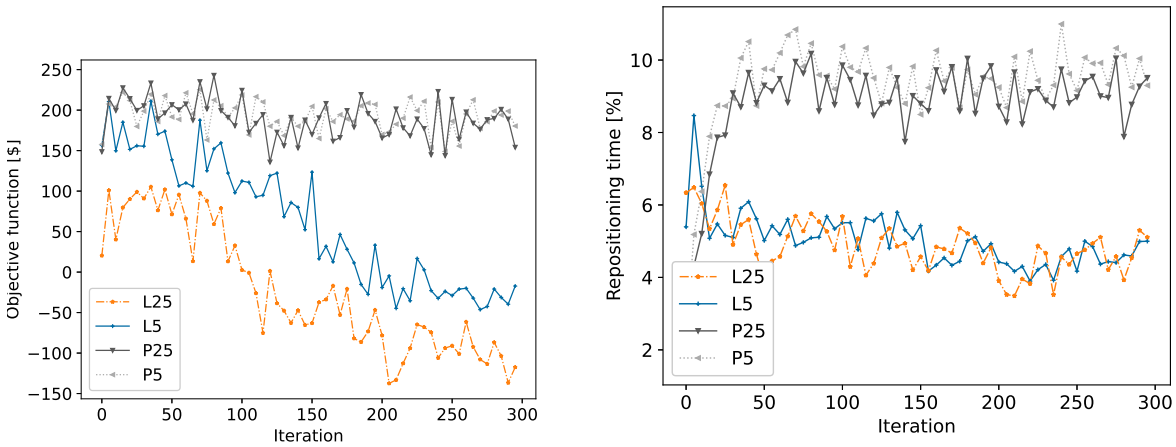
We first compare the different function approximation on the training set. While for LAVI, the dual variables are readily available, we want a fair comparison between the two approximation strategies. We use backlogged trip duals, with $W = 0$, in both approximations.

Additionally, we consider a strategy to avoid node congestion. Consider the domain of the piece-wise approximation, as in Eq. (3-17). We reduce the domain of y from the total fleet size $|R|$ to R_{max}. Since the slope of linear functions is constant, we have to introduce a method to avoid node congestion and add the following constraint to our LAVI algorithm:

$$R_{it} \leq R_{\max} \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T} \tag{5-1}$$

We showcase both linear (L) and piece-wise linear (P) approximation with and without congestion avoidance strategy, by choosing $R_{max} = 5$ and $R_{max} = 25$ respectively. We thus get a fourfold of strategies: $L5, L25, P5$ and $P25$, distinguishing the type of function approximator (L or P) and the value of R_{max} (5 or 25).

As per Fig. 5-2, linear approximations yield poor performance in this attribute space. The attribute space is one-dimensional and the slope is constant for different quantities of vehicles.



(a) The objective function for different configurations of $R_{\max}$.

(b) The percentage of total time spend repositioning for different configurations of $R_{\max}$

Figure 5-2: PLAVI and LAVI for different values of $R_{\max}$.

Due to constant slopes, vehicles move towards the highest value v_{it} in their vicinity and start to group up at nodes. Afterwards, they move in clusters through the network, as in Fig. A-2.

The clustering of vehicles results in a low service rate and overall poor service quality. The difference in objective function between $L5$ and $L25$ can be attributed to the limitation on the cluster size. For $L5$, the vehicles are widely available over the network and naturally service more passengers.

For PLAVI in Table 5-3, we see a slight increase in service rate by increasing the congestion strategies. Furthermore, from Fig. 5-2a, we note that the objective function is slightly higher on average for $P5$, which is supported by Table 5-3. $P5$ does issue more repositioning tasks on average compared to $P25$, see Fig. 5-2b and Table 5-3. However, the main difference between the two strategies is the reduced complexity of the value function, as is clear from the difference in computation time between P5 and P25 in Table 5-3.

As PLAVI shows the most potential as a repositioning strategy, we commit the rest of this case study to finding a parameter configuration that provides the best overall service quality.

	Objective Function($)	Service rate	Wait time (min)	Fleet status / total time				Comp. time (s)
				S	D	P	R	
L5	-30.0	52.6%	2.81	14.6%	3.6%	74.4%	7.4%	2.21
L25	-101.2	44.1%	2.93	12.0%	3.0%	77.4%	7.5%	2.37
P5	188.5	79.1%	2.39	21.1%	5.7%	57.7%	15.6%	2.34
P25	185.7	78.8%	2.43	21.0%	5.7%	58.7%	14.6%	2.71

Table 5-3: Average of the last 50 iterations of the trainingset for the objective function, service rate, computation time and different fleetstatus: Driving to pickup (D), Serving passenger (S), Parked (P) and Repositioning (R).

5-2-2 Parameters of PLAVI

This section is dedicated to exploring different configurations of learning parameters of the PLAVI algorithm. We first evaluate the effect of pure W and Y strategies, as explained increfsec: arti. Furthermore, we evaluate different configurations of the value update routines, SPAR and CAVE. Lastly, we consider the horizon and discount factor.

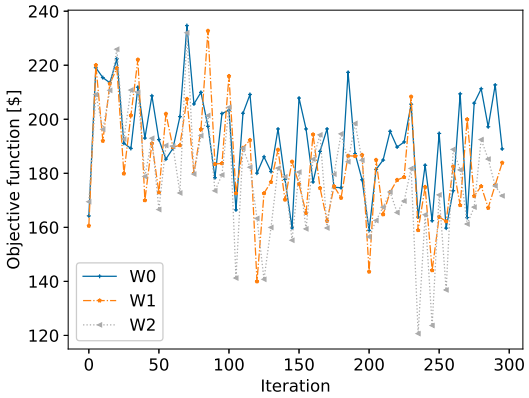
In general, we seek a parameter configuration that minimises repositioning and maximises the objective function. We consider the time spent repositioning to evaluate the stability of the learning. As initially seen in Fig. 5-2, the objective function reaches the optimal value within the first few iterations, while the number of repositioning tasks is still growing after about 50 iterations. Slow convergence rates lead to longer training and are generally undesirable if fast alternatives are available.

Learning rate

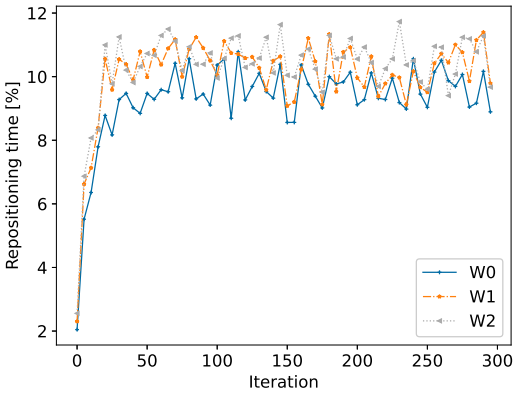
Learning rates can be regarded as a soft science of ADP [32]. In this work, we found a constant learning rate to work well. In Fig. A-3, we show the effect on the convergence rate of different constant learning rates. We keep the learning rate constant throughout all experiments and use $\alpha = 0.1$, as it is a common practice.

Backlogged Trips

First, we evaluate if backlogged trip duals can be used to generate an approximation of the value function. We denote the different strategies by $W0, W1, W2$, where $W1$ sets backlogged dual parameter $W = 0$. We maintain the monotonicity of the slopes using SPAR. The respective learning curves are given in Fig. 5-3.



(a) The objective function for different configurations of W.



(b) The percentage of total time spend repositioning for different configurations of W.

Figure 5-3: Learning curves of W.

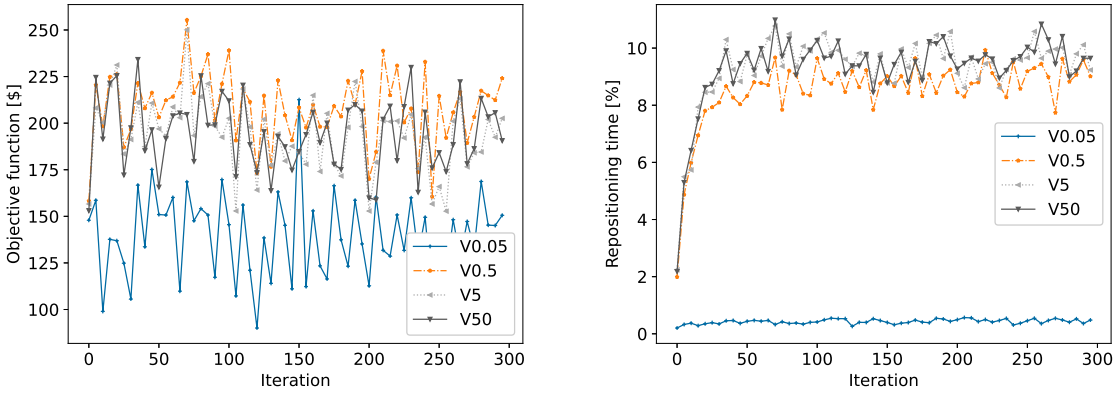
In this particular network, only the slope at quantity $y = 0$ gets updated, i.e., only at empty nodes trips get backlogged. The value function is empty except for quantity $y = 0$. The

algorithm sends vehicles to empty nodes. In order to sample values, only previously empty nodes can be visited during the entire horizon. The algorithm issues repositioning tasks to avoid exceeding this quantity.

Of the three strategies depicted in Fig. 5-3, $W0$ is superior in both metrics. This is logical since $W1$ and $W2$ generate duals for previously backlogged trips. All strategies incite rebalancing to nodes, where we previously experienced sub-optimal trip coverage. However, except for $W0$, the strategies also promote sub-optimal trip coverage with additional value updates. In an optimal situation, we do not backlog trips. Therefore, we consider strategies based on backlogged trips with no previous delay and hence $W0$, in the rest of the case studies.

Upper bound

We empirically set the upper bound of the slopes for the previous best strategy $W0$. Since we work with duals based on missed trip revenue, the values of the slopes have a natural upper bound of the highest revenue trip. We previously limited the slopes by $v_{max} = 5$ and now consider different upper bounds on a logarithmic scale. From Fig. 5-4, we observe that an upper-bound of $v_{max} = 0.5$ results in less repositioning and favourable performance. We therefore set $v_{max} = 0.5$ as a new upper-bound.



(a) The objective function for different configurations of V_{max}.

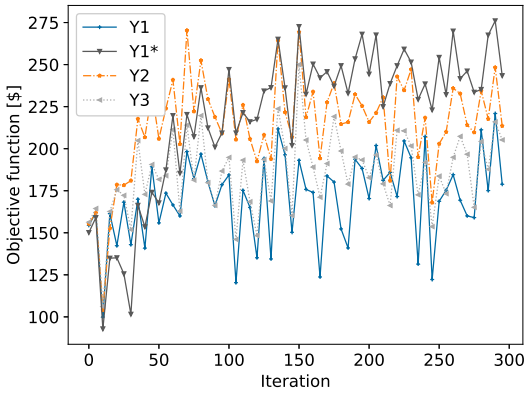
(b) The percentage of total time spend repositioning for different configurations of V_{max}.

Figure 5-4: Learning curves for different V_{max}.

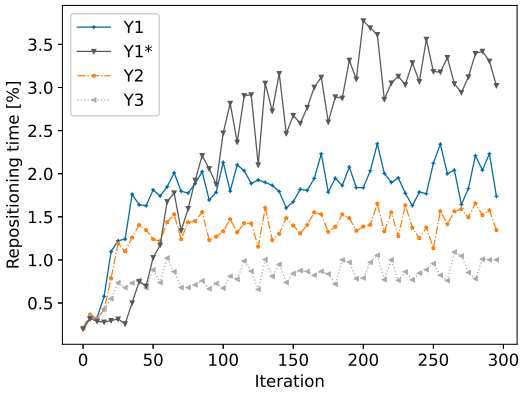
Value of staying

Next, we consider a strategy based on the value of staying. Once the quantity exceeds our threshold, i.e., $y > Y$, we assume that this vehicle will stay at this position and update the value of staying for this node. For node i and $Y1$, if $x^r_{iit} > 1$, meaning that there are at least two idle vehicles within region i at time t, we generate a value of staying dual $v^s(y)$ for quantity $y = R_{it}$, as in Eq. (3-31) If $x^r_{iit} \leq 1$, which means one or no idle vehicles are present within region i at time t, no duals are generated. We present four strategies, $Y1, Y1^*, Y2, Y3$

where $Y1^*$ denotes a special variant of $Y1$ where v_{min} has not been set and the value function is negatively unbounded.



(a) The objective function for different configurations of Y.



(b) The percentage of total time spend repositioning for different configurations of Y.

Figure 5-5: Learning curves of Y.

The value of staying pushes cars away instead of attracting them. It creates negative duals based on the cost of parking. Throughout the learning process, duals are generated at overcrowded nodes and the value of staying at crowded nodes declines. For the strategy $Y1$, all nodes get the desired quantity of 1. For $y > 1$, duals are generated and these make a quantity of $y > 1$ undesirable.

Strategy $Y1^*$ highlights what happens if the value function can grow negatively unbounded. The penalties of exceeding the desired quantity $y = 1$ grow unbounded. Therefore, the algorithm does not let cars idle at nodes and executes additional repositioning tasks to avoid value function sample of quantity $y > 1$, as highlighted in Fig. 5-5b. While it has the best performance at later iterations, the number of repositioning tasks is still growing after 300 iterations. It indicates that the slopes are still changing and thus we do not have a stable algorithm. More iterations could potentially lead to convergence, however the slopes could also keep growing negatively and convergence will never occur.

CAVE

We have considered the separate dual strategies of W and Y. Both suffer from only being able to update a part of the value function. We also use the alternative update routine CAVE, as described in Section 3-2-3. To maintain monotonicity, we solve the same projection problem, namely Eq. (3-21), with additional slope updates. CAVE updates the value function over a wider interval, which over the course of iterations declines to δ_{min}.

The main parameter of CAVE δ is linked to the maximum resource quantity, in this instance $R_{max} = 5$, we choose our starting interval to be $20, 40$ and 60% of R_{max}. Which results in δ_0 equal to $1, 2$,and 3 respectively.

To update δ, we track the objective function, as in Eq. (3-28). In this case study, we set $\delta_{min} = 1$ and $K = 5$. Since $\delta_{min} = 1$, for $C1$ the update interval is always $\delta = 1$. For $C2$

Table 5-4: Overview of the CAVE parameters.

Parameter	Description	Base line value
K	Minimum witdh update interval	5
δ_{min}	Minimum interval width δ	1
ϵ	Objective function tracker	0.02
Wi	Backlogged trips parameter	$W = i$
Yi	Value of staying parameter	$Y = i$
Ci	Cave initial interval width	$\delta_0 = i$

and $C3$, δ gradually declines from 2 and 3 to 1 respectively, according to Eq. (3-28). We summarise the parameters of CAVE in Table 5-4.

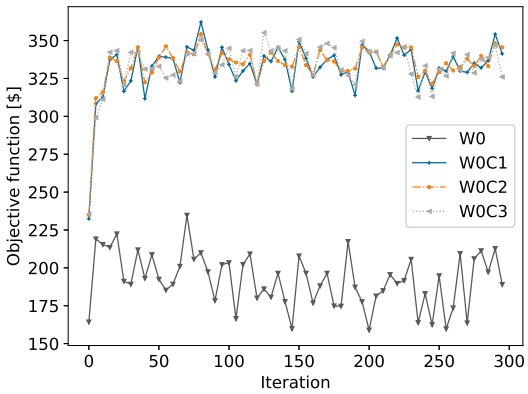
First, we combine strategies from between CAVE and backlogged trips duals. We denote the strategy using $W0$ and $C1$, as $W0C1$. We additional combine $W0$ with $C2$ and $C3$ and depict their respective learning curves in Fig. 5-6. From Fig. 5-6a, we observe that the boost in performance is quite significant for any CAVE strategy. Since each value function update now updates slopes at a wider interval. The value function gets updated at a range of quantities instead of just $y = 0$. The algorithm issues fewer repositioning tasks as can be observed Fig. 5-6b.

We can now exceed the quantity $y = 0$ and sample a non-zero slope at $y = 1$ or even $y = 2$. For this particular setup, the difference between all cave strategies is minimal.

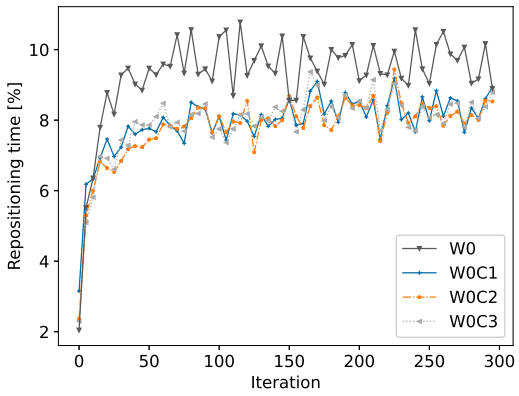
Secondly, we evaluate configurations of CAVE C in combination with Y. We denote the configurations $Y2, Y2C1, Y2C2$ and $Y2C3$. In contrast to the W strategies, implementing CAVE for Y does not yield better performance. Since these strategies are based on congestion avoidance, settings a wider update interval does not increase the performance as it would with W strategies. Moreover, setting a wider interval works counterproductive. If we update the first quantity $y = 0$ by means of a very wide interval with $Y2C3$ then all larger quantities $y > 0$ additionally get updated due to monotonicity of the value function. Overall, pure strategies based on idle vehicles do not give adequate results. However, we employ them to provide additional slope updates, which we further explore in Section 5-2-3.

Value horizon

In our algorithms, the horizon parameter serves a dual purpose. First, it dictates the maximum length a repositioning task can take up. The horizon limits the number of reachable nodes. From Table B-1, we note that with the choice of $H^V = 9$, only a few nodes have less than 4 neighbours. As we increase the horizon, the number of neighbours will naturally increase. Hence, vehicle have more repositioning options and can more easily repositioning away from nodes with low values. Secondly, the horizon dictates the number of slopes any decision will sample. At long horizons, slopes could become dominant in decision-making. To elaborate, we want to avoid the situation where vehicles would rather stay idle than service trips. The value function should not keep cars from their primary objective of servicing trips. It is essential to find a combination of the horizon, discount factor and upper bound on the slopes that avoids the value function being dominant in decision-making.

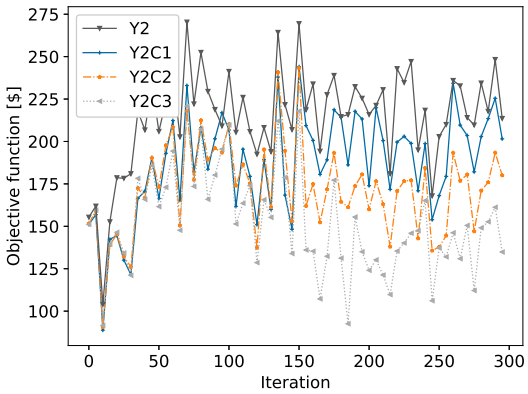


(a) The objective function for different horizons.

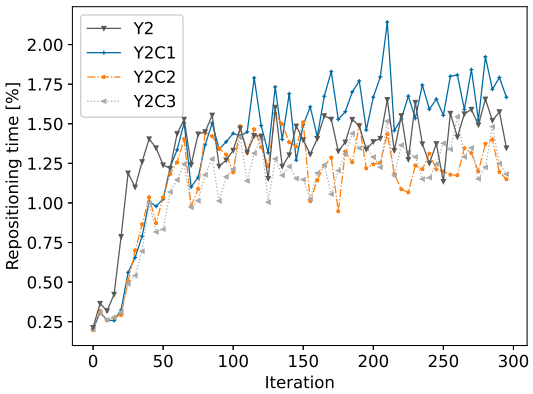


(b) The total time spend repositioning for different horizons.

Figure 5-6: Learning curves of SPAR and CAVE for $W0$.



(a) The objective function for different configurations CAVE.



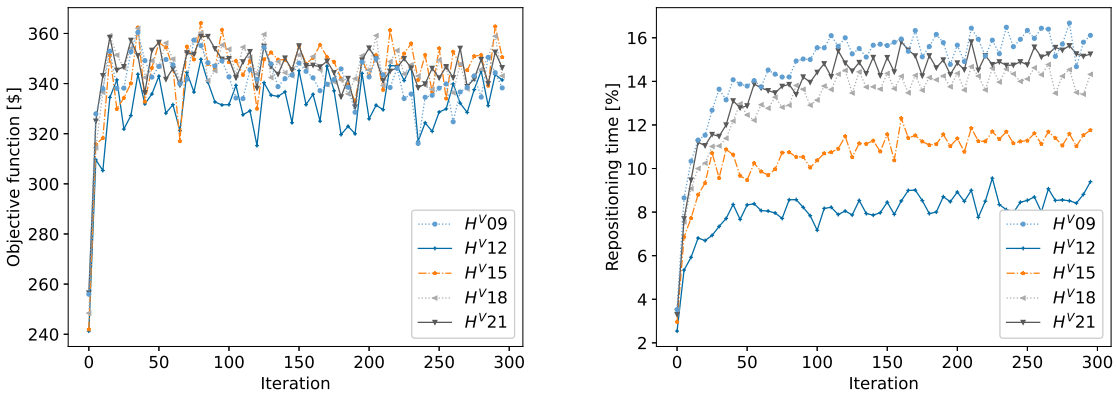
(b) The total time spend repositioning for different horizons.

Figure 5-7: Learning curves of SPAR and CAVE for $Y2$.

We have summarised our results for different horizon in Fig. 5-8 with learning configuration $C2W0Y2$. Starting with the objective function in Fig. 5-8a, we mark a small boost in performance when we consider a horizon beyond 9. However, the differences in maximum objective between horizons 12 to 21 are meagre. $H12$ does have more dips and peaks in the objective function. The longer horizons produce more stable learning and retain less variance between iterations. Furthermore, if we focus on $H15$,$H18$ and $H21$, we observe that the objective of $H21$ is consequently lower than $H15$ or $H18$. For the objective function, $H15$ and $H18$ rank the highest in terms of stability and maximum value. If we now additionally consider the time spend repositioning of Fig. 5-8b, the difference between horizons is more apparent. First, the difference between $H9$ and $H12$ and $H12$ and $H15$ is roughly equal, while the gap between the longer horizons is much smaller. Longer horizons schedule more repositioning tasks, yet

the increase gets damped as we consider longer horizons.

Based on these two metrics, both $H12$ and $H15$ are valid options. $H12$ is more efficient with repositioning though $H15$ is more stable at the cost of 4% more repositioning.



(a) The objective function for different discount factors.

(b) The total time spend repositioning for different discount factors.

Figure 5-8: Learning curves of CAVE for different horizons H^v

Discount factor

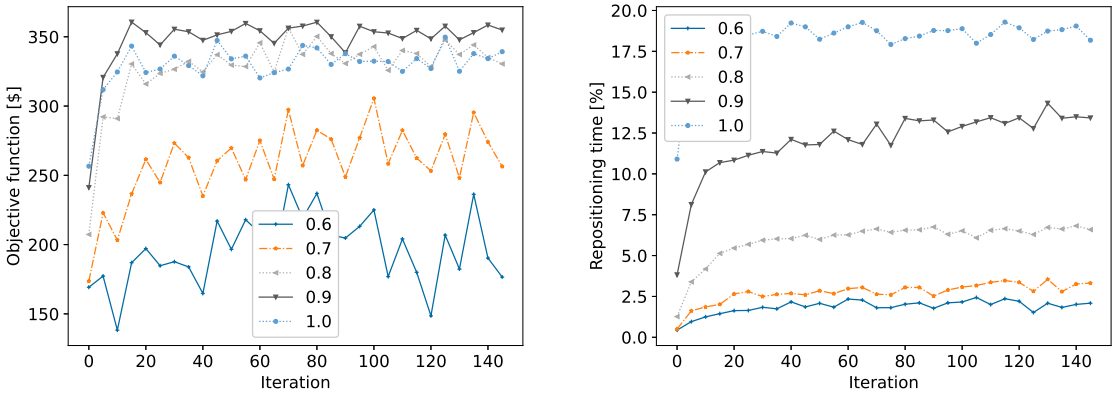
The general ADP community puts the discount factor between 0.5 and 1, where the best value should be found empirically. In our application, the effect of the discount factor is vital in decision-making since it affects the value of every slope within the horizon.

We show the effect of different discount factors on the objective function and repositioning in Fig. 5-9. For this experiment, we used a horizon of $H^V = 15$ to have the most stable performance.

From Fig. 5-9a, we see that $\gamma = 0.6$ and $\gamma = 0.7$ are insufficient to provide a proper repositioning strategy. Logically, increased discounting should decrease repositioning. The discount of the value function at the end of the horizon is significant to incite long repositioning tasks. The value of the slopes $v_{it't}$ of large t' is too small, i.e., at $t' = t + 9$, the discounts of slope $v_{it't}$ with $\gamma = 0.6$ and $\gamma = 0.9$ are 0.017 and 0.39 respectively. The discount of 0.017 is not enough to induce repositioning tasks, except for exceptionally large slopes. Since the slopes have an upper-bound of $v_{max} = 0.5$, discounting with 0.6 results in myopic behaviour.

At the other end of the spectrum, we can consider no discounting with $\gamma = 1$. From Fig. 5-9a, we observe no discounting offers about the same level of objective function compared to $\gamma = 0.8$. Yet, as seen in Fig. 5-9b with $\gamma = 1$ cars spend about three times as much time repositioning.

In general, we observe that a discount factor in the range of $\gamma = 0.8$ to $\gamma = 0.9$ offers the best-combined results in terms of repositioning and objective function.



(a) The objective function for different discount factors.

(b) The total time spend repositioning for different discount factors.

Figure 5-9: Learning curves of $W0Y2C2$ with horizon $H^V = 15$ for different discount factors γ

5-2-3 Verification

We verify the findings of the training iterations on the test data set. To increase applicability, we modify the fleet size and demand factor. The goal is to provide more insight into networks with fleet abundance, as repositioning is more sensible in these networks.

We increase the demand resize factor by 25%, to 0.25 and the fleet size to 40. Based on the results of previous section, we set $H^v = 15$ and $\gamma = 0.8$. We verify with backlogged trips $W = 0$ and vary Y and the update routine between SPAR and different CAVE configurations denoted by C. The results are summarized in Table 5-5.

First, we note the difference between $W0$ with different update routines C. The routine with a $W0C2$ works best with just backlogged trip slopes $W0$ for service rate, objective function and waiting time. For pure W strategies, the difference between interval width of C is minimal, which is consistent with our training iterations of Fig. 5-6a. Since $W0C3$ and $W0C2$ only update from backlogged trips, slopes updated in early iterations are unreachable once the update interval has declined back to $\delta = 1$.

We now introduce the additional value update with Y. We focus on the SPAR update routine results in Table 5-5, denoted with $C0$. If we consider the five different SPAR strategies. $W0Y3$ has the best waiting time and least time spent repositioning while $W0Y2$ performs best in terms of objective function and service rate. The drop in performance at $W0Y0$ stands out and is due to repositioning to avoid vehicle idling. Updates for every idle vehicle in combination with SPAR make idling undesirable. Hence, additional repositioning and decreased performance.

Next, we introduce CAVE as an update routine. Through the wider update interval, slopes at quantity $y = 1$ can be updated by both backlogged trips and idle vehicles. We obtain a more rounded value function thanks to CAVE. Opposite to our previous findings with SPAR, updating for every idle vehicle,$Y0$ works well for CAVE strategies. $C2W0Y0$ and $C3W0Y0$ are the best strategies for all metrics, except for time spend repositioning.

Table 5-5: Results of learning configurations on the test data set. Performance indicators consist of the average results over 35 testing instances. Fleet status: Driving to pickup (D), Serving passenger (S), Parked (P) and Repositioning (R)

Duals		CAVE	Objective Function	Service rate	Wait time (min)	Fleet status / total time				Comp. time (s)
W	Y	C	($)			S	D	P	R	
0	-	0	456.3	93.0%	2.12	23.5%	6.5%	55.4%	14.6%	4.78
0	0	0	409.5	90.3%	2.21	22.8%	6.1%	53.1%	18.0%	14.33
0	1	0	454.0	93.0%	2.09	23.5%	6.5%	55.1%	15.0%	8.65
0	2	0	464.7	93.8%	2.10	23.7%	6.5%	55.1%	14.7%	6.24
0	3	0	462.7	93.3%	2.06	23.6%	6.4%	55.4%	14.6%	4.81
0	-	1	497.0	95.2%	1.97	24.1%	6.6%	58.4%	10.9%	4.88
0	0	1	479.6	93.7%	2.02	23.7%	6.7%	59.9%	9.7%	18.95
0	1	1	479.9	94.5%	2.03	23.9%	6.6%	56.3%	13.1%	14.49
0	2	1	499.5	95.6%	1.98	24.2%	6.7%	57.6%	11.6%	7.15
0	3	1	497.8	95.4%	1.96	24.2%	6.6%	58.2%	11.0%	6.33
0	-	2	506.1	95.8%	1.95	24.3%	6.6%	58.3%	10.8%	5.32
0	0	2	522.6	96.8%	1.87	24.6%	6.6%	55.3%	13.5%	9.80
0	1	2	494.8	95.3%	2.01	24.2%	6.7%	55.9%	13.3%	15.34
0	2	2	497.3	95.3%	1.95	24.1%	6.7%	58.5%	10.8%	9.41
0	3	2	509.9	96.0%	1.91	24.3%	6.7%	57.9%	11.2%	6.38
0	-	3	502.5	95.4%	1.95	24.2%	6.7%	58.7%	10.4%	5.29
0	0	3	521.7	96.8%	1.85	24.5%	6.6%	55.4%	13.6%	7.83
0	1	3	520.6	96.7%	1.86	24.6%	6.6%	55.1%	13.7%	8.28
0	2	3	499.0	95.6%	1.99	24.2%	6.7%	57.8%	11.3%	11.42
0	3	3	506.9	95.8%	1.94	24.3%	6.7%	58.6%	10.5%	8.11

We look at all update routines and observe a general trend when we introduce Y, the total time repositioning jumps up and goes down once we increase Y. However, this trend does not hold for $C1$. At $Y0$, the repositioning time goes down and spikes up again at $Y1$. Due to these kinds of inconsistencies in the results, we cannot find set a general rule for finding the best Y and CAVE configuration.

5-2-4 Summary

In this first case study, we gradually improved the performance of our repositioning strategy. Based on our results for piece-wise and linear value function approximation, we conclude that linear function approximations result in poor quality of service in a simple attribute space, which is in line with Section 3-2-2. Although, their faster run-time might make them attractive to initialise piece-wise approximation where we switch to piece-wise linear approximation after a set number of iterations, as previously observed in Topaloglu et al. [20].

We empirically evaluate several PLAVI strategies. We set ad-hoc upper and lower bounds to the value function to limit the total repositioning time. The value function is updated

using backlogged trip duals. The results indicate that CAVE update routine offers the most potential. Additional slope updates generated by idle vehicles, can offer a boost in performance. We show that different configurations of CAVE and slope update provide adequate repositioning strategies. To conclude, the optimal configuration of the learning parameters depends on the demand and fleet configuration.

5-3 Case study B: Combining the horizon through value functions

In this case study, we explore combining ADP and MPC. From our findings in Case study A, we keep the previous lower and upper bounds and set the discount factor $\gamma = 0.9$. We explore different configurations of ADP, MPC and hybrid horizon on a fleet size of 25 and demand resize factor of 0.2.

Prediction horizon

For our MPC algorithm, we evaluate two different forecasting configurations. We generate trips from our data set with a resize factor of 0.2. For our standard MPC, we generate an additional set of trips to use as forecasts. Additionally we provide the exact trip demand to the MPC to serve as an upper bound, denoted by $(.)^*$ in Table 5-6.

Table 5-6 summarises the results of our experiments for six different horizon configurations. First, let us focus on the performance of the realistic non-perfect forecasting algorithm. The optimal performance is achieved by $H12$, closely followed by $H15$. With performance in service rate, waiting time and time spend repositioning evenly matched between the two. At longer horizons, general performance declines, whilst computation times rises. Hence, we choose to not extend the horizon beyond 8.

If we now consider perfect demand forecasting, we observe the similar patterns. The performance peaks at a horizon of 12 and then slowly declines as we increase the horizon. At $H12^*$ and $H15^*$, all performance indicators match closely, except for the computation time.

If we compare both MPC configurations across all horizons, we find that the objective function and service rate are consistently within a close margin of the perfectly known demand performance. Logically, the average waiting time is significantly higher across all horizons. The increased waiting time highlights the the main flaw of deterministic MPC: Errors within the forecasting. These errors get disclosed once new trip requests arrive. Upon arrival, the algorithm needs to fullfill previously unanticipated demand. It allocates nearby available vehicles to serve the newly arrived trips at any time step within the horizon, which takes time and additional repositioning. Hence, the average waiting time and time spend repositioning rise.

Value horizon

In Section 5-2-2, we looked at the effect of the value horizon during the training phase. We now consider the trained value function on the test data. We ran the best performing configuration of $W0Y0C3$ of our parameter validation study, see Section 5-2-2. We exclude

the horizon of 3 from our experiments due to the minimum travel times between nodes in the network, see Table B-1. Hence, with a horizon of 3, cars can not be rebalanced. For a horizon of 6, we used the trained value function of H^V9.

The results of Table 5-6 are generally in line with the results from Fig. 5-8. The objective function is pretty similar for all horizons considered in Fig. 5-8, except that H^V12 is slightly higher then H^V15 and H^V18. For the time spend repositioning, both Fig. 5-8 and Table 5-6 follow the same trend: the longer the horizon the more time is spend repositioning.

A great attribute of the pure ADP strategies is the low average waiting time. Across all horizons, it is comparable to perfect MPC. However, we observe that the waiting time goes up as we increase the value horizon beyond H^V12. The value function incites additional repositioning at H^V15 and H^V18, as the time spend repositioning is over 20%. This behaviour is undesirable since it differs from the primary objective of servicing ride request. Especially, since cars are unavailable for new decisions while repositioning.

Hybrid horizon

In our hybrid configuration, we like to look at a range of value horizon H^V and prediction horizon H. However, as in the previous section, training a multi-period value function with a really short value horizon was infeasible. Therefore, in our hybrid configuration whenever we use a value horizon of 3 or 6, it utilises the trained multi-period value function trained with a different value horizon, i.e., H^V9.

First, we focus on the light blue rows of Table 5-7. We consider a short prediction horizon of $H3$ and utilise the ADP algorithm as a long term planner. For each configuration, we observe a boost in performance in both objective function and service rate while maintaining a low average waiting time. Furthermore, the time spend repositioning drops significantly about 4% to 5% depending on the configuration. These results indicate that the combined MPC and ADP has great merit from the perspective of pure ADP. The short MPC is effective at reducing over repositioning. The forecasted demand occupies part of the available fleet. Therefore, the value function is now sampled by a smaller proportion of the fleet and issues less repositioning tasks.

Secondly, we consider all configurations with a total horizon of 12 in Table 5-6. At first glance, all configurations show merit. Depending on the performance indicator, we would choose pure MPC for maximum service rate and objective function while opting for pure ADP for minimal waiting and computation time. Our hybrid horizon algorithm acts a gold mean for all quality indicators. The configuration of H^V3H9 offers an excellent alternative to conventional MPC. At the cost of 0.01% of service rate, it decreases of the average waiting time by 6.2%. As we then increase the value horizon, the hybrid algorithm starts to behave similar to ADP. At the shortest MPC horizon, H^V9H3 offers a decrease of 19.1% in average waiting times at the cost of 3% service rate compared to $H12$ Compared to ADP, $H3^V9H3$ increases the revenue by 3% by less time repositioning.

For the configurations with a total horizon of 15 in Table 5-6, we observe that the best performance is achieved with the hybrid configuration H^V3H12. It surpasses conventional MPC and ADP in all three quality of service indicators. We again observe a trend, where as we increase the value horizon the hybrid configuration starts to behave similar to ADP. At

$H^V 6H9$, we observe at an inconstancy in this trend. The service rate and objective at this configuration take a small dip.

Lastly, at configurations with a total horizon of 18 in Table 5-6. Three hybrid configurations stand out, marked in grey in Table 5-6. All three offer a similar quality of service, however $H^V 9H9$ takes the crown in computational efficiency. For both ADP and MPC, performance drops at this total horizon, while the hybrid configurations still offer competitive quality of service.

5-3-1 Conclusions

In this second case study, we examined different configurations of horizon based on MPC, ADP and MPC-ADP. In general, MPC configurations have relatively long average waiting times and high computation times, while the objective and service rate are high. ADP configurations have a general low average waiting time and computation time, at the cost of the service rate and objective function. Our hybrid algorithm was competitive across a range of horizons in a range of configurations.

The best performance was achieved for ADP and MPC at a horizon of 12. At this horizon, our hybrid algorithms offer a significant decrease in waiting time for a small loss in service rate. At the horizon of 15 and 18, our hybrid algorithm offered the most competitive solution for all service quality indicators. Furthermore, the optimal hybrid strategies can be computed within 70% of the time allocated for conventional MPC. To conclude, within the current network a hybrid ADP and MPC repositioning strategy provides promising results in terms of the general quality of service and computation time.

Table 5-6: Results of ADP, MPC and hybrid configurations corresponding to Case Study B. The first three columns denote the horizon configuration, where the first column denotes the total horizon used, the second and third column the value and prediction horizon respectively. Performance indicators consist of the average results over 35 testing instances. Fleet status: Driving to pickup (D), Serving passenger (S), Parked (P) and Repositioning (R).

Horizon configuration			Objective Function ($)	Service rate	Wait time (min)	Fleet status / total time				Comp. time (s)
Total	ADP	MPC				S	D	P	R	
3	-	3	144.4	70.9%	2.26	20.7%	6.3%	72.0%	1.0%	14.95
	-	3*	142.8	70.4%	2.21	20.7%	6.3%	72.3%	0.7%	13.78
6	6	-	171.8	74.1%	2.29	22.2%	6.5%	66.0%	5.4%	1.75
	-	6	297.9	90.4%	2.43	26.2%	7.2%	59.9%	6.9%	29.42
	-	6*	328.1	91.4%	2.10	26.7%	7.5%	61.2%	4.7%	27.73
9	9	-	306.9	91.1%	2.11	26.2%	7.2%	54.5%	12.1%	3.35
	-	9	342.3	95.5%	2.58	28.1%	8.0%	55.8%	8.2%	50.64
	-	9*	378.9	97.4%	2.08	28.2%	8.2%	58.1%	5.5%	46.69
	3	6	333.4	94.9%	2.29	27.3%	7.5%	52.8%	12.5%	17.70
	6	3	316.8	92.4%	2.12	26.5%	7.2%	55.2%	11.2%	9.90
12	12	-	321.5	93.3%	2.07	26.8%	7.3%	48.2%	17.8%	4.62
	-	12	354.1	96.8%	2.57	28.3%	8.1%	55.7%	8.0%	87.67
	-	12*	386.2	98.1%	2.10	28.4%	8.3%	58.3%	5.1%	90.31
	3	9	349.0	96.7%	2.41	27.8%	7.8%	53.8%	10.7%	43.05
	6	6	343.9	96.0%	2.32	27.6%	7.6%	51.3%	13.5%	23.04
	9	3	331.5	93.8%	2.08	26.9%	7.2%	52.1%	13.8%	13.70
15	15	-	308.4	92.4%	2.16	26.6%	7.2%	45.9%	20.3%	6.45
	-	15	350.5	96.8%	2.57	28.0%	8.0%	56.1%	8.0%	98.26
	-	15*	384.9	98.0%	2.13	28.4%	8.3%	58.2%	5.1%	118.17
	3	12	357.2	97.3%	2.46	28.0%	8.0%	55.5%	8.6%	66.76
	6	9	322.2	93.1%	2.35	26.8%	7.4%	56.0%	9.8%	25.52
	9	6	339.3	95.6%	2.31	27.5%	7.4%	50.8%	14.3%	33.00
	12	3	332.8	94.1%	2.13	27.1%	7.2%	51.1%	14.6%	15.94
18	18	-	307.5	91.6%	2.21	27.0%	7.4%	44.7%	20.9%	8.22
	-	18	346.1	96.2%	2.63	28.2%	8.0%	55.3%	8.5%	119.35
	-	18*	377.5	97.6%	2.16	28.2%	8.2%	58.1%	5.6%	192.16
	3	15	351.9	97.1%	2.53	28.0%	8.0%	55.9%	8.2%	103.40
	6	12	353.3	97.0%	2.46	27.9%	7.9%	55.2%	9.1%	73.54
	9	9	350.0	96.7%	2.39	27.7%	7.8%	52.8%	11.7%	45.82
	12	6	341.8	95.9%	2.32	27.6%	7.5%	50.3%	14.7%	33.74
	15	3	327.0	93.4%	2.11	26.8%	7.2%	50.8%	15.3%	20.89

5-4 Case study C: Manhattan network

In this case study, we evaluate which type of repositioning strategy can provide the best service for the Manhattan network. We first evaluate the two strategies along a set of horizons separately and then consider the mixed results. To account for increased network size, we increased the maximum vehicle per region from 5 to 10, thus $R_{max} = 10$.

MPC

For our MPC algorithms, we once again look at the different horizons. In Table 5-7, we summarised the results of our experiments.

For the first three horizons, namely 5, 10 and 15, we computed both types of forecasting setups. $(.)^*$ again denotes the upper bound of MPC, obtained by using perfect knowlegde of the demand over the prediction horizon. The forecast for realistic MPC is obtained from a different sample of resized demand data. We note that in this network, the difference in performance between the controllers is more substantial than in our previous case study. In Table 5-6, the objective function for $H15$ was 91.1% of the upper bound, while in this network it is 85.5%.

Within the network, the computational burden of MPC becomes evident, see the last column of Table 5-7. For a prediction horizon of $H = 15$, the computation time exceeds the simulation time, i.e., 85 minutes. When we further increase the horizon to 20 and 25, the computational time explodes. Whereas $H15$ is nearly real-time realisable, $H20$ and $H25$ far exceed the available computational time with about 3 and 6 times respectively.

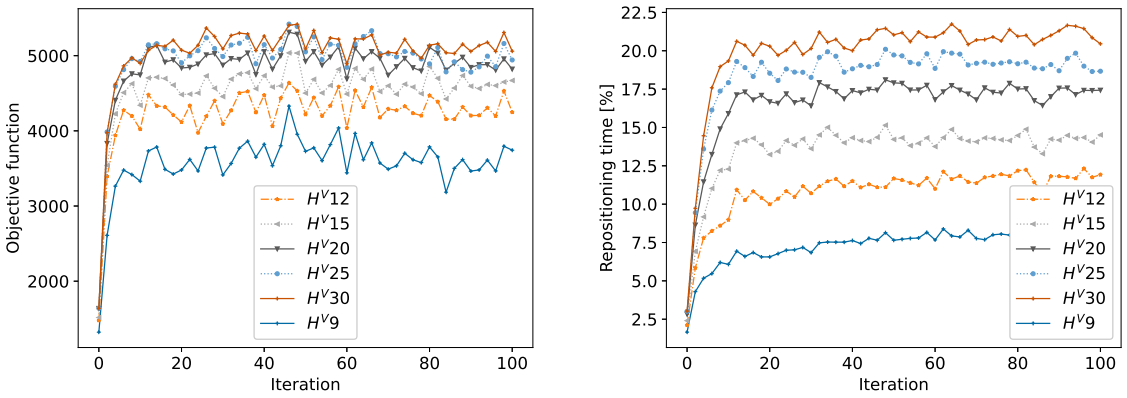
Furthermore, if we compare the performance across all horizons, we observe that $H15$ provides the best quality of service. At $H20$, the time repositioning and waiting time are slightly lower then at $H15$, however they do not make up for the dip in objective function and service rate. Performance wise $H25$ is evenly matched to $H10$, although $H25$ requires a factor of 15 in computation time. To preserve time, we choose not to extend the horizon to $H30$.

In general, our results indicate that forecasting errors have a more significant impact on performance in this bigger network. Furthermore, increasing the horizon increments the effect of error forecasting further, resulting in degrading performance. Lastly, at longer horizons, the MPC is not real-time realisable.

ADP

We compute the value function using artificial duals, as introduced in Section 3-2-4. The two dual parameters are W and Y. In Section 5-2, we explored the effect on the learning curves and for this case study, we compute the slopes using $W = 0$ and $Y = 0$. Next, we choose the width of our update routine CAVE, as explained in Section 3-2-3. We maintain the parameters of Table 5-4 and choose to $\delta_0 = 0.6 \cdot R_{max}$, which we denote as $C6$. To summarise, we denote the learning strategy as $W0Y0C6$ and train Algorithm 2 for a 100 iterations.

Fig. 5-10 shows the respective learning curves for the objective function and repositioning time for different value horizons. We include shorter horizons in Fig. 5-10 to showcase the leaps



(a) The objective function for different configurations of H^V.

(b) The total time spend repositioning for different configurations of H^V.

Figure 5-10: Learning curves for different H^V in the Manhattan network.

in performance and repositioning time between horizons. For the our testing simulations, we exclude these short horizons from the results.

For ADP configurations in Table 5-7, we observe results that are in line with our previous case studies. As we increase the horizon, the time spend repositioning grows. In this new network, the performance seems to stagnate less at longer horizons. From Table 5-7, we observe that H^V30 provides the best quality of services for all three metrics.

The main strength of the value based solutions is the average waiting time, as apparent from sixth column of Table 5-7. For all horizons, ADP services at least 80% of the demand with the shortest waiting times.

At the longest horizon H^V30, the computation time of a single iteration exceeds $20 minutes$. Compared to MPC, 20 minutes is relatively fast, however for ADP we have to take into account the training iterations. It is common to train a value function for at least a hundred iterations. To compute horizons beyond 30, would require more then two days of training time. We therefore choose not to extend the horizon beyond 30. Additionally, we note the resemblance between Fig. 5-8 and Fig. 5-10. We sense that extending the horizon further would not be worth the added computational burden. Extensive learning parameter tuning, as in Section 5-2-2, is more reasonable for improvements in quality of service.

Hybrid

First, we consider the configurations of total horizon of 15 in Table 5-7. For conventional MPC, this configuration $H15$ is optimal however it computes just outside the allocated time. If we shorten of the MPC by 10, we can decrease the computational burden by a factor 3.9. This configuration, H^V10H5, additionally reduces the waiting time by 18%. On the other hand, it transports to 2% less customers. This trade off is reflected by the objective function, which lie very close together. Both these strategies are dwarfed by H^V10H5, which provides better quality of service for all metrics in about half the computational time. It services 4.5%

more customers at a reduced waiting time of 5.2%. Next, we consider the golden prediction counter parts for horizon 15. The hybrid algorithms operate within the 90% of the optimal performance, while conventional MPC only performance within 85% margin. Shortening the horizon appears to reduce the effect of prediction errors on performance.

For next set of rows in Table 5-7, where the total horizon is 20, it is a similar story as the previous section. We can sacrifice a loss in service rate for shorter waiting times. For the best service rate, H^V10H10 is the best choice, while for the shortest waiting time and fast solutions, H^V20 is the best option. For a middle ground between the previous two, H^5H15 is a valid option. Both hybrid options, H^V10H10 and H^V15H5, are solid, especially since they both operate with 97% margin of their perfect predictions counter part.

At the longer horizons of 25 and 30, we choose to only include short MPC horizons. At these horizons, we observe a stagnation of performance in terms of objective function for our hybrid algorithms. The difference between these horizons are marginal at best for both perfect and non-perfect forecasting MPC. Of course, the shorter total horizon has an advantage in terms of computational complexity.

The hybrid strategy H^V20H5 is best strategy that utilises the multi-period value function as long term planner and has a short prediction horizon. Overall, if we disregard the total horizon and look for the high quality repositioning algorithm in Table 5-7. The optimal realisable configuration for service rate and objective function is H^V10H10. While for waiting time, optimality is achieved by a pure value based strategy H^V30.

Fleet activity

To provide further insight into fleet activity over a single iteration, we plot the activity of each vehicle in Fig. 5-11 over the course of a simulation for different configuration of total horizon 15. Each configuration receives the exact same demand scenario from the 7th of June. Furthermore, we initialise the vehicles at the same random positions within the street-level network across all configurations. The simulation consist of a total of 170 steps and at step 20 the first trip requests arrive. Until at step 140, the last requests enter the network and the terminations period starts.

From Fig. 5-11, Fig. 5-11a immediately stands outs compared to Figs. 5-11b to 5-11d. The ADP order over 75% of the fleet to repositioning within the first 20 steps, while in Figs. 5-11b to 5-11d begin reposition activity later and in smaller proportions. However, after the initial peak in repositioning activity, the ADP repositioning activity is significantly smaller during the rest of the simulation.

An added bonus of the ADP is the strategic repositioning at the end of the simulation. Once demand has subsided after 140 steps, the ADP still orders new repositioning tasks, while a purely MPC strategy, see Fig. 5-11b, keeps all vehicles idle. With MPC, the vehicles end the simulation at their last trip destination, while ADP provides last minute congestion avoidance tasks.

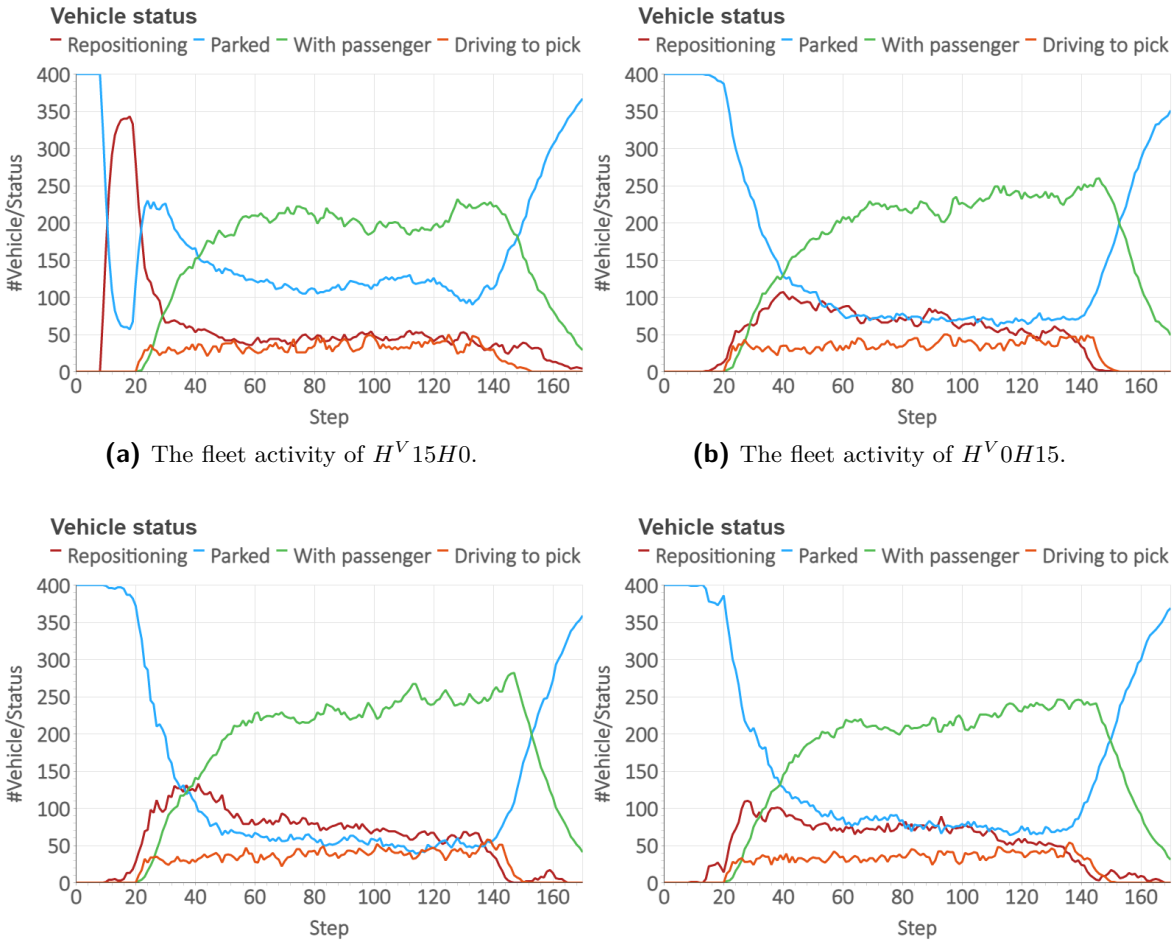
Table 5-7: Results of ADP, MPC and hybrid configurations corresponding to case study C. The first three columns denote the horizon configuration, where the first column denotes the total horizon used, the second and third column the value and prediction horizon respectively. Performance indicators consist of the average results over 7 testing instances. Fleet status: Driving to pickup (D), Serving passenger (S), Parked (P) and Repositioning (R).

Horizon configuration			Objective function	Service rate	Wait time (min)	Fleet status / total time				Comp. time (min)
Total	ADP	MPC				S	D	P	R	
5	-	5	1233.1	53.6%	2.70	26.5%	3.4%	67.6%	2.6%	12.73
	-	5*	1372.9	53.7%	2.44	26.8%	3.4%	67.9%	1.8%	12.79
10	-	10	4361.6	81.4%	3.40	37.4%	4.8%	48.1%	9.7%	38.07
	-	10*	4931.1	84.0%	3.00	38.3%	5.1%	48.5%	8.0%	36.39
15	15	-	4615.6	81.4%	2.40	35.6%	4.9%	45.5%	14.0%	5.04
	-	15	5033.7	87.9%	3.49	39.1%	5.2%	44.4%	11.3%	97.40
	-	15*	5869.7	92.1%	3.00	40.6%	5.6%	44.5%	9.3%	95.79
	5	10	5654.0	92.4%	3.31	40.6%	5.5%	40.7%	13.3%	46.78
	5	10*	6280.7	95.0%	2.82	41.4%	5.7%	41.5%	11.4%	45.87
	10	5	5071.0	85.9%	2.88	37.9%	5.0%	44.7%	12.4%	24.87
	10	5*	5591.3	87.9%	2.42	38.7%	5.1%	45.6%	10.6%	25.33
20	20	-	4788.5	82.9%	2.40	36.3%	5.0%	42.1%	16.7%	8.10
	-	20	4887.7	86.0%	3.41	38.9%	5.0%	45.2%	10.9%	271.96
	10	10	5732.5	93.2%	3.30	40.7%	5.5%	39.7%	14.1%	77.06
	10	10*	5904.8	92.2%	2.90	40.6%	5.5%	43.7%	10.2%	77.15
	15	5	5200.9	87.2%	2.91	38.3%	5.0%	43.5%	13.1%	52.10
	15	5*	5249.8	84.9%	2.40	37.7%	5.0%	48.3%	9.0%	39.95
25	25	-	4793.5	83.1%	2.42	36.6%	5.0%	39.8%	18.6%	14.02
	-	25	4445.2	81.9%	3.40	37.7%	4.8%	47.7%	9.7%	542.60
	20	5	5291.0	88.0%	2.91	38.6%	5.1%	42.7%	13.7%	49.41
	20	5*	5819.7	89.9%	2.51	39.6%	5.2%	43.3%	11.8%	50.24
30	30	-	5081.6	86.2%	2.36	37.0%	5.1%	37.2%	20.7%	20.32
	25	5	5315.5	88.1%	2.93	38.7%	5.1%	42.4%	13.8%	67.62
	25	5*	5891.3	90.9%	2.50	39.7%	5.2%	42.7%	12.4%	70.19

5-5 Conclusions

In this chapter, we evaluated the merits of repositioning strategies based on value functions, MPC and combinations of the aforementioned. We performed street-level ride-hailing simulations across two realistic street-level networks, based in the city of New York. We evaluated the quality of service through the objective function, service rate and waiting time. Furthermore, we examined the time spend repositioning and computation time to analyse the fleet occupancy behaviour and computational burden of our algorithms respectively.

In the first study, we focused on learning a value function. We compared two approximation



(a) The fleet activity of $H^V 15H0$.

(b) The fleet activity of $H^V 0H15$.

(c) The fleet activity of $H^V 5H10$.

(d) The fleet activity of $H^V 10H5$.

Figure 5-11: Overview of fleet activity for demand scenario of Tuesday June 7th

techniques based on linear and piece-wise linear functions. Based on the initial results, we continued with piece-wise linear approximation. We experimented with novel value update strategies based on backlogged trips and the value of staying. To maintain the monotonicity of the value function, we experimented with the update routines known as CAVE and SPAR. Our validation and initial results showed that a value function can be learned by exploiting the backlogged trips. To increase performance, we relaxed the update width of value function through CAVE. The combination of CAVE and backlogged trips produced fast converging and high-quality value functions.

One of the flaws of backlogged trips was addressed by additionally computing the value of staying. Once a value function had converged and provided high-quality service, the number of backlogged trips was greatly reduced. Hence, updates of the value function became sparse. To provide additional value updates, the value of staying could be computed. The value function was then updated more frequently and over a larger interval. These additional updates provided slightly favourable performance in our test data simulations.

In our second case study, we started to compare ADP and MPC. In general, MPC offered

service to the largest groups of customers at relatively long waiting times. While value-based strategies offered faster service to a smaller percentage of customers. At all horizons, hybridisation of the horizon offered a middle ground between the two methodologies.

For optimal performance, we set the horizon to $H12$. At this horizon, hybrid horizons offered two competitive alternatives to conventional MPC. The first option, decreased the waiting time by 6.2% at the cost of servicing 0.01% fewer passengers. The second option, decreased the waiting time by 19.1% at the cost of servicing 3% fewer passengers. Additionally, both options can be computed 49.1% and 84.4% faster respectively.

While both ADP and the hybrid algorithm showed great initial results, the size of the network was inadequate to provide widespread conclusions about our value function and hybrid algorithm. Performance stagnated at longer horizons and the margins between horizon configurations were insignificant.

In our last case study, we addressed the issues of network size. The street-level network covered the entirety of Manhattan Island and the fleet size was increased to 400. The aggregate network was scaled by over a factor of ten and now consisted of 192 regional centres. In this Manhattan network, we performed large-scale ride-hailing simulations.

For conventional MPC, we encountered the two flaws of deterministic MPC. First, we ran into the computational limit. The computation time required more time than available. For horizons above 7.5 minutes, the conventional MPC solutions were not real-time realisable. Second, performance dropped at horizons beyond 7.5 minutes due to forecasting errors. To efficiently solve the combined matching and repositioning sequential decision-making problem, MPC is not a valid option in large networks.

Piecewise-linear value functions showed promise in this larger network. First of all, the value functions exhibited fast convergence rates. Second, they offered service with minimal waiting time. Compared to MPC, customers had to wait 30% less time for transport. Contrarily, the percentage of the total potential customers receiving this fast service was 6.5% smaller. Lastly, the simulations took on average 94.8% less time to compute than MPC.

The hybrid strategies in our final case study again addressed both the computation time of MPC and the subpar service rate of ADP. At peak performance, hybrid strategies offered an increase of 4.5% in service rate and a decrease of 5.2% in the average waiting time over conventional MPC, at roughly half the computational burden.

Chapter 6

Conclusion and discussion

ADP and MPC are state of the art fleet management strategies in shared mobility systems. In this thesis, a hybrid ADP-MPC algorithm is proposed to decrease the computational burden of MPC in MOD. We present two value function approximation algorithms based on linear and piece-wise linear function. In the first case study, the two approximation algorithms were compared and piece-wise linear approximation chosen for our hybrid algorithm. In the second and third case study, MPC, ADP and our hybrid ADP-MPC algorithms were tested across a range of horizon configurations on a realistic street-level network. In this chapter, the research questions are answered, final conclusions are drawn and recommendations for future work are given.

6-1 Conclusions

The main research question of this thesis was:

> Can MPC and ADP be combined for computationally efficient rebalancing of MOD ride-hailing services?

To answer the main research questions, we first cover the three sub-questions:

1. *Can we capture the dynamics of ride-hailing into a MOD model?* In this work, we first analysed the ride-hailing process. The process consists of four core modules: pricing, matching, routing and repositioning. We present a general shared mobility model that incorporates all of these modules. We use a fixed pricing scheme for all trips request and route vehicles over a realistic street-level network. We combine matching and repositioning into a sequential-decision making problem. To reduce the computational complexity, the problem is solved on an aggregated scale. The aggregated network consists of a set of regional centres. Matching and repositioning decisions are determined on an aggregate scale and subsequently assigned to individual trips and vehicles per

regional centre. While this aggregate model suffices for devising potential repositioning and matching strategies, we acknowledge a few limitations of our model:

(a) Network Aggregation: Trips originating within a regional centre can only be pickup by cars within the same regional centre.

(b) Repositioning: Cars cannot be interrupted while repositioning. Once a vehicle enters its target region, it will instantly be available for a new task and remain idle at the entrance node.

(c) Travel times: All travel times within the street and aggregate network are deterministic and have a fixed length. In the aggregate network, all travel times are determined from centre to centre while we simulate trips from door to door. When we forecast trips and movements, the estimated travel time differs from the actual travel time.

(d) Post-decision idling: To determine the number of vehicles at each node, we assume that once a vehicle has finished its task it remains idle at its current position for all time-step remaining in the horizon of the multi-period value function.

To summarise, our model captures the basic dynamics of ride-hailing under mild assumptions.

2. *Can we learn a value function for rebalancing operations in MOD networks?* In short, yes. While we cannot learn the exact value function due to the curse of dimensionality, we learn a geographically separable approximation of the value function. We focused on two approximation strategies based on linear and piecewise-linear functions. A key insight for MOD was to use the slopes of the value function rather than the value function itself. Section 5-2-1 showcased that piecewise-linear approximation works best in simple attribute space. Therefore, we continued on piecewise-linear approximation.

   When we introduced multi-period travel times, we included the arrival of cars into the state set. Tracking vehicles is easy due to deterministic travel times. However, the post-decision states created a problem for approximation of the value function. In short, staying idle samples the slope of next time step while moving to a node samples slopes further into the future. We titled this problem far-sight and addressed it with a novel multi-period value function in Section 3-3.

   The multi-period value functions adopts the idea to use the maximum repositioning length as a horizon. The value of a decision is now the sum of all slopes within the horizon starting from the post-decision state. The multi-period value function has two defining properties. First, long trip requests can have a post-decision state beyond the horizon and are thus not considered within the value function. Additionally, we assume post-decision idling, i.d. vehicles stay idle for all time steps within the horizon after their post-decision state. The value function works as a tool for measuring the optimal fleet distribution, given the current state of available vehicles and vehicles finishing their tasks within the horizon.

   What can be learned from this type of constraint-value function? It is a computationally cheap way to introduce non-myopic fleet repositioning. It could be generally applicable in resource allocation problems or fleet management problems that need the value of

the current state of their resources within the near future. The type of value function could easily be adapted to serve the broader type of inventory problems.

To obtain the slopes for piecewise-linear approximations, we presented two novel strategies based on backlogged trips and idling cars. A key characteristic of piecewise-linear value functions is that the slope is monotonically decreasing. To maintain monotonicity, we used two different slope update routines, SPAR and CAVE. In Section 5-2, we explored different configurations of update routines and slope updates.

We update the value function with the readily available information at each time step. For linear approximation, dual variables are readily available as a product of optimisation. However, for piecewise-linear approximation obtaining dual variables requires additional computation. We proposed to exploit the post-decision demand vector and idling vehicles.

The idea of exploiting the post-decision demand vector, i.e., unserviced trips, could be more widely applicable to other algorithms using partial gradients. While we used it to avoid computation of piecewise-linear duals, it might be applicable in settings that need to obtain partial gradients where dual variables are not readily available. The general notion of using the non-empty post-decision vector to help update the value function has a wider appeal than just piecewise-linear slopes. It should work best in settings where there is often a residual post-decision demand, i.d. excess demand or tasks. Furthermore, the main benefit is in early iterations when the value function is still developing and additional updates can accelerate learning.

Furthermore, we present a novel value update strategy based on idle behaviour. Once a region contains a set quantity of idling vehicles, we assume that an extra vehicle would also remain idle. We then compute the value of the additional vehicle and update the value function. In practice, this strategy is complementary to the backlogged trips values. It mainly provides slope updates when there are no backlogged trips. It counteracts the slope updates provided by backlogged trips and helps to balance the value function.

3. *Can value function approximation be utilised as a terminal cost within the MPC framework to provide real-time realisable rebalancing of MOD in networks?* The motivation of this work was to get insight into the possible merits of combined ADP and MPC in MOD networks. We successfully reduced the computational burden of MPC by shortening the horizon while utilising a value function as a long-term planner.

   In case studies B and C, we evaluated the performance of our constraint horizon value function within the MPC framework. Our hybrid algorithm included two horizon parameters and we experimented with different configurations. The combination of ADP and MPC showed promise in both case studies. In study B, the size of the network limited the insights into performance gains, which stalled at longer horizons for both ADP and MPC. While in study C, we came across unrealisable MPC solutions.

   In this work, we expressed quality of service with the following indicators: objective function, service rate, and computation time. In our studies , ADP offers low average wait and computation time while MPC offers decent objective function and service rate. Our hybrid ADP-MPC algorithm performs as a golden mean of the strengths of both methodologies. We make our final conclusions based on the merit of combined ADP and MPC.

Our findings add to a growing corpus of research showing that combining ADP and MPC has significant potential. With our hybrid configuration, we managed to significantly reduce the computational complexity of repositioning in MOD services. However, all results were generated under the previously mentioned assumptions, Items 1a to 1d, for our MOD model. In spite of these assumptions, our results are notably positive. First, we developed a multi period value function approximation that offers similar quality of service as conventional MPC within 3% of the computation time across multiple horizons. Second, by shorting the horizon of MPC our hybrid algorithm suffers less from forecasting errors and requires less computation time. At long horizons, our hybrid algorithm performs favourably over MPC and ADP. In case study C, compared to best conventional MPC, the hybrid algorithm serves 4.5% more customers at a reduced wait time of 5.2%. To conclude, hybrid repositioning strategy has shown great potential in MOD fleet management. In summary, the contributions of this theses are as follows:

- In Section 3-2-4, we provide two novel methods to approximate the slopes of value functions based on backlogged trips and the value of staying.
- As explained in Section 3-3, we adopt a multi-period value function that approximates the value of all slopes within a rebalancing horizon.
- As shown in Section 4-2, we create a hybrid ADP-MPC algorithm by integrating the multi-period value function into the framework of MPC as a terminal cost.

From an operational perspective, the choice of combined repositioning and matching might not be suitable for every MOD system. Furthermore, most shared mobility services have more complex operations than considered within this work. While our ADP or hybrid strategy would work well as a repositioning module in the ride-hailing process, we recommend further research to validate our findings on multi-period value functions and the use of value function as terminal costs within MPC.

6-2 Future work

Looking forward, further research into the following topics could prove beneficial to the literature.

Piecewise-linear duals

In this work, we use artificial duals to update our approximation of the value function. It would be interesting how real dual variables and artificial duals compare in terms of computational burden and performance. Since it requires additional computation to obtain the dual variables of mixed-integer problems, artificial duals might turn out to be a computationally cheap alternative.

Online learning

n this work, we ran offline training iterations and evaluated use a learned value function on test data. Our learning algorithms all converged to high objective functions within the first few iterations. In online learning, training and validation is combined and it would be interesting to investigate how combined ADP-MPC performance would change when the value function is updated online.

Data-driven network aggregation

In this work, we computed regional centres based on [39]. We hypothesise that the performance of the multi-period value function will increase if the aggregate network is based on the ride-hailing data used during training and validation. The number of nodes per regional centre could vary based on the expected demand for each region, i.e., business district would contain less nodes while residential districts would contain a larger amount of nodes. K-means clustering is prime example of a data-driven method for network aggregation.

Separate matching and repositioning

In this work, we consider the combined demand matching and repositioning problem at every sampling interval. While demand matching at high frequency is good practice, we could consider a lower repositioning frequency to avoid over-rebalancing. In particular, with computationally expensive algorithms such as MPC, the computational burden of the combined problem can grow rapidly, as highlighted in Case Study C.

To combine ADP and MPC, we took the approach of most ADP literature and consider repositioning at every sampling interval. While this is computationally expensive, we find the optimal fleet distribution at every interval and thus repositioning tasks are always in line with the current fleet status. To further reduce the computational burden of fleet management, future work could consider a lower rebalancing frequency and evaluate the impact on the service quality.

Model extensions

Within the MOD literature, common extensions to the basic ride-hailing structure include the following: Ride pooling, heterogeneous fleet size and demand, electric vehicles and dynamic pricing.

High level traffic simulation

The rebalancing problem is practice-oriented and simulation tools that can emulate the high-level interactions in traffic are valuable. Our current model is limited to street-level network simulations with fixed speeds. Traffic simulator SUMO can simulate coupled inter modal trips and can model the influence of congestion (see Lopez et al.[64]). AModeus is a SUMO based open-source simulation-based testbed designed for autonomous mobility-on-demand systems

(see Ruch et al. [65]). The integration in high-level simulators could allow for realistic simulations and gives more insight into the merits of rebalancing for real-life systems (see [14, 66, 58]).

Stochastic approximation

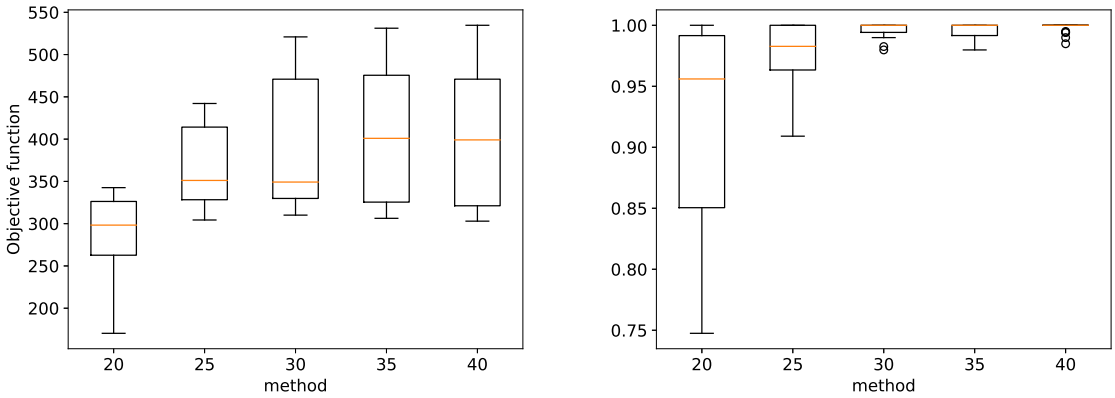
The approximation of the value function is based around deterministic post-decision state. A further point of interest is to adapt our multi period value function approximation to uncertainty in the travel time. It might be trivial due to margins inherited from the sampling rate. Still, it could introduce unforeseen hindrances to the approximation routine.

Appendix A

Supportive Figures

A-1 Fleet Configuration

The effect on objective function and service rate, Fig. A-1, for different fleet sizes for 35 iterations of the test data of the MPC algorithm with a horizon of H^*9.



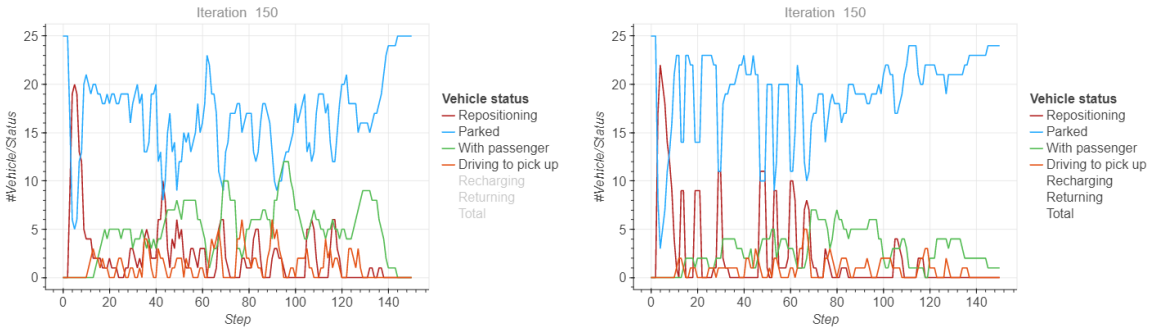
(a) The objective function for different fleet configurations.

(b) The service rate for different fleet configuration.

Figure A-1: Fleet size tests

A-2 Case study A

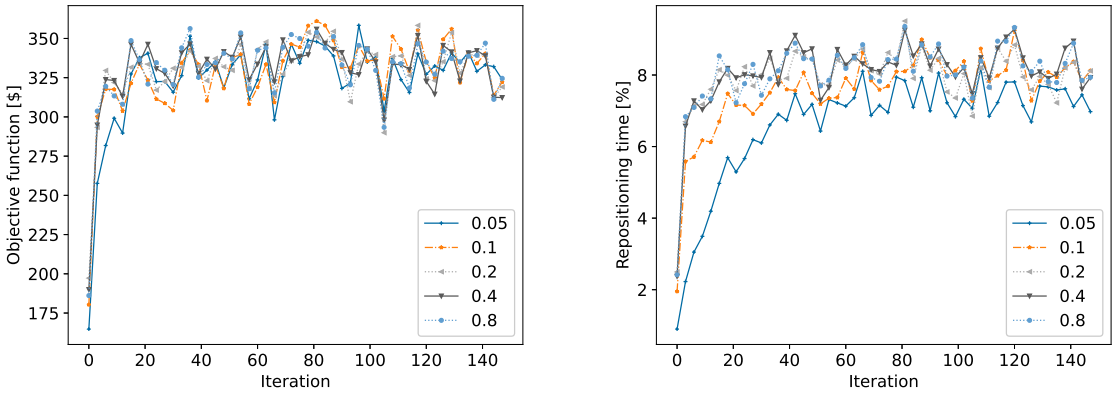
This section provides the supportive figures for case study A.



(a) L5 **(b)** L25

Figure A-2: Fleet status for iteration 150 for L5 and L25

In Fig. A-2, the activity of the fleet is shown for different fleet node congestion strategies $L5$ and $L25$. The red line marks the number of vehicles repositioning at any step within the iteration. In both figures, we observe an initial spike in repositioning activity when vehicles move to the most attractive nodes. Throughout the iteration, the peaks in repositioning activity are larger within Fig. A-2b. At these peaks, vehicles move in clusters to new nodes.



(a) The objective function for different learning rates.

(b) The repositioning time for different learning rates.

Figure A-3: Learning curves of different constant step sizes

Appendix B

Supportive Tables

B-1 OD matrix

In Table B-1, we present the origin destination matrix of our benchmark network used in Section 5-2 and Section 5-3. The last column and row in Table B-1 denote the number of neighbours that can be reached within 9 time steps, with a sampling time of 30 seconds.

Table B-1: OD matrix of travel times in steps between the 120s regional centres used in case study A & B.

Node ID		Destinations															
		448	417	416	197	300	175	465	83	54	279	412	506	251	188	Min	$\# \leq 9$
Origins	448	-	19	15	12	13	11	10	19	23	7	15	7	6	18	6	3
	417	19	-	5	7	7	11	11	12	5	16	9	19	13	5	5	6
	416	15	6	-	5	9	16	7	17	10	20	14	21	13	9	5	5
	197	12	7	5	-	5	11	4	14	11	15	10	17	8	6	4	6
	300	13	7	10	6	-	7	5	10	11	11	6	13	7	6	5	7
	175	11	11	15	11	8	-	10	8	12	5	4	10	5	7	4	6
	465	8	11	7	4	5	9	-	12	15	13	8	15	6	10	4	7
	83	14	21	26	22	18	11	21	-	23	9	15	9	14	17	**9**	2
	54	18	21	25	21	18	10	20	7	-	12	14	16	13	17	7	**1**
	279	7	13	18	13	10	5	12	13	17	-	9	7	6	12	5	5
	412	15	10	14	10	7	4	9	4	11	9	-	13	9	6	4	7
	506	9	17	21	17	14	9	16	16	20	4	12	-	9	15	4	4
	251	6	13	13	8	7	5	7	13	17	8	9	9	-	12	5	8
	188	18	5	9	6	6	7	10	10	6	12	6	16	12	-	5	7
	Min	6	5	5	4	5	4	4	4	5	4	4	7	5	5	-	-
	$\# \leq 9$	4	4	4	6	8	7	5	3	2	6	7	4	8	6	-	-

Bibliography

[1] Mayor of London, "London Infrastructure plan 2050: Transport supporting paper," tech. rep., Greater London Authority, London, 2015.

[2] J. Beaudoin, Y. H. Farzin, and C. Y. Lin Lawell, "Public transit investment and sustainable transportation: A review of studies of transit's impact on traffic congestion and air quality," *Research in Transportation Economics*, vol. 52, pp. 15–22, oct 2015.

[3] S. A. Shaheen, A. Cohen, and E. Farrar, "Mobility on Demand," No. September, pp. 125–155, 2020.

[4] W. J. Mitchell, C. E. Borroni-Bird, and L. D. Burns, "Reinventing the automobile: personal urban mobility for the 21st century," *Choice Reviews Online*, vol. 48, no. 03, pp. 48–1430, 2010.

[5] M. Furuhata, M. Dessouky, F. Ordóñez, M. E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research Part B: Methodological*, vol. 57, pp. 28–46, nov 2013.

[6] Z. Qin, H. Zhu, and J. Ye, "Reinforcement learning for ridesharing: An extended survey [unpublished work]," 2021.

[7] A. Braverman, J. G. Dai, X. Liu, and L. Ying, "Empty-Car Routing in Ridesharing Systems," *Operations Research*, vol. 67, pp. 1437–1452, sep 2019.

[8] R. Iglesias, F. Rossi, K. Wang, D. Hallac, J. Leskovec, and M. Pavone, "Data-Driven Model Predictive Control of Autonomous Mobility-on-Demand Systems," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–7, IEEE, may 2018.

[9] M. Tsao, R. Iglesias, and M. Pavone, "Stochastic model predictive control for autonomous mobility on demand," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-Novem, pp. 3941–3948, dec 2018.

[10] MarketsAndMarkets, "Ride Sharing Market | Industry Key Players, Size, Forecast by 2025."

[11] S. Banerjee and R. Johari, "Ride Sharing," in *Sharing Economy*, pp. 73–97, 2019.

[12] N. Katoh and T. Ibaraki, "Resource Allocation Problems," in *Handbook of Combinatorial Optimization*, pp. 905–1006, Springer US, 1998.

[13] M. Morari and J. H. Lee, "Model predictive control: Past, present and future," *Computers and Chemical Engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[14] A. Carron, F. Seccamonte, C. Ruch, E. Frazzoli, and M. N. Zeilinger, "Scalable Model Predictive Control for Autonomous Mobility-on-Demand Systems," *IEEE Transactions on Control Systems Technology*, pp. 1–10, 2019.

[15] R. Zhang, F. Rossi, and M. Pavone, "Model predictive control of autonomous mobility-on-demand systems," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1382–1389, 2016.

[16] C. Riley, P. van Hentenryck, and E. Yuan, "Real-time dispatching of large-scale ride-sharing systems: Integrating optimization, machine learning, and model predictive control," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2021-Janua, pp. 4417–4423, 2020.

[17] B. A. Beirigo, R. R. Negenborn, J. Alonso-Mora, and F. Schulte, "A business class for autonomous mobility-on-demand: Modeling service quality contracts in dynamic ridesharing systems," *Transportation Research Part C: Emerging Technologies*, vol. 136, no. January, p. 103520, 2022.

[18] B. Beirigo, F. Schulte, and R. R. Negenborn, "Overcoming Mobility Poverty with Shared Autonomous Vehicles: A Learning-Based Optimization Approach for Rotterdam Zuid," vol. 12433 LNCS, no. project 14894, pp. 492–506, 2020.

[19] L. Al-Kanj, J. Nascimento, and W. B. Powell, "Approximate dynamic programming for planning a ride-hailing system using autonomous fleets of electric vehicles," *European Journal of Operational Research*, vol. 284, pp. 1088–1106, aug 2020.

[20] H. Topaloglu and W. B. Powell, "Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems," *INFORMS Journal on Computing*, vol. 18, pp. 31–42, feb 2006.

[21] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality: Second Edition*. Wiley, 2011.

[22] A. D. Little, "Rethinking on-demand mobility," tech. rep., Future of mobility Lab, 2020.

[23] M. W. P. Savelsbergh and M. Sol, "The General Pickup and Delivery Problem," *Transportation Science*, vol. 29, pp. 17–29, feb 1995.

[24] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, pp. 462–467, jan 2017.

[25] BBC, "'Boris bikes': The facts behind 10 years of London's cycle hire scheme - BBC News," jul 2020.

[26] Go Sharing, "GO Sharing | How it works." https://nl.go-sharing.com/en/how-it-works/.

[27] C. Yan, H. Zhu, N. Korolko, and D. Woodard, "Dynamic pricing and matching in ride-hailing platforms," *Naval Research Logistics*, vol. 67, no. 8, pp. 705–724, 2020.

[28] E. Özkan and A. R. Ward, "Dynamic Matching for Real-Time Ride Sharing," *Stochastic Systems*, vol. 10, pp. 29–70, mar 2020.

[29] M. Hu and Y. Zhou, "Dynamic Matching in a Two-Sided Market," *SSRN Electronic Journal*, apr 2015.

[30] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye, "Deep Reinforcement Learning for Multi-Driver Vehicle Dispatching and Repositioning Problem," tech. rep.

[31] W. B. Powell, H. P. Simao, and B. Bouzaiene-Ayari, "Approximate dynamic programming in transportation and logistics: a unified framework," *EURO Journal on Transportation and Logistics*, vol. 1, pp. 237–284, sep 2012.

[32] H. P. Simão, J. Day, A. P. George, T. Gifford, J. Nienow, and W. B. Powell, "An approximate dynamic programming algorithm for large-scale fleet management: A case application," *Transportation Science*, vol. 43, no. 2, pp. 178–197, 2009.

[33] Contributors OpenStreetMap, "OpenStreetMap." https://www.openstreetmap.org/#map=14/40.7701/-73.9535, 2021.

[34] G. Boeing, "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks," *Computers, Environment and Urban Systems*, vol. 65, pp. 126–139, sep 2017.

[35] J. Wen, J. Zhao, and P. Jaillet, "Rebalancing shared mobility-on-demand systems: A reinforcement learning approach," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-March, pp. 220–225, 2018.

[36] M. Gueriau and I. Dusparic, "SAMoD: Shared Autonomous Mobility-on-Demand using Decentralized Reinforcement Learning," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-Novem, pp. 1558–1563, Institute of Electrical and Electronics Engineers Inc., dec 2018.

[37] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (New York, NY, USA), pp. 1774–1783, ACM, jul 2018.

[38] K. Spieser, S. Samaranayake, W. Gruel, and E. Frazzoli, "Shared-Vehicle Mobility-on-Demand Systems: a Fleet Operator'S Guide To Rebalancing Empty Vehicles," *TRB 2016 Annual Meeting*, pp. 0–16, 2016.

[39] C. Toregas, R. Swain, C. ReVelle, and L. Bergman, "The Location of Emergency Service Facilities," *https://doi.org/10.1287/opre.19.6.1363*, vol. 19, pp. 1363–1373, oct 1971.

[40] The New York City Taxi and Limousine Commission, "TLC Trip Record Data," 2020.

[41] S. Shaheen, *Shared Mobility: The Potential of Ridehailing and Pooling*, pp. 55–76. Washington, DC: Island Press/Center for Resource Economics, 2018.

[42] T. Vinks, "Literature Survey: Model and learning-based methods in Mobility-on-demand," 2021.

[43] T. Oda and C. Joe-Wong, "MOVI: A Model-Free Approach to Dynamic Fleet Management," *Proceedings - IEEE INFOCOM*, vol. 2018-April, pp. 2708–2716, apr 2018.

[44] M. Pavone, S. L. Smith, E. Frazzoli, and D. Rus, "Robotic load balancing for mobility-on-demand systems," *International Journal of Robotics Research*, vol. 31, pp. 839–854, jun 2012.

[45] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control," *arXiv*, pp. 1–15, nov 2018.

[46] M. Han, P. Senellart, S. Bressan, and H. Wu, "Routing an autonomous taxi with reinforcement learning," *International Conference on Information and Knowledge Management, Proceedings*, vol. 24-28-Octo, pp. 2421–2424, 2016.

[47] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[48] M. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[49] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, vol. 3. MIT press, 2018., sep 1999.

[50] W. Powell, A. Ruszczyński, and H. Topaloglu, "Learning algorithms for separable approximations of discrete stochastic optimization problems," *Mathematics of Operations Research*, vol. 29, no. 4, pp. 814–836, 2004.

[51] W. B. Powell, "Review of sensitivity results for linear networks and a new approximation to reduce the effects of degeneracy," *Transportation Science*, vol. 23, no. 4, pp. 231–243, 1989.

[52] R. J. Willia, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[53] G. A. Godfrey and W. B. Powell, "An adaptive dynamic programming algorithm for dynamic fleet management, II: Multiperiod travel times," *Transportation Science*, vol. 36, no. 1, pp. 40–54, 2002.

[54] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, pp. 733–764, jul 2003.

[55] J. B. Rawlings and D. Q. Mayne, *Model predictive control: theory and design*. Nob Hill Publishing, 2009.

[56] M. Verhaegen and V. Verdult, *Filtering and system identification: A least squares approach*, vol. 9780521875. Cambridge University Press, jan 2007.

[57] J. Miller and J. P. How, "Predictive positioning and quality of service ridesharing for campus mobility on demand systems," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1402–1408, 2017.

[58] M. Wittmann, L. Neuner, and M. Lienkamp, "A Predictive Fleet Management Strategy for On-Demand Mobility Services: A Case Study in Munich," *Electronics*, vol. 9, p. 1021, jun 2020.

[59] F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas, "Taxi Dispatch with Real-Time Sensing Data in Metropolitan Areas: A Receding Horizon Control Approach," *IEEE Transactions on Automation Science and Engineering*, vol. 13, pp. 463–478, apr 2016.

[60] R. Zhang and M. Pavone, "Control of robotic mobility-on-demand systems: A queueing-theoretical perspective," *The International Journal of Robotics Research*, vol. 35, pp. 186–203, jan 2016.

[61] S. Liu and J. Liu, "A terminal cost for economic model predictive control with local optimality," *Proceedings of the American Control Conference*, pp. 1954–1959, 2017.

[62] L. Hewing, A. Liniger, and M. N. Zeilinger, "Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars," in *2018 European Control Conference, ECC 2018*, pp. 1341–1348, IEEE, jun 2018.

[63] P. M. Bösch, F. Becker, H. Becker, and K. W. Axhausen, "Cost-based analysis of autonomous mobility services," *Transport Policy*, vol. 64, no. September 2017, pp. 76–91, 2018.

[64] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. P. Flotterod, R. Hilbrich, L. Lucken, J. Rummel, P. Wagner, and E. Wiebner, "Microscopic Traffic Simulation using SUMO," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-Novem, pp. 2575–2582, Institute of Electrical and Electronics Engineers Inc., dec 2018.

[65] C. Ruch, S. Horl, and E. Frazzoli, "AMoDeus, a Simulation-Based Testbed for Autonomous Mobility-on-Demand Systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, vol. 2018-Novem, pp. 3639–3644, IEEE, nov 2018.

[66] C. Fluri, C. Ruch, J. Zilly, J. Hakenberg, and E. Frazzoli, "Learning to Operate a Fleet of Cars," *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pp. 2292–2298, 2019.

Glossary

List of Acronyms

MOD	mobility-on-demand
SD	supply-demand
LSTM	long short-term memory
MPC	model predictive control
ADP	approximate dynamic programming
MDP	Markov decision process
MILP	mixed-integer linear program
DP	dynamic programming
AMOD	autonomous mobility-on-demand
VFA	value function approximation
SPAR	separable projective approximation routine
LAVI	linear approximate value iteration
PLAVI	piecewise-linear approximate value iteration
OD	origin-destination

List of Symbols

Symbols related to the mobility-on-demand model

τ	Maximum travel time [Δt]
Δt	Sampling time [s]
$\mathcal{A}$	Set of vehicle attribute vectors
$\mathcal{B}$	Set of trip attribute vectors
$\mathcal{I}$	Set of regional centres
$\mathcal{N}$	Set of nodes

$\mathcal{T}$	Set of time steps
τ_{ij}^{s}	Travel time from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ in seconds
τ_{ij}	Travel time from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ in update intervals Δt
a	Vehicle attribute vector
b	Trip attribute vector
C	Contribution [$]
c_{ijt}^{d}	The cost associated with backlogging a trip from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$
c_{ijt}^{p}	The cost associated with a loaded movement from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$
c_{ijt}^{r}	The cost associated with an empty movement from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$
c_{delay}	Cost of backlogging a trip [$/min]
c_{reject}	Cost of rejecting a trip [$]
c_{time}	Operational cost of a vehicle [$/km]
D_t	The demand state vector of the system at time t
D_{it}	Number of trips from origin $i \in \mathcal{I}$ to destination $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$
F	Objective function [$]
G	Graph
n_d	Destination node
n_o	Origin node
p_{base}	Base fare [$]
p_{time}	Time dependent fare [$/km]
R_t	The resource state vector of the system at time t
R_{it}	The number of resources with at location $i \in \mathcal{I}$ at time $t \in \mathcal{T}$
R_{max}	Maximum vehicles per node
$R_{t',t}$	Number of vehicles inbound to location $i \in \mathcal{I}$ at time period $t \in \mathcal{T}$ and will arrive at location $i \in \mathcal{I}$ at time period $t' \in \mathcal{T}$
S_t	The current state of the system at time t
T	Maximum time period [min]
w	Delay [Δt]
w_{max}	Maximum delay [min]
x_{ijt}^{d}	Number of backlogged trips with origin $i \in \mathcal{I}$ and destination to $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$
x_{ijt}^{p}	Number of vehicles moving loaded from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$
x_{ijt}^{r}	Number of vehicles moving empty from location $i \in \mathcal{I}$ to $j \in \mathcal{I}$ at time period $t \in \mathcal{T}$

Symbols related to approximate dynamic programming

α	Learning rate
$\bar{v}_{it}^{n}$	Estimate of the partial gradient of node i at time t after n iterations

δ	CAVE interval width
δ_{min}	Minimum CAVE interval width
ϵ	perturbation parameter
$\hat{D}$	Sample demand scenario for simulation with length T
$\hat{v}_{it}^{s}$	Value of staying slope
$\hat{v}_{it}^{w}$	Backlogged trip slope
$\hat{v}_{it}$	Partial gradient or slope of node i at time t
$\hat{W}$	Arrival of new information
$\mathbb{P}$	One-step transition probability matrix
D_t^x	Post-decision demand state at time t after taking decision x_t
H^V	Value horizon
K	Minimum iteration count
k	Count variable
R_t^x	Post-decision resource state at time t after taking decision x_t
S_t^x	Post-decision state at time t after taking decision x_t
$s^{S,x}$	Transition function
v^{max}	Upper bound slopes
v^{min}	Lower bound slopes
W	Threshold parameter for computing the value of staying
Y	Threshold parameter for computing backlogged trips slopes
y	Quantity of resources/vehicles
γ	Discount factor
π	Policy
s	State
V	Value function
x	Decision

Symbols related to model predictive control

$\mathcal{U}$	Set of feasible states
$\mathcal{X}$	Set of admissible states
$\mathbf{u}$	Input
$\mathbf{x}$	State
$f(\mathbf{x}, \mathbf{u})$	Dynamical model
H	Prediction horizon
J	Cost function
Q	Cost matrix for the state
R	Cost matrix for the input
U	Input sequence
X	State sequence