

Modeling an On-Demand Ride-Pooling System in Multimodal Microscopic Traffic Simulation

Oytun Arslan

Vollständiger Abdruck der von der Fakultät für Bauingenieurwesen und Umweltwissenschaften der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

angenommenen Dissertation.

Gutachter/Gutachterin:

1. Univ.-Prof. Dr.-Ing. Silja Hoffmann
2. Prof. Dr. Hakan Güler

Die Dissertation wurde am 02.12.2025 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Bauingenieurwesen und Umweltwissenschaften am 24.03.2026 angenommen. Die mündliche Prüfung fand am 27.03.2026 statt.

Acknowledgment

First and foremost, I would like to express my deepest gratitude to Univ.-Prof. Dr.-Ing. Silja Hoffmann for her invaluable guidance, continuous support, and encouragement throughout the course of this research. Her expertise, thoughtful feedback, and dedicated supervision were instrumental in shaping both the direction and quality of this thesis. I am truly thankful for the opportunity to work under her mentorship.

I would also like to extend my thanks to Prof. Dr. Hakan Güler for kindly agreeing to serve as the second examiner of this thesis. His constructive input and valuable feedback helped to significantly enhance the final work. Hakan Hocam, I am especially glad that you could be part of this journey, thank you very much.

Throughout this research, I greatly appreciated the stimulating and insightful discussions with my colleagues at the Chair of Intelligent, Multimodal Transport Systems at the University of the Bundeswehr Munich. Your contributions, collegiality and collaboration have been an important part of this thesis, thank you all.

Lastly, a heartfelt thank you goes to my wife, Julie, for standing by my side through the most challenging phases of this journey, and to my daughters, Dafné and Marlène, whose presence gave me strength. Without your devoted support, this achievement would not have been possible. I am also deeply grateful to my parents, whose steady encouragement and belief in me have laid the foundation for all I have accomplished.

This research is part of the MORE – Munich Mobility Research Campus project, funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr, with financial support from the European Union – NextGenerationEU. I sincerely appreciate the opportunity and the support provided, which have been essential in carrying out this work.

Abstract

This thesis explores the design, implementation, and evaluation of an on-demand ride-pooling system within a multimodal microscopic traffic simulation environment. As cities continue to face challenges such as congestion, emissions, and growing mobility demands, there is an increasing need for flexible and efficient transport solutions. Ride-pooling, where multiple passengers with similar origins and destinations share a vehicle, offers the potential to support more sustainable urban mobility. This research investigates how such systems can be effectively modeled using microscopic simulation, while also incorporating efficient vehicle-request matching methodologies.

The research begins with an extensive literature review covering on-demand mobility systems, shortest route algorithms, vehicle routing problems, and previous studies on ride-hailing and ride-pooling. The review provides essential insights into system implementation, including matching strategies, objective function perspectives and key performance indicators (KPIs). Notably, it also highlights that the modeling of on-demand mobility within microscopic simulations has remained largely unexplored.

Two vehicle-request matching algorithms, spontaneous and batch assignment, are presented in detail and illustrated with examples. Building on this, the application of an on-demand ride-pooling system in the Vissim microsimulation tool is demonstrated. The steps necessary to reflect the dynamic nature of ride-pooling in such a simulation are outlined, including an external control via the COM interface. A further contribution is the development of an interface that enables an automatic connection between a camera-based traffic data collection system with the simulation, simplifying the essential data entry in the simulation.

Extensive simulation experiments are performed for a typical weekday on the campus of the University of the Bundeswehr Munich, providing a realistic test environment. A broad set of KPIs including service rate, sharing rate, mean waiting time, mean detour time, total mileage, mean mileage, vehicle occupancy, mileage index, and empty mileage ratio, are systematically analyzed. Several simulation runs evaluate the impact of operational parameters on system performance, while others investigate the temporal evolution of KPIs throughout the entire simulation day. Additional scenarios assess the effects of travel demand, cost function weighting, and batch periods. Furthermore, comparisons between ride-hailing and ride-pooling operations, as well as between different matching strategies, are conducted. The findings confirm that the developed matching methodologies can be effectively applied within a microscopic simulation, and offer valuable insights into the operational performance of the on-demand ride-pooling system.

In conclusion, this thesis provides both methodological and practical contributions to the modeling and evaluation of ride-pooling services within a microscopic traffic simulation. The insights and methodologies derived from this research are particularly valuable for researchers and practitioners who aim to implement on-demand systems in microsimulation frameworks. While the proposed methodology has been applied in a small-scale model, extending this approach to larger areas presents a promising direction for future research.

Kurzfassung

Die vorliegende Arbeit untersucht die Konzeption, Implementierung und Bewertung eines On-Demand-Ridepooling-Systems in einer multimodalen mikroskopischen Verkehrssimulation. Angesichts der zunehmenden Herausforderungen in Städten, wie Staus, Emissionen und steigender Mobilitätsnachfrage, besteht ein wachsender Bedarf an flexiblen und effizienten Verkehrslösungen. Ridepooling, bei dem mehrere Fahrgäste mit ähnlichen Start- und Zielpunkten ein Fahrzeug gemeinsam nutzen, bietet das Potenzial, eine nachhaltigere urbane Mobilität zu unterstützen. Ziel dieser Arbeit ist es, zu untersuchen, wie solche Systeme mithilfe mikroskopischer Simulationen effektiv modelliert werden können, wobei gleichzeitig effiziente Methoden zur Zuweisung von Fahrzeugen zu Anfragen berücksichtigt werden.

Die Arbeit beginnt mit einer umfassenden Literaturrecherche zu On-Demand-Mobilitätssystemen, Algorithmen für kürzeste Wege, Tourenplanungsproblemen sowie bestehenden Studien zu Ridehailing und Ridepooling. Diese Analyse liefert wesentliche Erkenntnisse für die Systemimplementierung, insbesondere im Hinblick auf Matching-Strategien, Zielfunktionsperspektiven sowie relevante Leistungskennzahlen (KPIs). Dabei wird deutlich, dass die Modellierung von On-Demand-Mobilität in mikroskopischen Simulationen bislang nur unzureichend untersucht wurde.

Im weiteren Verlauf werden zwei Algorithmen zur Zuweisung von Fahrzeugen zu Anfragen, die spontane Zuweisung sowie die Batch-Zuweisung, detailliert vorgestellt und anhand von Beispielen veranschaulicht. Darauf aufbauend wird die Anwendung eines On-Demand-Ridepooling-Systems im Mikrosimulationstool Vissim demonstriert. Die hierfür notwendigen Schritte zur Abbildung der dynamischen Eigenschaften von Ridepooling werden erläutert, einschließlich einer externen Steuerung über die COM-Schnittstelle. Ein weiterer Beitrag der Arbeit ist die Entwicklung einer Schnittstelle, die eine automatische Anbindung eines kamerabasierten Verkehrsdatenerfassungssystems an die Simulation ermöglicht und somit die notwendige Dateneingabe in die Simulation vereinfacht.

Umfangreiche Simulationsexperimente werden für einen typischen Werktag auf dem Campus der Universität der Bundeswehr München durchgeführt und stellen damit eine realitätsnahe Testumgebung dar. Eine breite Auswahl an Leistungskennzahlen, darunter Annahmequote, Poolingquote, mittlere Wartezeit, mittlere Umwegzeit, Gesamtfahrleistung, durchschnittliche Fahrleistung, Besetzungsgrad, Mileage-Index sowie der Leerkilometeranteil, wird systematisch analysiert. Mehrere Simulationsläufe untersuchen den Einfluss betrieblicher Parameter auf die Systemleistung, während weitere Analysen die zeitliche Entwicklung der KPIs über den gesamten Simulationstag hinweg betrachten. Zusätzliche Szenarien bewerten die Auswirkungen der Verkehrsnachfrage, der Gewichtung der Zielfunktion sowie der Batch-Intervalle. Darüber hinaus werden Vergleiche zwischen Ridehailing- und Ridepooling-Betrieb sowie zwischen unterschiedlichen Matching-Strategien durchgeführt. Die Ergebnisse zeigen, dass die entwickelten Methoden erfolgreich in einer mikroskopischen Simulation angewendet werden können und wertvolle Erkenntnisse zur operativen Leistungsfähigkeit von On-Demand-Ridepooling-Systemen liefern.

Zusammenfassend leistet diese Arbeit sowohl methodische als auch praktische Beiträge zur Modellierung und Bewertung von Ridepooling-Diensten in mikroskopischen Verkehrssimulationen. Die gewonnenen Erkenntnisse und entwickelten Methoden sind insbesondere für Forschende und Anwender von Bedeutung, die On-Demand-Systeme in mikroskopischen Simulationen umsetzen möchten. Während die vorgeschlagene Methodik im Rahmen eines begrenzten Untersuchungsgebiets angewendet wurde, stellt die Übertragung auf größere Untersuchungsräume eine vielversprechende Perspektive für zukünftige Forschungsarbeiten dar.

Table of Contents

1. Introduction	1
1.1. Background and Motivation	1
1.2. Research Objective	4
1.3. Scientific Contribution	4
1.4. Thesis Outline.....	5
2. State of the Art.....	6
2.1. On-Demand Mobility Systems	6
2.1.1. Car Sharing.....	6
2.1.2. Car Pooling.....	7
2.1.3. Micromobility Sharing	7
2.1.4. Ride-hailing.....	8
2.1.5. Ride-pooling.....	8
2.2. Shortest Route Algorithms	10
2.2.1. Floyd-Warshall Algorithm	10
2.2.2. Dijkstra's Algorithm.....	11
2.2.3. Yen's Algorithm.....	13
2.3. Vehicle Routing Problems.....	14
2.3.1. Fundamentals.....	14
2.3.2. Overview	15
2.3.3. Traveling Salesman Problem.....	17
2.3.4. Capacitated Vehicle Routing Problem	18
2.3.5. Classical Static Dial-a-Ride Problem	20
2.4. Review on Ride-Hailing and Ride-Pooling.....	22
2.4.1. Ride-Hailing Studies	23
2.4.2. The Concept of Shareability	24
2.4.3. Simulation Platforms	25
2.4.4. Matching Strategies.....	33
2.4.5. Objective Function Perspectives	34
2.4.6. Key Performance Indicators	35
2.5. Summary	36
3. Matching Methodologies.....	40
3.1. Comparison of Spontaneous and Batch Assignment.....	41

3.2.	Spontaneous Assignment of Requests	42
3.2.1.	Creating Options.....	42
3.2.2.	Filtering out Options	45
3.2.3.	Determination of the Best Option.....	45
3.2.4.	Illustration of an Example	46
3.3.	Batch Assignment of Requests	48
3.3.1.	Illustration of an Example	51
3.4.	Summary	57
4.	Application in Microsimulation Environment.....	58
4.1.	Case Study Area	58
4.2.	Modeling in Simulation.....	59
4.2.1.	General Simulation Setup.....	60
4.2.2.	Simulation Setup for the On-Demand Shuttle Traffic.....	66
4.2.3.	Description of the External Python Code	72
4.2.4.	Calibration of the Simulation	74
4.3.	Generation of a Request Pool	75
4.4.	Simulation Workflow	82
5.	Simulation Experiments and Results.....	85
5.1.	Initial Setup	85
5.1.1.	Required Number of Simulation Runs	85
5.2.	Results of Spontaneous Assignment	87
5.2.1.	Impact on Service Rate.....	88
5.2.2.	Impact on Sharing Rate	89
5.2.3.	Impact on Mean Waiting Time.....	90
5.2.4.	Impact on Mean Detour Time	91
5.2.5.	Impact on Mean Mileage.....	92
5.2.6.	Impact on Mean Occupancy	93
5.2.7.	Impact on Mileage Index.....	94
5.3.	Results of Batch Assignment.....	95
5.3.1.	Base Scenario	96
5.3.2.	Impact of Travel Demand.....	101
5.3.3.	Impact of Weighting Parameters in the Cost Function.....	104
5.3.4.	Comparison of Ride-hailing and Ride-pooling	106
5.3.5.	Impact of Batch Period.....	108
5.3.6.	Comparison of Matching Strategies	109
5.4.	Summary	112

6. Conclusions and Outlook.....	114
6.1. Summary and Conclusions	114
6.2. Limitations of the Model	116
6.3. Future Research	117
7. References	119
List of Figures	128
List of Tables.....	130
List of Abbreviations.....	131

1. Introduction

This chapter provides the foundational context for the thesis by presenting its motivation, defining the central research questions, and outlining the key scientific contributions. In addition, it offers an overview of the thesis structure to guide the reader through the subsequent chapters.

1.1. Background and Motivation

The landscape of urban and regional mobility is undergoing a significant transformation. With growing environmental concerns, space constraints in cities, and the limitations of conventional public transport systems, there is an increasing demand for innovative, flexible, and user-oriented mobility solutions. Among these, on-demand mobility systems have emerged as a viable and dynamic alternative, leveraging real-time data, digital platforms, and algorithmic matching to flexibly serve user travel requests (Assmann, 2020). These systems, when effectively integrated, have the potential to enhance accessibility, reduce traffic congestion, and optimize vehicle utilization (Liebchen et al., 2020).

In conventional public transport, vehicles operate according to predefined timetables at predefined routes. An alternative to this approach is on-demand services. On-demand services allow users to submit trip requests by specifying their origin, destination, and preferred departure time. A central dispatching system then determines in the background which available vehicle should serve the request. Two main operational models exist here: ride-hailing and ride-pooling. Ride-hailing services provide exclusive, non-shared trips, functioning similarly to traditional taxi systems. Ride-pooling, on the other hand, enables shared rides, provided that certain criteria, such as spatial and temporal compatibility, are met. This can occur for example if routes of the trips are similar, or if an additional pick-up causes an acceptable detour for the passenger on board (Arslan & Hoffmann, 2023). Such detours are very common in ride-pooling systems, and a direct trip cannot be guaranteed (Narayanan et al., 2020).

A fundamental difference of on-demand systems from conventional public transport lies in service availability. Unlike public transport, which generally ensures service, on-demand systems may reject requests when no suitable vehicle is available. Nevertheless, a primary objective of an on-demand system is to serve as many requests as possible, aiming to achieve a high service rate.

As ride-pooling is a dynamic system where the system state changes constantly, several conditions must be continuously monitored by the central dispatching system to ensure service quality and feasibility.

Firstly, it must be ensured that the capacity of the vehicles is never exceeded. Assigning a new pick-up to a vehicle that is already operating at full capacity can cause operational issues, as accommodating the new passenger would not be possible unless at least one of the current passengers is dropped off beforehand.

Secondly, passenger waiting times must remain within acceptable limits. A newly submitted request should not be assigned to a vehicle that is too far away, as this would result in a long waiting time. Moreover, this constraint must also be considered for passengers who have already been assigned to a vehicle but have not yet been picked up. For example, consider a vehicle en route to pick up passenger A. If another passenger B is assigned in the meantime, whose pick-up location is near the vehicle's current trajectory, the vehicle might be rerouted to pick up B first. While this may seem efficient, it

could result in a certain delay for passenger A. If the resulting waiting time for passenger A exceeds a defined threshold, such an assignment must be avoided.

Lastly, passenger detour times must be kept within acceptable limits. Passengers already on board should not experience disproportionately long detours due to additional pick-ups. For this reason, the detour times of passengers must be checked regularly, and any scenario where the detour exceeds a defined limit must be rejected.

In addition to adhering to the aforementioned operational constraints, a fundamental challenge in ride-pooling lies in the question: “How to match requests with vehicles optimally?”. The answer to this question has a significant impact on system efficiency, service quality and user satisfaction. A simplistic approach might involve assigning each request to the nearest available vehicle, thereby minimizing waiting times. In a static setting, where all vehicles are idle, this strategy might prove effective. However, ride-pooling is a highly dynamic system: vehicles are constantly in motion, picking up or dropping off passengers, and their operational states change continuously.

The disadvantage of this simplistic approach can be illustrated through the scenario depicted in Figure 1.1. In this example, three ride-pooling vehicles move on a square grid network, where each edge representing a travel time of one minute. Vehicle 1 has an on board passenger, whose destination is D_1 . Vehicle 2 also carries a passenger whose destination is D_2 . Vehicle 3 is idle and waiting without any passenger assigned or on board. A new request arrives in the system, with pick-up point X_p and drop-off point X_d .

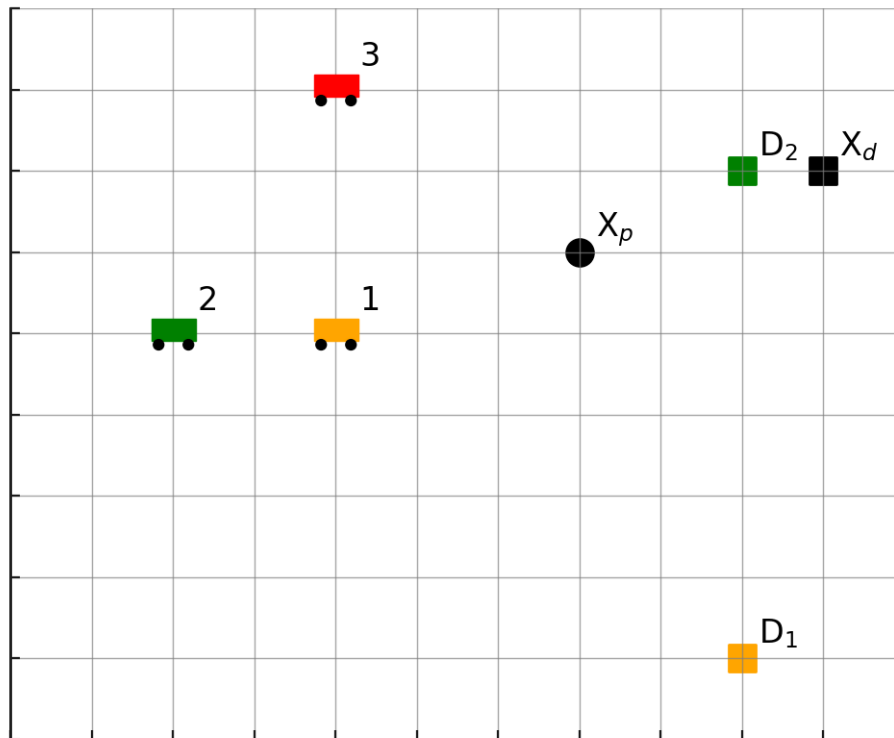


Figure 1.1: Illustration of a scenario on a square grid network.

Applying the “match nearest vehicle” method would lead to the match of vehicle 1 with this request, which is only 4 minutes away from X_p , shorter than the distances from vehicle 2 (6 minutes) and vehicle 3 (5 minutes). However, this match is likely suboptimal, as vehicle 1 is already committed to transporting a passenger to a destination far from the new request's route.

An alternative strategy would be to evaluate the total tour times for each vehicle if they were assigned the new request. In this case, vehicle 1 would likely be excluded due to an excessive detour. Vehicle 2 would require 10 minutes, when the on board passenger is dropped off after the new passenger is picked up. Vehicle 3, on the other hand, would need a tour time of 9 minutes to serve the new request. Based on tour time alone, vehicle 3 might appear to be the better choice. However, this approach still fails to capture the potential routing synergies, because current trajectory of vehicle 2 aligns well with the new request's route, making it the most efficient and logical match.

One can intuitively argue that vehicle 2 is the most appropriate match in this case, due to its current route. However, each of the two methods proposed above leads to different outcomes. This once again highlights the importance of employing efficient matching methodologies in ride-pooling. It also points to the first motivation of this thesis: **developing optimal matching methods for on-demand systems.**

Simulation of ride-pooling systems is crucial to evaluate the system efficiency and understand the impact on key system metrics such as service rate, waiting times, and mileage. Many existing studies in this domain have relied on macroscopic or agent-based simulation platforms to assess the viability and performance of ride-pooling services (see Section 2.4.3). Although these approaches provide useful abstractions and scalability, they often lack the spatial and temporal resolution needed to capture detailed traffic interactions, and infrastructure-specific constraints.

At the same time, microscopic simulation tools are widely used in traffic engineering for assessing and planning road infrastructure, evaluating traffic management measures, as well as testing new technologies and systems (Daamen et al., 2015). However, the modeling of on-demand mobility has remained largely unexplored within microsimulations (see Section 2.4.3.1). This is most likely due to the absence of native support for on-demand features in standard tools. Conventional microsimulation frameworks typically depend on static routing, in which vehicles follow predefined paths at intersections, serving as decision points. This approach can be effective when there are limited route options between origin and destination.

An on-demand system, however, differs fundamentally from this structure. Here, vehicles start their trips in response to real-time events, such as a new request or the need to drop off a passenger, and must compute routes dynamically, based on their current position and prevailing conditions. Moreover, vehicle trips can be generated between any pair of stops, leading to a vast number of routing possibilities. As a result, static routing becomes infeasible for such systems.

This leads to the second motivation of this thesis: **implementing an on-demand ride-pooling system in a microscopic simulation platform.**

This application offers several advantages for researchers:

- It allows a holistic microscopic traffic simulation that incorporates all transport modes, including on-demand services.
- It provides a simulation platform for other on-demand modes as well, such as micromobility and taxis.
- It enables interaction of on-demand vehicles with other transport modes and road infrastructure.
- It facilitates a new testing environment to evaluate various matching algorithms.

Within the scope of MORE Project (MORE Mobility Research, 2024), it was intended to build a Vissim microsimulation model (PTV AG, 2024) of the University of the Bundeswehr Munich campus, and evaluate a potential on-demand shuttle system operating within the campus. By combining these two

work packages, this thesis successfully implemented and evaluated an on-demand shuttle system in the Vissim microsimulation model of the university campus.

As part of the same project, traffic data collection cameras were installed at various locations on campus, supported by a back-end system for storing historical data. This infrastructure was identified as a valuable data source for the simulation model. Accordingly, an interface was developed in this thesis to enable an automated connection between the historical camera data and the simulation.

The modeling of the on-demand system with microsimulation in this thesis is limited to the university campus, thereby serving as a practical example particularly for small-scale models. The insights and methodologies derived from this work can be readily transferred to other small-scale environments such as university campuses, hospital complexes, or constrained urban neighborhoods. Scaling this approach to larger areas presents an opportunity for future research.

In conclusion, this thesis establishes a solid foundation for researchers and practitioners interested in implementing on-demand systems in microscopic traffic simulation frameworks.

1.2. Research Objective

The overarching objective of this thesis is to design, implement, and evaluate an on-demand ride-pooling system within a Vissim microscopic simulation environment. This entails the development of optimum assignment strategies, integration of passenger and operator preferences into the decision-making process, and analysis of system performance under varying operational scenarios.

To address this objective systematically, the research is guided by the following four research questions:

Research Question 1: How can an on-demand ride-pooling system be implemented in a microsimulation model, specifically in Vissim?

Research Question 2: How can requests be matched with vehicles in an optimized way, taking both operator's and passenger's perspectives into account?

Research Question 3: How to build an interface between a camera-based traffic data collection system and the microsimulation model, to automate the vehicle data feed?

Research Question 4: What are the impacts of system parameters and design conditions on the on-demand system KPIs, and how do these KPIs evolve throughout the microsimulation?

1.3. Scientific Contribution

The key contributions of this thesis can be summarized as follows:

A major contribution lies in the design and implementation of an on-demand ride-pooling system in the microscopic traffic simulation tool Vissim. Unlike conventional microsimulation approaches relying on static routing, this work introduces a dynamic routing framework supported by essential configurations in the simulation setup, creating a framework that can be reused and adapted for future studies (see Research Question 1).

A second contribution is the further development and discussion of vehicle-request matching algorithms that incorporate both the operator's and passenger's perspectives to achieve a more efficient matching.

These algorithms are thoroughly described and provided with examples to support future researchers in building upon this work (see Research Question 2).

Another essential contribution is the development of a methodology to enable an automatic connection between a data collection technology and the simulation model, which simplifies the initial data entry in the simulation (see Research Question 3).

Finally, this thesis provides a comprehensive evaluation of an on-demand ride-pooling system on a university campus. The analysis is based on a highly detailed, full-day simulation using a wide set of system metrics, offering valuable insights into operational efficiency and user experience under various system configurations (see Research Question 4).

1.4. Thesis Outline

The structure of the thesis is organized as follows:

Chapter 1 **Introduction** briefly outlines the background and motivation of the thesis, formulates the research questions, and highlights the scientific contributions of the thesis.

Chapter 2 **State of the Art** provides a comprehensive literature review. It starts with a general overview of on-demand mobility systems, followed by a discussion of the mathematical foundations, including shortest route algorithms and vehicle routing problems. The chapter then reviews existing ride-hailing and ride-pooling studies, with particular attention to used simulation platforms, applied matching strategies, adopted objective function perspectives and assessed key performance indicators (KPIs).

Chapter 3 **Matching Methodologies** introduces and explains the two core matching strategies developed for this research: spontaneous assignment and batch assignment. The chapter presents their mathematical formulations, decision logic, algorithm pseudocode, and illustrative examples.

Chapter 4 **Application in Microsimulation Environment** begins with introducing the selected case study area. It then describes the general simulation setup, including the automated integration of traffic camera data into the simulation. The chapter continues with the implementation of the on-demand traffic system, including extensions to the standard simulation tool. After addressing model calibration and request generation for the on-demand system, the chapter concludes with a general overview of the simulation workflow.

Chapter 5 **Simulation Experiments and Results** presents, analyzes, and discusses the results of the full-day simulation experiments built on the foundations of the two previous chapters. A wide range of key performance indicators (KPIs) are used to evaluate the system's performance under various operational scenarios and strategic configurations.

Chapter 6 **Conclusions and Outlook** summarizes the thesis and the key findings, discusses the limitations of the model, and outlines possible directions for future research.

2. State of the Art

As on-demand mobility has gained significant popularity in recent years, numerous research studies have been conducted on the topic. This chapter explores the state of the art in on-demand systems, with a particular focus on ride-hailing and ride-pooling. It begins with an overview of on-demand mobility systems, followed by a description of the most commonly used shortest route algorithms and vehicle routing problems. A comprehensive review of recent ride-hailing and ride-pooling studies is then presented. The chapter concludes with a summary of the review, key lessons learned, and potential applications relevant to this thesis.

2.1. On-Demand Mobility Systems

On-demand systems have been steadily gaining more attraction worldwide in various aspects. Sharing instead of owning goods or services can be observed in different fields, such as streaming services like Netflix, where users pay to access to a vast number of movies, series, or documentaries instead of purchasing individual DVDs. A sharing economy has emerged over the past decades, which can be regarded as a new socio-economic system based on sharing resources rather than owning them. Internet technologies form the basis for this type of trade and business, where an online registration is almost always required (Assmann, 2020).

This trend is also clearly observable in the mobility sector. The ownership of a private car has become less attractive and shared systems are gaining more importance (Ackermann, 2023). These systems are employed by users only when there exists a demand. In this section, different types of on-demand mobility systems are explored with relevant terms and definitions.

2.1.1. Car Sharing

The idea of car sharing is that many people use the same vehicle whenever needed. Although the name contains the word “car”, other vehicle types such as vans can also be part of a car sharing system. It is particularly convenient for leisure and shopping activities as the vehicle is only needed for a limited time, which can be effectively and cost-efficiently covered by a car sharing service. While owning a vehicle incurs costs for the owner - such as taxes, maintenance, or insurance - using car sharing avoids these expenses for the user, since they are borne by the car sharing operator.

There are three main forms of car sharing (Schmöller et al., 2019). In station-based car sharing, users start and end their bookings at predefined stations. Here, two types can be distinguished: (i) in round-trip station-based car sharing, the start and end station must be identical, however (ii) in one-way station-based car sharing, these stations can differ. The latter offers greater flexibility to the user, as it eliminates the need to return the vehicle to the starting station; on the other hand, it creates a requirement for fleet rebalancing on the operator’s side. In Germany, Cambio and Flinkster are some examples of station-based car sharing operators (Cambio, 2024; Flinkster, 2024).

Dependency on the stations causes a certain inconvenience to the users, which has led to the emergence of the second form of car sharing: free-floating car sharing. Here, users can book a vehicle within a predefined area and leave the vehicle anywhere in that area at the end of the booking. It has become

more popular in large cities due to its flexible nature. In Germany, Miles and FreeNow are some examples of free-floating car sharing operators (FreeNow, 2024; Miles, 2024).

Both of these forms are operator-managed. The operator determines the pricing (mostly based on the duration of use and distance driven) and owns the vehicles. The third form of car sharing, the peer-to-peer car sharing, is different from the other two in that individuals can offer their vehicles for the usage of others. They are the vehicle owners and can define the prices as desired. This form is not as common as the other two and mostly observable in the USA (Schmöller et al., 2019).

2.1.2. Car Pooling

In car sharing, the users are also the drivers of the vehicles, and no trip will take place as long as the vehicle is not booked. Car pooling differs from car sharing in the sense that the users are not the drivers, but rather the passengers, and the trip takes place anyway regardless of the car pooling passenger. Here, the driver of the car searches for possible passengers who have a similar route, and shares the trip with them. Car pooling appears mostly in two forms, either (i) informally between individuals, for example, co-workers who live in the same neighborhood and work at the same facility can share the daily trip and also the costs, or (ii) formally via car pooling agencies (Mitfahrzentrale) where trip offers and demands are matched through an internet portal (Harz & Sommer, 2021). In the latter case, the driver defines a fee for the trip and can compensate for the fuel costs in that way, after a certain commission is deducted by the organizing agency. This is a very common form of sharing long-distance trips in Germany, for instance Blablacar is a popular agency used throughout the country (Blablacar, 2024).

2.1.3. Micromobility Sharing

Micromobility sharing has emerged as a transformative solution for urban mobility, providing an efficient, flexible, and sustainable alternative to traditional transportation modes. Shared micromobility includes bicycles, cargo bikes, e-bikes, e-scooters, and other lightweight, low-speed vehicles designed primarily for short-distance trips (Tiessler et al., 2023). These systems are widely used in cities worldwide, helping to reduce traffic congestion, improve first- and last-mile connectivity, complement other transport modes, and promote environmentally friendly travel (Sellaouti et al., 2024). On the other hand, challenges such as infrastructure limitations, regulatory concerns, and issues related to vehicle maintenance and parking need to be addressed to ensure the long-term success of micromobility programs.

Users can access micromobility sharing systems through mobile applications that allow them to locate, unlock, reserve, and rent vehicles. Typically, they must first register on the platform, add a payment method, and scan a QR code on the vehicle to unlock it. Once the ride begins, they can travel to their destination and park the vehicle in designated areas or, in the case of dockless systems, in authorized public spaces. The trip is then completed through the app, which automatically calculates the fare based on the duration or distance traveled.

Some examples of micromobility sharing are the services offered by the companies Tier, Lime, Voi, and Emmy in German cities (Sharing-Vergleich, 2024) and the MORE Sharing implementation in the campus of the University of the Bundeswehr Munich within the scope of the mobility research project MORE (MORE Mobility Research, 2024; Pobudzei et al., 2023).

2.1.4. Ride-hailing

Another type of on-demand mobility systems is ride-hailing, which provides app-based mobility solutions via platforms of so-called transportation network companies (TNC) such as Bolt, Lyft, and Uber (Bolt, 2024; Lyft, 2024; Uber, 2024). These services have gained significant popularity in urban areas, where they provide an alternative to car ownership and improve accessibility, both temporally and spatially. Although they have similarities to the traditional taxi system, they can better leverage dynamic pricing, real-time data, and algorithm-driven route optimization to enhance travel efficiency and reduce wait times (Negro et al., 2021).

In ride-hailing, users can send requests for a trip by typically stating the origin and destination, and the desired time of travel. The matching decision is taken either centrally in a dispatch system, or by individual drivers. Ride-hailing systems substantially offer exclusive services which can be considered a major advantage to the users (Iacobucci & Schmöcker, 2021). Exclusivity here means that the trips are not shared with any other system user, and the next customer can only be served after the current customer's trip is completed. A more efficient way of serving multiple customers could be to bundle similar trips together, which will be introduced in the following section.

2.1.5. Ride-pooling

In conventional public transport, the timetables and departure times are fixed. In addition, the route of the public transport vehicle is also fixed, so transfers between vehicles to reach from origin to destination are not rare. This rigidity leads to user dissatisfaction, which is why new thoughts to evolve the traditional public transport to a more flexible form have been developed, such as ride-pooling (Arslan & Hoffmann, 2025b).

As in ride-hailing, users of ride-pooling can send requests for a trip by stating the origin and destination, and when to travel. The main difference from ride-hailing is that trips may be bundled (pooled) if there is a certain similarity between them, such as similar route and departure times (Kloth & Mehler, 2018). Finding out pooling potentials by individual drivers is not practical, therefore a central dispatching system collects the requests and decides which ones to pool together in which vehicle.

Due to pooling, several trips can be served at the same time with fewer vehicles, which brings efficiency to the whole system (Soteropoulos et al., 2019). However, since other users may be picked up or dropped off on the route, leading to certain detours, the users are generally not guaranteed a direct trip anymore. Mainly, two parameters for the users are essential concerning immediate requests: a maximum waiting time, which defines how long a user can wait to be picked up by a vehicle, and a maximum detour time, which defines how much deviation from the direct trip time can be accepted by the users in case of pooling. The latter also appears as a ratio, describing what percentage is allowed to deviate from the direct trip time (maximum detour ratio).

Currently, ride-pooling is defined in German legislation within the Passenger Transport Act (Personenbeförderungsgesetz, PBefG) with two different forms; either as public ride-pooling (Linienbedarfsverkehr) or private ride-pooling (gebündelter Bedarfsverkehr). While the former is an offer by the public transport operator, the latter is offered by a private entity, but also controlled by the licensing authority (Ritzer-Angerer, 2021).

The idea of bundling trips together in an on-demand manner in Germany dates back to the early 1980s (Assmann, 2020). The first and still existing applications of ride-pooling are in forms of ALT (Anruf-Linien-Taxi, public taxi on-call) and AST (Anruf-Sammel-Taxi, shared taxi on-call). In **public taxi on-**

call, certain bus trips at public transport routes are performed by taxis. Just as in conventional public transport, these trips have fixed routes and timetables. On the other hand, the main contrast is that the trips take place only if there is a pre-registration by the users. Some real-world examples from Germany are the alita in Dresden (alita Anrufdienst, 2024) and the Rufbus in Lüneburg (Rufbus Lüneburg, 2024). Whereas in **shared taxi on-call**, the main distinction from the public taxi on-call is the flexible route. Although the departure times of individual stops are kept as much as possible (small deviations can still be allowed), the routes can deviate from the originally defined route depending on the pre-registrations of the users. That is why the stop sequence may not be strictly followed, some stops may be skipped, and shorter connections may be taken. In case there is only one passenger on a trip, even a direct connection is possible. Some real-world examples from Germany are the RufTaxi in Munich (RufTaxi, 2024) and the AST in Harburg (AST Harburg, 2024).

Both these mobility forms are operated by the local public transport operator and are fully integrated into its fare structure (with a possible comfort surcharge). They run only if there exists a demand and aim principally to complement the existing public transport system. Yet, a fully on-demand feature can be questioned, because there are still timetables; furthermore, the routes are not fully flexible.

Modern ride-pooling operations have gained popularity in the past years, too. Recently, more and more applications have started all over the world, including Germany. Here, ride-pooling is operated hand in hand with public transport, and regarded complementary to the available system, both in temporal and spatial sense, and not as a competitor. Ride-pooling is considered as a first- and last-mile solution, offers affordable services to regions with low demand, and covers time slots with limited public transport availability. For this reason, most of the ride-pooling projects are carried out by the public transport operators themselves. By the end of 2022, there were more than 80 on-demand ride-pooling offers in Germany (Ackermann, 2023). Moreover, ride-pooling is perceived as a future investment, with the hope that autonomous driving will become available and hence eliminate driver costs in the operations.

Some of the ongoing ride-pooling operations in Germany are in Stuttgart (SSB Flex, 2024), in Essen (Bussi, 2024), in Darmstadt (Heiner Liner, 2024), in Leipzig (Flexa, 2024), in Hamburg (MOIA, 2024), in Offenbach (Hopper, 2024), in Munich (MVV Flex, 2024), and in Oberhausen (Revierflitzer, 2024). A review of these systems has revealed that the operators offer typically a seating capacity of 3 to 8 passengers per vehicle. The fleet size also varies; for example, in Hamburg, the operator MOIA has a fleet of 300 vehicles, whereas in Essen there are only 5 Bussi vehicles. The decisive parameters here are the service area and the operation hours.

Operation hours and days differ between operators too: while some serve only during late hours at the weekend (e.g., in Essen), some serve only late hours but throughout the week (e.g., in Oberhausen), and some on the contrary serve daily and 24/7 (e.g., in Offenbach).

There is also a clear difference between operators in terms of advanced bookings. While some of them allow only immediate bookings (e.g., in Essen and Oberhausen), some also offer advanced bookings where the users can book a ride in advance (e.g., in Darmstadt, Offenbach and Munich).

The two main parameters of ride-pooling differ from operation to operation as well. The maximum waiting time observed varies between 5 to 30 minutes, whereas the maximum detour time can be either an absolute value (e.g., 15 minutes in Oberhausen) or a percentage value (e.g., 50% in Darmstadt). These are the parameters set by the operators or the regulatory authorities in order to provide a certain customer satisfaction, and may differ due to the various characteristics of the ride-pooling system.

Pricing of ride-pooling services generally lies between a single public transport ticket and a taxi fare. Different pricing policies observed during the reviews can be summarized as follows:

- Public transport tariff without any surcharge (e.g., in Munich),
- Public transport tariff with a certain surcharge (e.g., in Offenbach),
- Independent tariff, but with a discount for passengers with a subscription in public transport (e.g., in Essen),
- Totally independent tariff (e.g., in MOIA Hamburg).

Apart from that, accompanying passengers generally pay less than the main passenger (up to 50%) in case of a group booking. There are also similar discounts for children if they are accompanied by an adult.

2.2. Shortest Route Algorithms

Calculating the shortest paths is highly essential for on-demand systems since vehicles have flexible routes, and the route with the lowest cost is selected among many. “Lowest cost” is a general term, as each route can have a certain cost value, which can, for example, be related to the distance, travel time, comfort, or road tolls. Distance is used most of the time as the cost for a route, therefore such algorithms are named shortest route algorithms, although they do not have to be the “shortest” literally.

Shortest route algorithms are directly related to graph theory. A road network can be represented as a graph $G = (V, E)$ where V are the vertices and E are the edges of the network. Vertices can be regarded as branching points of the network (e.g., intersections) and edges are the connections between them with certain cost values. In the general case, there are multiple ways of reaching from one vertex to another. Finding out the one with the lowest cost is the objective of a shortest route algorithm.

In the next parts, the most important shortest route algorithms in the literature are explored.

2.2.1. Floyd-Warshall Algorithm

One of the most used algorithms to find out the shortest route in a graph is the Floyd-Warshall algorithm. This algorithm was developed based on dynamic programming; it aims to update the graph matrix gradually and provides an exact solution (Floyd, 1962).

The inputs of the algorithm are the vertices V and the graph matrix of the road network G , which has a size of $|V| \times |V|$. The rows and columns of the graph matrix represent the vertices, and each cell (i, j) of the matrix represents the cost of the edge from i to j . If there is no direct connection, i.e., no edge between two vertices, the corresponding cell has a value of $+\infty$. The diagonal elements of the matrix have a cost value of 0.

At the beginning of the algorithm, a cost matrix is created based on the graph matrix. Then, it is checked for every vertex pair (i, j) if there is a better way of connection, via a third vertex k . If this is the case, then the cell value of this vertex pair (i, j) in the cost matrix is replaced by the value $(i, k) + (k, j)$, which is more optimal. The algorithm returns the cost matrix at the end. A pseudocode of the algorithm can be found below in Algorithm 2.1.

An advantage of the Floyd-Warshall algorithm is that it finds the shortest routes for all vertex pairs. Yet, the disadvantage is that the path of the shortest route is not directly gathered from the algorithm, but rather only the cost of the route. However, it is possible to extend the algorithm with a path matrix to enable finding out the path via backtracking. Moreover, the algorithm is also able to work with negative edge costs, but the emergence of negative cycles within the graph has to be avoided, since this leads to

an infinite cycling at those points while finding the shortest route. The algorithm possesses a time complexity of $O(|V|^3)$ (Gkiotsalitis, 2022).

Algorithm 2.1 Floyd-Warshall Algorithm

```

1  Input: Vertices  $V$  and Graph-Matrix  $G$ 
2  Output: Cost-Matrix  $C$ 
3  set  $C = G$ 
4  for  $k \in V$  do
5      for  $i \in V$  do
6          for  $j \in V$  do
7              if  $C(i,k) + C(k,j) < C(i,j)$  then
8                  set  $C(i,j) = C(i,k) + C(k,j)$ 
9  return  $C$ 

```

2.2.2. Dijkstra's Algorithm

Another well-known shortest route algorithm is the algorithm of Dijkstra. The algorithm operates using a greedy approach combined with a priority queue to iteratively expand the shortest routes (Dijkstra, 1959). Just as with the Floyd-Warshall algorithm, Dijkstra's algorithm also provides an exact solution.

The inputs of the algorithm are the vertices V and the graph matrix of the road network G , which has a size of $|V| \times |V|$. The rows and columns of the graph matrix represent the vertices, and each cell (i,j) of the matrix represents the cost of the edge from i to j . If there is no direct connection, i.e., no edge between two vertices, the corresponding cell has a value of $+\infty$. The diagonal elements of the matrix have a cost value of $+\infty$, too.

A starting vertex can be determined at the beginning of the algorithm. Alternatively, the algorithm can be run for each vertex as the starting vertex, thus the shortest route can be calculated for all vertex pairs. A cost matrix with a size of $|V| \times |V|$ is initialized with all cell values set to $+\infty$, except the diagonal elements, which are set to 0. The cell (i,j) of the cost matrix represents the cost of the shortest route from vertex i to j . In addition, a path matrix with a size of $|V| \times |V|$ is initialized with all cell values set to "undefined". The cell (i,j) of the path matrix contains the penultimate vertex of the shortest route from vertex i to j , which will be used for backtracking later.

The algorithm utilizes a priority queue to determine the order of processing the vertices. At the beginning, the priority queue consists only of the starting vertex (row 8 in Algorithm 2.2). Later during the algorithm, the content of the priority queue is subject to change: The processed vertex will be removed (row 11 in Algorithm 2.2), and other vertices will be added (row 16 in Algorithm 2.2).

Firstly, the starting vertex is processed, as the only vertex in the priority queue. Here, the neighbors of the processed vertex, i.e., the vertices that are connected to this vertex with an edge in the graph are filtered (row 12 in Algorithm 2.2). For each neighbor, it is checked whether its current cost (the cost for reaching it from the starting vertex) can be improved by the cost of the processed vertex and the cost of the edge in question (row 13 in Algorithm 2.2). If this is the case, the corresponding value in the cost matrix is updated (row 14 in Algorithm 2.2) and the processed vertex is written to the corresponding cell of the path matrix (row 15 in Algorithm 2.2). Finally, the neighbors are added to the priority queue for the next iteration (row 16 in Algorithm 2.2). The algorithm continues with the next vertex in the priority queue and runs in this manner until the priority queue is empty (row 9 in Algorithm 2.2).

The crucial point of the algorithm is the order of processing the vertices. This is ensured by the priority queue concept: the vertex with the lowest cost in the priority queue will always be processed as next (row 10 in Algorithm 2.2). When the priority queue is empty, the algorithm terminates and returns the cost matrix as well as the path matrix. The cost matrix includes the costs of shortest routes between each vertex pair, whereas the path matrix is used to reconstruct the path of the shortest routes between each vertex pair using backtracking (see Algorithm 2.3).

Algorithm 2.2 Find the Cost of Shortest Route with Dijkstra's Algorithm

```

1  Input: Vertices  $V$  and Graph-Matrix  $G$ 
2  Output: Cost-Matrix  $C$  and Path-Matrix  $P$ 
3  for  $i \in V$  do
4      set  $Queue = \emptyset$ 
5      for  $j \in V$  do
6          set  $C(i,j) = +\infty$  and  $P(i,j) = \text{undefined}$ 
7      set  $C(i,i) = 0$ 
8      add  $i$  to  $Queue$ 
9      while  $Queue \neq \emptyset$  do
10         set  $v =$  the vertex in  $Queue$  with  $\min C(i,v)$ 
11         remove  $v$  from  $Queue$ 
12         for all  $n \in V$  where  $G(v,n) \neq +\infty$  do
13             if  $C(i,v) + G(v,n) < C(i,n)$  then
14                 set  $C(i,n) = C(i,v) + G(v,n)$ 
15                 set  $P(i,n) = v$ 
16                 add  $n$  to  $Queue$ 
17  return  $C$  and  $P$ 

```

Algorithm 2.3 demonstrates how to find the path of the shortest route using backtracking. The inputs of the algorithm are the path matrix generated in Algorithm 2.2, the start vertex, and the end vertex. To find out the vertex sequence of the shortest route between the start and end vertex, we start with the end vertex (row 3 and 4 in Algorithm 2.3), and then find out the vertex before that within the shortest route between the start and end (row 6 in Algorithm 2.3). In the next step, the same is applied to that vertex (row 7 in Algorithm 2.3), and again the vertex before that within the shortest route between the start and that vertex is explored. This iteration continues until the start vertex is reached (row 5 in Algorithm 2.3). At the end, the formed sequence is reversed (row 8 in Algorithm 2.3) and is returned as the result of the algorithm.

Algorithm 2.3 Find the Path of Shortest Route with Dijkstra's Algorithm using Backtracking

```

1  Input: Path-Matrix  $P$ , Start-Vertex  $s$  and End-Vertex  $e$ 
2  Output: Vertex-Sequence  $S$ 
3  set  $S = \{e\}$ 
4  set  $x = e$ 
5  while  $x \neq s$  do
6      add  $P(s,x)$  to  $S$ 
7      set  $x = P(s,x)$ 
8  reverse  $S$ 
9  return  $S$ 

```

The advantage of Dijkstra's algorithm is that the path of the shortest route can be gathered from the algorithm as well, using backtracking. Yet, the disadvantage is that the algorithm cannot work with

negative edge costs. This is in general not problematic in transportation problems since the edge costs are almost always represented with positive costs. As in the Floyd-Warshall algorithm, Dijkstra's algorithm has a time complexity of $O(|V|^3)$ provided that the algorithm is run for all vertex pairs. If only the shortest routes beginning from a specific vertex, instead of all vertices, are desired, the time complexity becomes $O(|V|^2)$ (Gkiotsalitis, 2022).

2.2.3. Yen's Algorithm

The two algorithms in the previous sections find the shortest route in a road network. However, if the objective is to find not only the shortest route, but a certain number of the shortest routes, for example, to offer more alternatives to road users, these algorithms are insufficient. Yen's algorithm can solve this problem by using a shortest route algorithm such as Dijkstra's as its base (Yen, 1971).

Similar to Dijkstra's, the inputs are the vertices V and the graph matrix of the road network G , which has a size of $|V| \times |V|$. The rows and columns of the graph matrix represent the vertices, and each cell (i, j) of the matrix represents the cost of the edge from i to j . If there is no direct connection, i.e., no edge between two vertices, the corresponding cell has a value of $+\infty$. The diagonal elements of the matrix have a cost value of $+\infty$, too. Additionally, K is the number of shortest routes to be found, s is the start and e the end vertex.

The algorithm uses two containers, A for the K -shortest routes (which will also be returned as the result of the algorithm at the end) and B for the candidates of the shortest routes. At the beginning, both are set as empty (row 3 and 4 in Algorithm 2.4). Then the shortest route from s to e is calculated via a shortest route algorithm, e.g., Dijkstra's (row 5 in Algorithm 2.4), and stored in A as the first route (row 6 in Algorithm 2.4).

The algorithm then loops $K-1$ times to find the next shortest routes (row 7 in Algorithm 2.4). In each loop, the vertices (except last) of the previous shortest route are processed one-by-one (row 8 in Algorithm 2.4). First, the root path is set as the part of the previous shortest route until (including) the processed vertex (row 9 in Algorithm 2.4). After that, for all the shortest routes that contain the same root path, the cost of the edge from the processed vertex to the next vertex in the shortest route is set to $+\infty$ (rows 10-13 in Algorithm 2.4). Here, the aim is to avoid the selection of the same route again: with the edge cost set to $+\infty$, the algorithm is forced to find another way to reach the end vertex e .

With the adjusted graph, the shortest route from the processed vertex to e is calculated (row 14 in Algorithm 2.4). Then, the root path is concatenated with this shortest route (row 15 in Algorithm 2.4) and the resulting route is added to the candidates of the shortest routes B (row 16 in Algorithm 2.4). Before processing the next vertex of the previous shortest route, the edge costs that were assigned to $+\infty$ are replaced with their original values (row 17 in Algorithm 2.4).

After all vertices of the previous shortest route are processed, the route with the lowest cost among the ones in B is selected (row 19 in Algorithm 2.4) and stored in A as the next shortest route (row 20 in Algorithm 2.4). This route is then removed from B (row 21 in Algorithm 2.4).

The algorithm stops normally when the loops are completed for $K-1$ times (row 24 in Algorithm 2.4), or earlier if there are no more shortest route candidates in B (row 22 and 23 in Algorithm 2.4).

Yen's algorithm is able to find the K -shortest routes in a road network as long as there are no negative cycles, since this leads to an infinite cycling at those points while finding the shortest routes. Moreover, the ability of the algorithm to work with negative edge costs depends on the shortest route algorithm on

which Yen's algorithm relies. For example, using Dijkstra's algorithm will prevent Yen's algorithm from using negative costs. The algorithm has a time complexity of $O(K|V|^3)$ (Gkiotsalitis, 2022).

Algorithm 2.4 Find the K-shortest Routes with Yen's Algorithm

```

1  Input: Vertices  $V$ , Graph-Matrix  $G$ , Number of shortest routes  $K$ , Start-Vertex  $s$  and
    End-Vertex  $e$ 
2  Output: Container of  $K$ -shortest routes  $A$ 
3  set  $A = \emptyset$ 
4  set  $B = \emptyset$ 
5  set  $A^1$  = shortest route of  $Dijkstra(s, e)$  based on  $G$ 
6  add  $A^1$  to  $A$ 
7  for  $k = 2; k \leq K; k++$  do
8      for each vertex  $i$  in the shortest route  $A^{k-1}$ , except  $e$  do
9          set  $RootPath$  as the part of  $A^{k-1}$  until (including) the vertex  $i$ 
10         for  $a \in A$  do
11             if  $a$  contains the same root path as  $RootPath$  then
12                 set  $j$  = the vertex after  $i$  in  $a$ 
13                 set  $G(i, j) = +\infty$ 
14             set  $X$  = shortest route of  $Dijkstra(i, e)$  based on  $G$ 
15             set route  $P = RootPath \cup X$ 
16             add  $P$  to  $B$ 
17             restore the edge costs of  $G(i, j)$  that were assigned to  $+\infty$  to their previous values
18         if  $B \neq \emptyset$  then
19             set  $A^k$  = the route in  $B$  with the lowest cost
20             add  $A^k$  to  $A$ 
21             remove  $A^k$  from  $B$ 
22         else
23             return  $A$ 
24  return  $A$ 

```

2.3. Vehicle Routing Problems

Besides finding out the shortest route between two points, another crucial aspect in on-demand ride-pooling is to route the vehicles in an efficient way while ensuring that the customers are served within the operational circumstances. This phenomenon is studied comprehensively in the literature under the Vehicle Routing Problems (VRP). A ride-pooling study therefore cannot be considered without including VRP approaches. This section summarizes the fundamentals of VRP, gives an overview on VRP and presents the solutions to certain VRP types based on the literature.

2.3.1. Fundamentals

The vehicle routing problem (VRP) is a general term for mathematical problems where one or multiple vehicles must travel on certain routes to visit certain points. Vehicles can start at specific locations (often referred to as depots) and return back there at the end of the route, but this is not a necessity. Solutions usually have the objective function to minimize the costs of the route, for example minimizing the distance traveled.

There is a vast number of VRP applications which is why a sub-categorization of VRP has been proven useful. The most common ones are the capacitated vehicle routing problem (CVRP), vehicle routing problem with time windows (VRPTW), pick-up and delivery problem (PDP), and dial-a-ride problem (DARP). In these sub-categories, besides the objective to minimize the route costs, extra constraints have to be applied. In CVRP, each request has a quantitative demand, and each vehicle has a certain capacity which cannot be exceeded during the tour. In VRPTW, each request is associated with a time window, i.e., the visit of this request can only take place within this time window, not earlier and not later. In PDP, unlike the previous categories, each request has a delivery point, too. So, the vehicle picking up the request has to drop off that request as well. PDP is mostly associated with goods transport. In DARP, requests are the travel demand of passengers desiring to travel from one point to another, thus incorporating the pick-up and drop-off issues. DARP usually involves the constraints of the aforementioned sub-categories, such as (i) the vehicles have a limited capacity, (ii) requests are both picked up and dropped off, and (iii) a latest pick-up time for the request (in the form of a maximum waiting time), as well as a latest drop-off time (in the form of a combination of the maximum waiting and ride time) is assigned. Since DARP deals with passenger transport, and not goods transport, user convenience plays here a major role which is reflected by the maximum waiting or ride times (Cordeau & Laporte, 2007; Pillac et al., 2013).

VRP is often studied under the taxonomy in terms of information evolution (static vs. dynamic) and information quality (deterministic vs. stochastic) (Pillac et al., 2013). Concerning the information evolution, if the information, e.g., the passenger requests is known to the operator before any route execution is made, this is defined as a static problem. On the other hand, if such information is revealed throughout the execution, for example new passenger requests appear in the system while the vehicles are on their way, this is defined as a dynamic problem (Cordeau & Laporte, 2007). Another opinion in this context is that the distinction between static and dynamic problems is often not very clear in practice since requests are often cancelled and, as a result, transporters may allow the introduction of new requests into a solution initially designed for a static problem (Borndörfer et al., 1999). In dynamic routing problems, the time invested to search for better decisions comes at the price of reduced reactivity to input changes, which means that there often exists a trade-off between the decision quality and reactivity (Pillac et al., 2013).

Concerning information quality, it depends on whether the information received with the request is certainly known (deterministic) or still undetermined and subject to changes (stochastic) (Ghandeharioun & Kouvelas, 2023; Ho et al., 2018). An example of a stochastic input in this sense can be a passenger request where the boarding time of the passenger is not clearly known (as could be the case of a handicapped person), even if the request itself certainly exists.

2.3.2. Overview

In the literature, the dynamic vehicle routing problem was firstly addressed by Wilson and Colvin (1977). They used a single vehicle and solved a DARP with insertion heuristics. After them, Psaraftis (1980) developed an exact optimization procedure for the single-vehicle DARP with dynamic programming. He examined both the static case where all requests are already known before the vehicle starts its tour, and the dynamic case where requests appear one by one during the course of the vehicle. In both cases, the vehicle capacity and a so-called maximum position shift value were used as parameters. The maximum position shift represented how much the order of a request can be moved compared to the incoming order, in order to avoid an indefinite deferment of a request. The objective function used contained not only the operator side (vehicle mileage), but also the passenger side (waiting time and ride time of a passenger). His idea was using states and defining some rules to check if a state

was feasible or not. The rules involved checking the consistency of the state, the vehicle capacity constraints, the maximum position shift constraints, and the feasibility of its next states. In the last rule, the feasibility of the state depended on the feasibility of the next states, which implied the recursive nature that could be best implemented by dynamic programming. This algorithm did not include any time window constraints, yet a subsequent study from the same author handled time windows on request departure and arrival times as well (Psaraftis, 1983).

Due to the vast number of studies conducted on the topic of VRP in the past, several comprehensive reviews have been carried out on various occasions (Cordeau & Laporte, 2007; Ho et al., 2018; Pillac et al., 2013; Psaraftis, 1995; Psaraftis et al., 2016). Readers are referred to these works for more detailed information. Nevertheless, the most important takeaways from these reviews will be elaborated in the next parts.

In his survey paper, Psaraftis (1995) listed examples from real-world applications where dynamic vehicle routing was involved. One of them was the combined pick-up and delivery services that he defined as follows: *“A fleet of vehicles is available to service demand requests that arrive by phone or some other communication means (e.g. a call-box located on the street) to a central dispatching or scheduling center. These requests are not known in advance to the dispatcher, but rather arrive in real time, and must be serviced as soon as possible. The dispatcher must decide in real time which vehicle to send to a specific request, and how to modify the routing of that vehicle (which may already be committed to pick-up and/or drop-off some other requests). At the same time, a reasonable level of service should be maintained, and the use of vehicle resources should be optimized”*. According to him, the dynamic fashion of the problem led to the fact that the dispatcher is forced to do the assignments in a sequential way, solve the part of the problem based on the available information at the time, and try to solve a new problem as soon as this information changes (Psaraftis, 1995).

Cordeau and Laporte (2007) reviewed the scientific literature on the DARP, updating and extending their previous survey paper on the same topic (Cordeau & Laporte, 2003). They categorized the review as single-vehicle static, single-vehicle dynamic, multi-vehicle static and multi-vehicle dynamic DARP. For the multi-vehicle static case where most of the review took place, besides exact solution methods with integer linear programming, other algorithms were mentioned as well, e.g., heuristics to cluster requests before the routing process that could be served by the same vehicle. The authors concluded their review stating that excellent heuristics exist for the static case solving instances with several hundred users within reasonable times, and that more emphasis should be put on the dynamic case from then on.

In Psaraftis et al. (2016), a detailed taxonomy of the 117 investigated papers has been developed. This taxonomy included the following 11 criteria: Type of problem, logistical context, transportation mode, objective function, fleet size, time constraints, vehicle capacity constraints, the ability to reject customers, the nature of the dynamic element, the nature of stochasticity, and the solution methods.

Ho et al. (2018) focused more on the solution techniques of the DARP and explored 86 papers published in this field after 2007. The investigated solution methods included (i) exact methods, namely branch-and-cut, branch-and-price, and branch-and-price-and-cut algorithms, which are more suitable for the problems where solutions do not have to be produced very quickly, such as the static DARP; (ii) heuristics and metaheuristics, namely construction insertion heuristics, tabu search, simulated annealing, genetic algorithms, variable neighborhood search and large neighborhood search, which fit better for problems requiring fast solutions, such as the dynamic DARP. They also touched upon the issue of the typical objectives of a DARP, indicating both the operator's and the passengers' perspectives. They added that the objectives from the two perspectives can conflict with each other – improving passengers'

experience (e.g., reducing the total ride time and waiting time) may necessitate increasing the operating costs (e.g., operating more vehicles).

2.3.3. Traveling Salesman Problem

The traveling salesman problem (TSP) is a well-known routing problem that aims to find the shortest tour. In this problem, there is a fictitious salesman who starts from his home and must visit every city exactly once before he returns back home (Flood, 1956). Translating this to transportation, this problem is equivalent to the problem of finding the optimal route of an on-demand vehicle that starts from a specific location, picks up a set of passengers from different locations, and returns to its start location (Gkiotsalitis, 2022).

TSP can be ideally expressed in a mathematical graph $G = (V, E)$ where V are the pick-up and start points (vertices) and E are the connections between them (edges) with certain cost values. An edge from vertex i to vertex j is represented as $(i, j) \in E$. If the edge cost values are the distances, then TSP aims to find the shortest tour possible. In this way, TSP tries to find a Hamiltonian cycle with the minimum distance (Bierlaire, 2015).

Although many methods using heuristics and metaheuristics were developed to solve the popular TSP, a basic way is to utilize the integer linear programming (ILP) and obtaining an exact solution. The idea is to use the fact that each vertex is visited exactly once. Let $V = \{0, 1, 2, 3, \dots, n-1\}$ be the set of n vertices where 0 is the start vertex of the vehicle, and each edge (i, j) have a cost of c_{ij} . Also, let x_{ij} be the binary variables which take the value 1 if the edge (i, j) is part of the optimal solution and 0 if not. Then, the optimization problem can be defined as follows:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \quad (2.1)$$

$$\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V \quad (2.2)$$

$$\sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \quad (2.3)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in V \quad (2.4)$$

Equation (2.1) is the objective function to minimize the total sum of the edge costs used in the tour. If an edge is used, its corresponding binary variable x will be 1, leading to the consideration of the cost of that edge. If not, its corresponding binary variable x will be 0, implying that the cost of that edge will not be involved in the tour cost. Constraints (2.2) assure that only 1 outgoing edge is used for every vertex. Similarly, constraints (2.3) ensure that only 1 incoming edge is used for every vertex. Constraints (2.4) show that x_{ij} is a binary variable.

Even if the set of constraints above ensures that each vertex is visited only once, it does not prevent the emergence of sub-tours. In this case, multiple independent cycles (instead of one single cycle) might occur while the given constraints still hold. In order to avoid this, additional constraints in the optimization problem are necessary. A common methodology used here is the method known as the Miller-Tucker-Zemlin (MTZ) formulation (Miller et al., 1960). The main idea of the MTZ formulation is to keep track of the order of the vertices in the tour. If vertex j is visited after vertex i , the position of j in the tour must be 1 more than that of i . Therefore, a re-visit of a vertex, e.g., start and end in a subtour, will be infeasible (Bierlaire, 2015). To address this issue, an extra variable u_i is used, which can only

get integer values and denotes the position of the vertex i within the tour (except the start vertex). Constraints (2.5) and (2.6) represent these conditions:

$$1 \leq u_i \leq n - 1 \quad \forall i \in V \setminus \{0\} \quad (2.5)$$

$$u_i \in \mathbb{Z} \quad \forall i \in V \setminus \{0\} \quad (2.6)$$

The following expression becomes then valid, which implies that the position of the subsequent vertices must increase by at least 1 in the tour:

$$x_{ij} = 1 \Rightarrow u_i + 1 \leq u_j \quad \forall i, j \in V \setminus \{0\} \quad (2.7)$$

However, this expression is not linear and cannot be used in the ILP in this form, therefore it must be linearized first. Conditional expressions can be linearized using the Big-M method which is widely used in optimization tasks in operations research (Williams, 2013). Applying this to the constraints (2.7), the following linearized inequality is achieved:

$$(n - 1)x_{ij} + u_i - u_j \leq n - 2 \quad \forall i, j \in V \setminus \{0\} \quad (2.8)$$

This inequality is indeed correct, because if x_{ij} gets 1, $(u_j - u_i)$ must be greater or equal to 1 according to this inequality, which is always true since $x_{ij} = 1$ also implies that j is visited right after i , and $(u_j - u_i)$ gets the value of 1. In the other case, if x_{ij} gets 0, $(u_i - u_j)$ is smaller or equal to $(n - 2)$, which is also always true since in the extreme case, a vertex can be the first visited one ($u_j = 1$) and another vertex can be the last visited one ($u_i = n - 1$), hence the difference $(u_i - u_j)$ can be maximum $(n - 2)$.

In summary, the Traveling Salesman Problem can be solved by ILP with the objective function given in (2.1) and the linear constraints (2.2) - (2.6) and (2.8).

2.3.4. Capacitated Vehicle Routing Problem

A special case of the TSP is the capacitated vehicle routing problem (CVRP). In CVRP, there are a certain number of identical vehicles which start their tour from the start vertex, aim to pick up the passengers from different locations and bring them back to the starting point. Each vertex in the graph has a passenger demand and is visited once. There are three main constraints in the problem: (i) there is a maximum number of vehicles that can be used, (ii) each vehicle can perform only one route, and (iii) each vehicle has a capacity that cannot be exceeded during the tour.

The capacitated vehicle routing problem was first introduced by Dantzig and Ramser (1959) with the aim of optimizing the routes of a fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal. An exact solution can be obtained by utilizing mixed integer linear programming (MILP). Let $V = \{0, 1, 2, 3, \dots, n-1\}$ be the set of n vertices where 0 is the start vertex of the vehicles, K the number of vehicles available in the fleet, C the capacity of each vehicle, d_i the passenger demand at the vertex i , and c_{ij} the cost of the edge (i, j) . Also, let x_{ij} be the binary variables which take the value 1 if the edge (i, j) is part of the optimal solution and 0 if not. Additionally, let u_i be the continuous variable that denotes the number of passengers in the vehicle while leaving the vertex i (except the start vertex). Then, the optimization problem can be defined as follows:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \quad (2.9)$$

$$\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V \setminus \{0\} \quad (2.10)$$

$$\sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \setminus \{0\} \quad (2.11)$$

$$\sum_{i \in V} x_{i0} = \sum_{j \in V} x_{0j} \quad (2.12)$$

$$\sum_{j \in V} x_{0j} \leq K \quad (2.13)$$

$$Cx_{ij} + u_i - u_j \leq C - d_j \quad \forall i, j \in V \setminus \{0\} \quad (2.14)$$

$$d_i \leq u_i \leq C \quad \forall i \in V \setminus \{0\} \quad (2.15)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in V \quad (2.16)$$

$$u_i \in \mathbb{R} \quad \forall i, j \in V \setminus \{0\} \quad (2.17)$$

Equation (2.9) is the objective function to minimize the total sum of the edge costs used in the tours. If an edge is used by any vehicle, its corresponding binary variable x will be 1, leading to the consideration of the cost of that edge. If not, its corresponding binary variable x will be 0, implying that the cost of that edge will not be involved in the objective function. Constraints (2.10) and (2.11) ensure that only 1 outgoing as well as 1 incoming edge is used for every vertex, respectively. It is important to note that these constraints do not hold for the start vertex, since the outgoing edges of the start vertex can be more than 1 in case multiple vehicles are used. The same also applies to the situation where vehicles return back to the start vertex, and then its incoming edges can be more than 1, too. For the special case of the start vertex, therefore, the constraints (2.12) and (2.13) are used. Constraint (2.12) ensures that the number of outgoing edges of the start vertex must be equal to the number of incoming edges, which can be translated as the number of vehicles leaving the start location must be equal to the number of vehicles returning back there. Constraint (2.13) ensures that this number can be maximum K , which is the number of available vehicles in the fleet.

Constraints (2.14) result from the application of the Big-M method to linearize the conditional expression (2.18), which implies that the number of passengers in a vehicle leaving a certain vertex must be at least as much as the passenger demand of that vertex plus the number of passengers in the vehicle leaving the previous vertex:

$$x_{ij} = 1 \Rightarrow u_i + d_j \leq u_j \quad \forall i, j \in V \setminus \{0\} \quad (2.18)$$

The inequality in constraints (2.14) is indeed correct, because if x_{ij} gets 1, u_j must be greater or equal to $(u_i + d_j)$ according to this inequality, which is always true since $x_{ij} = 1$ also implies that j is visited right after i , and u_j gets the value of $(u_i + d_j)$. In the other case, if x_{ij} gets 0, $(d_j - u_j + u_i)$ is less than or equal to C , which is also always true since in the extreme case, $(d_j - u_j)$ can be at most 0 (u_j cannot be smaller than d_j) and u_i can be at most the capacity of the vehicle C ; hence, the expression $(d_j - u_j + u_i)$ can be maximum the value of C .

Constraints (2.15) state that the number of passengers in the vehicle while leaving a certain vertex can be at most equal to the capacity of the vehicle C and at least equal to the passenger demand at that vertex. Together with constraints (2.14), it is ensured that sub-tours do not occur as well. Lastly, constraints (2.16) and (2.17) show the variables used in the formulation, x_{ij} as binary and u_i as continuous variables.

Comparing the formulations of TSP and CVRP, it can be deduced that TSP is a simplified version of CVRP with the passenger demands for all vertices equal to 1 ($d_i = 1$), the vehicle capacities equal to $n - 1$ ($C = n - 1$), and the number of available vehicles equal to 1 ($K = 1$).

2.3.5. Classical Static Dial-a-Ride Problem

In the dial-a-ride problem (DARP), a trip request is generated for a passenger with known origin and destination points. Although picking up passengers are already considered in CVRP, drop-off events are not taken into account, which is why the combination of the pick-up and drop-off requirements in the scheduling of vehicle routes cannot be addressed by TSP or CVRP formulations (Gkiotsalitis, 2022).

The mathematical model of the DARP which has been studied extensively also by other authors and has formed the basis for other problem extensions, was developed by Cordeau (2006) (Ho et al., 2018). This classical static DARP was defined as follows: There is a fleet of identical vehicles based at the same starting location with a certain capacity. There is also a set of passenger requests with specific pick-up and drop-off locations, each having its own pick-up and drop-off time windows, and passenger demand. The objective is to find out the least cost vehicle routes that cover all requests and satisfy additional constraints such as the maximum ride time of a passenger or the maximum route time of a vehicle. Each vehicle can only perform one route and must return to the starting location at the end.

For the MILP solution, following parameters are defined: Let $V = P \cup D \cup \{0, 2n + 1\}$ be the set of vertices where $P = \{1, 2, \dots, n\}$ is the set of vertices for pick-up points, $D = \{n + 1, n + 2, \dots, 2n\}$ is the set of vertices for drop-off points, and $\{0, 2n + 1\}$ are the two copies of the start vertex of the vehicles. Passenger i is picked up at vertex i and dropped off at vertex $n + i$. Each vehicle $k \in K$ has a capacity of Q_k , and its maximum route time is T_k . Each vertex $i \in P$ has a passenger demand q_i , and the corresponding drop-off vertex has the demand $q_{n+i} = -q_i$. Additionally, each vertex has a service duration (boarding or alighting time) of d_i and a time window of $[e_i, l_i]$. Finally, each passenger has a maximum ride time of L , and each edge (i, j) has a cost of c_{ij} as well as a travel time of t_{ij} .

The variables of the formulation are listed below:

- x_{ij}^k are the binary variables which take the value 1 if the edge (i, j) is travelled by vehicle k in the optimal solution, and 0 if not,
- u_i^k are the continuous variables that denote the time at which vehicle k starts servicing vertex i ,
- w_i^k are the continuous variables that denote the number of passengers in vehicle k while leaving vertex i ,
- r_i^k are the continuous variables that denote the ride time of the passenger associated with the request $(i, n + i)$ on vehicle k .

Then, the optimization problem can be formulated as follows:

$$\min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij}^k x_{ij}^k \quad (2.19)$$

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \quad \forall i \in P \quad (2.20)$$

$$\sum_{j \in V} x_{0j}^k = \sum_{i \in V} x_{i, 2n+1}^k = 1 \quad \forall k \in K \quad (2.21)$$

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{n+i, j}^k = 0 \quad \forall i \in P, k \in K \quad (2.22)$$

$$\sum_{j \in V} x_{ji}^k - \sum_{j \in V} x_{ij}^k = 0 \quad \forall i \in P \cup D, k \in K \quad (2.23)$$

$$u_j^k \geq (u_i^k + d_i + t_{ij})x_{ij}^k \quad \forall i, j \in V, k \in K \quad (2.24)$$

$$w_j^k \geq (w_i^k + q_j)x_{ij}^k \quad \forall i, j \in V, k \in K \quad (2.25)$$

$$r_i^k = u_{n+i}^k - (u_i^k + d_i) \quad \forall i \in P, k \in K \quad (2.26)$$

$$u_{2n+1}^k - u_0^k \leq T_k \quad \forall k \in K \quad (2.27)$$

$$e_i \leq u_i^k \leq l_i \quad \forall i \in V, k \in K \quad (2.28)$$

$$t_{i,n+i} \leq r_i^k \leq L \quad \forall i \in P, k \in K \quad (2.29)$$

$$\max\{0, q_i\} \leq w_i^k \leq \min\{Q_k, Q_k + q_i\} \quad \forall i \in V, k \in K \quad (2.30)$$

$$x_{ij}^k \in \{0, 1\} \quad \forall i, j \in V, k \in K \quad (2.31)$$

$$u_i^k \in \mathbb{R}, w_i^k \in \mathbb{R}, r_i^k \in \mathbb{R} \quad \forall i \in V, k \in K \quad (2.32)$$

Equation (2.19) is the objective function to minimize the total sum of edge costs used in the tours. If an edge (i, j) is used by any vehicle k , its corresponding binary variable x will be 1, leading to the consideration of the cost of that edge for that vehicle. If not, its corresponding binary variable x will be 0, implying that the cost of that edge will not be involved in the objective function.

Constraints (2.20) ensure that only 1 outgoing edge is used for every pick-up vertex. Constraints (2.21) make sure that each vehicle leaves the start vertex once and returns there also once. Constraints (2.22) state that if a pick-up vertex (i) is served by a vehicle, the same vehicle also serves the drop-off vertex $(n+i)$ of the relevant request. Constraints (2.23) ensure that for each pick-up and drop-off vertex, the number of incoming and outgoing edges are equal. Essentially, constraints (2.20), (2.22), and (2.23) together make sure that each pick-up and drop-off vertex is covered exactly once.

Constraints (2.24) guarantee that the service timestamp of vertex j (u_j^k) is at least as much as the service timestamp of the previous vertex i (u_i^k), plus the service duration of vertex i (d_i), plus the travel time from i to j (t_{ij}); provided that a vehicle k travels from i to j ($x_{ij}^k = 1$). Likewise, constraints (2.25) guarantee that the vehicle occupancy while leaving vertex j (w_j^k) is at least as much as that of the vehicle while leaving the previous vertex i (w_i^k), plus the passenger demand at vertex j (q_j); provided that a vehicle k travels from i to j ($x_{ij}^k = 1$). Both of these constraints are nonlinear, therefore applying the Big-M method results in the following linear constraints, where M and N should be chosen as sufficiently large values:

$$u_j^k \geq u_i^k + d_i + t_{ij} - M(1 - x_{ij}^k) \quad \forall i, j \in V, k \in K \quad (2.33)$$

$$w_j^k \geq w_i^k + q_j - N(1 - x_{ij}^k) \quad \forall i, j \in V, k \in K \quad (2.34)$$

Constraints (2.26) define the ride time of each passenger, which then has the upper bound of L (maximum ride time) and the lower bound of $t_{i,n+i}$ (direct travel time from pick-up to drop-off vertex), as stated in constraints (2.29).

Constraints (2.27) ensure that the route time of each vehicle can be at most equal to the maximum route time T_k . Constraints (2.28) restrict the service timestamp of vertex i to the predefined time windows, and constraints (2.30) impose the capacity restrictions for vehicle k at vertex i .

Lastly, constraints (2.31) and (2.32) show the variables used in the formulation; x_{ij}^k as binary and u_i^k , w_i^k , r_i^k as continuous variables.

Although the proposed methodology is robust and can find exact solutions to the MILP, Cordeau (2006) discusses that its applicability is limited rather to small problem instances, and it may not be used to

solve large-scale instances containing hundreds or thousands of users, as is sometimes the case in large cities.

2.4. Review on Ride-Hailing and Ride-Pooling

Several studies have been worked out in the literature regarding on-demand mobility. The works have concentrated mostly on passenger transport, whereas some studies have focused on the logistical aspects (Johnson & Chaniotakis, 2021; Musolino et al., 2021). Johnson and Chaniotakis (2021) investigated the concept of using public transport stops for last-mile delivery in freight services, which is known in the literature as piggybacking. They conducted simulations in SUMO microsimulation for Washington, D.C., involving two public transport bus routes whose stops were conveniently close to last delivery locations. The routes were optimized by solving a capacitated vehicle routing problem (CVRP). The results indicated that the total vehicle time around the network depots could be decreased.

In another logistics study, Musolino et al. (2021) presented a framework to integrate the path choice problem and vehicle routing problem (VRP) in freight transport, which were traditionally studied separately in the literature. They performed experimental tests using the criteria minimum distance, minimum energy, and minimum time to generate route alternatives. Results of the study showed that their model was useful to identify choice factors within the freight distribution process and to support the evaluation of city logistics measures.

Regarding passenger transport, the introduction of on-demand systems brought new aspects, such as the stopping events of vehicles. In almost all ride-hailing and ride-pooling systems, the vehicles can stop at virtual stops for boarding and alighting, which are most of the time not noticeable from outside but visible in the smartphone apps. The positions of these stops however have an impact on the traffic flow, especially in urban regions. The study of Stüger et al. (2022) touched upon this issue. They investigated the effects of on-demand mobility stop locations on the traffic flow via delay times. They proposed and evaluated two methods to mitigate the impacts of pick-up and drop-off stops in an urban context. In the first method, they determined the position of stops (upstream, downstream, left-turner, or right-turner) according to the real-time signal timings. In the second method, they applied a temporary reduction of the number of stopping maneuvers on saturated street sections. Both methods were evaluated using Vissim microsimulation with varying traffic demand, and results indicated that significant vehicle delay reductions could be achieved.

The dwell times at the stops of ride-pooling services were investigated in the work of Harmann et al. (2024). They aimed to fill a research gap as existing dwell time studies mostly focused on conventional transport modes, and few studies were conducted in the field of ride-pooling. For this purpose, they collaborated with a real-world ride-pooling service in Hannover and collected data. The findings revealed an average dwell time of 16 s and a median of 13 s for a total of 221 dwell processes, yet the sample was dominated by teenage users (ca. 50%), which is why it had to be considered carefully. They also investigated the effects of other parameters on dwell times such as the vehicle type, passenger group size, presence of luggage, presence of children, and type of dwelling (boarding or alighting).

Due to their nature, ride-hailing and ride-pooling systems cannot always guarantee a service. When a customer sends a request for an immediate trip, it may occur that the system cannot find an appropriate vehicle to perform the ride. Limitations on the passenger waiting times, detour times in ride-pooling case, and vehicle capacities can be the possible reasons. In those cases, rejection of the request becomes inevitable, although the operators aim to serve as many requests as possible.

The methodologies in the studies either remove the request from the system as soon as it is rejected (Berbeglia et al., 2011; Jung & Jayakrishnan, 2014; Simonetto et al., 2019), or keep the request for the next assignment round in the system for another chance (Engelhardt et al., 2019). Dandl et al. (2021) scrutinized this issue and explored the notion of request rejection in both ride-hailing and ride-pooling systems. They stated that rejecting the requests as early as possible is always good for passenger convenience and proved that this early rejection does not cause any disadvantage to the system. In their proofs, they demonstrated that a request which could not be matched with any vehicle in a certain batch period, thus to be rejected, will also not be matched with any vehicle in the next batches. They explored the proof in two cases. In the first case, they checked the non-availability of a vehicle: if there is no available vehicle at batch period n for a request r , there will also be no available vehicle at the next batch period $n+1$. In the second case, they checked the computing requests and bundles: if there was an available vehicle at batch period n for a request r , but this was assigned to another request/bundle due to better conditions, there will be no operational reason in the next batch period $n+1$ to swap request-vehicle assignments, or insert r to the pre-assigned bundle. Both cases have shown that delaying the rejection cannot make the system better later, thus the authors concluded that rejection of requests must be made as early as possible, bringing more passenger convenience (less frustration from long waiting for a rejection) and less computational complexity (fewer requests to be considered in the next batch periods).

2.4.1. Ride-Hailing Studies

In terms of passenger transport, it could be observed that the largest part of the literature involved ride-pooling; nevertheless, some works were done considering only ride-hailing. These works mostly concentrated on exclusive taxi systems, where sharing of the trip was not allowed, as in the case of ride-hailing. In the following, these ride-hailing studies are elaborated.

Kümmel (2016) dealt with the problem of dispatching taxis to passenger booking requests. He aimed to improve the dispatching strategies and used the stable marriage algorithm, which assigns and/or reassigns the requests to vehicles in batches at certain decision epochs. The requests in focus were both immediate and advanced bookings. In this study, taxis offered only exclusive trips, hence pooling was not the case. In contrast to the conventional taxi dispatching strategies such as a sequential first-come-first-serve assignment, he proposed a constructive reassignment, where the newly incoming requests were first assigned sequentially to give quick response to the passengers, but then the whole schedule (new as well as the previous requests) was torn down and constructed anew to reach a more efficient schedule. The total costs and total waiting time of the schedule before and after the reassignment were compared, and the more favorable one was accepted. The results indicated that the proposed methodology could serve more requests with the same taxi fleet, reduce waiting times and increase the operator profit.

With the vision of having autonomous taxi systems in the future, Dandl et al. (2017) implemented a microsimulation model of such a system in Munich using the traffic simulation software Aimsun. They had the research objectives of examining the impact of empty rides of autonomous taxis on the street network as well as investigating the effects of congestion on the fleet performance. Since the envisaged taxi system offered only exclusive trips, ride-hailing was the focus, and ride-pooling was not elaborated. A simple spontaneous assignment algorithm was used to match the requests with the vehicles, since faster responses to the passengers compared to a global assignment were then possible, and the authors did not want to focus on the operation strategy. In the model, passengers also had the chance to reject the offer of the autonomous taxi system if the estimated time of arrival did not meet their expectations. The most significant result of the study was that the one-to-one substitution of private trips with

autonomous taxi requests caused only a 1% increase in the network-wide delay of private vehicles due to induced empty rides.

Narayan et al. (2020a) highlighted the complementary effect of conventional public transport and on-demand systems, and focused on the integration of ride-hailing with public transport. In their methodology, besides the transport modes car, walking, bike and public transport, they introduced a ride-hailing system. The passengers had in total the following alternatives to use the ride-hailing: Either the whole trip was realized by ride-hailing, or the first- and last-mile were carried out by non-motorized modes (walk/bike) and/or by ride-hailing. The choice of these modes depended on a rule-based algorithm; only the public transport stops within a defined radius were taken into account, and depending on this distance, the passengers were obliged to use either the non-motorized mode or the ride-hailing. The multimodal route choice and assignment model were implemented in the agent-based simulation MATSim for the city of Amsterdam. Scenarios where conventional public transport and ride-hailing were offered as mutually exclusive, or integrated modes, were analyzed. Results revealed that ride-hailing was predominantly used for covering less than 30% of the trip length, indicating that it was mainly used as an access or egress mode to public transport. Also, the average waiting time of ride-hailing users could be reduced in complementary scenarios compared to the competitive scenarios.

Hyland and Mahmassani (2018) presented and evaluated optimization-based assignment strategies for autonomous ride-hailing vehicles. In a custom agent-based simulation framework, six traveler assignment strategies were tested, ranging from simple first-come, first-serve approaches to advanced optimization methods incorporating vehicle diversion and reassignment. Simulations of Chicago taxi trips showed that optimization-based strategies significantly reduced traveler wait times and fleet inefficiencies. The results highlighted the importance of advanced dispatch strategies, particularly when fleet resources are limited.

Iacobucci and Schmöcker (2021) dealt with the issue of rebalancing costs for ride-hailing in their study. As rebalancings are essential, they are at the same time too expensive for the operation due to empty rides. They had the idea to use dynamic pricing to control the demand and reduce rebalancing costs. In their work, they presented a dynamic pricing optimization tool for a ride-hailing service that was integrated with in-line predictive relocation. They indicated that the method could find dynamically the near-optimal fare for each origin-destination pair and each period of the day, maximizing the profit for the operator. In their numerical simulations in MATLAB, the results showed an increase in net profits of 8-11% compared to a constant pricing case at the near-optimal point, and a decrease of empty drive costs of 33%.

Another work focusing on the pricing aspects of ride-hailing was the study by Negro et al. (2021). They analyzed the cost composition of ride-hailing, and developed as well as quantified cost components for the case of a premium ride-hailing provider in Munich. The system was investigated for the two main stakeholders: (i) the on-demand service provider (providers like Uber, Lyft) and (ii) the fleet manager (owner of the drivers and vehicles), taking vehicle, operating, overhead, and platform costs into account. Based on a comprehensive literature review, the costs per passenger-km of the on-demand provider and the costs per vehicle-km of the fleet manager for various fleet composition scenarios (electric vehicles, autonomous vehicles, internal combustion engine vehicles, and their combinations) were calculated.

2.4.2. The Concept of Shareability

In the literature, some works were dedicated to elaborating the concept of shareability, i.e., which requests can share a trip with which other requests (Bilali et al., 2019a, 2019b; Santi et al., 2014). This

is a fundamental aspect of ride-pooling, as without trip sharing, only a ride-hailing system can be established.

Santi et al. (2014) introduced the notion of shareability network in their study. The idea here was to check whether the trips that appeared in a ride-pooling system could be shared with each other, or not. For this, a shareability network was constructed using the trips as a mathematical graph, and trips that could potentially be shared were connected in that graph with an edge. The shareability of the trips was decided in the following manner: “*Multiple trips are shareable if there exists a route connecting all the origin and destination points in any order such that each customer is picked up and dropped off with a certain maximum delay.*” Two main parameters were used in the study, (i) the maximum delay, and (ii) the number of trips that could be shared. The maximum matching of the trips, i.e., the maximum number of shared trips was set as the objective. This framework was applied to 150 million taxi trips of New York City, showing that the cumulative trip length could be cut by almost 40%.

Bilali et al. (2019b) studied the impacts of detour time, waiting time and boarding time on the shareability of requests. They modeled shareability shadows to find out which requests could be shared with each other. Based on nodes and edges in graph theory, edges were added between two nodes (requests) if they were shareable. At the end, a shareability index was calculated, which represented the percentage of requests that were shareable. The results showed that shareability increased with higher maximum waiting and detour time. Another finding was that the lower parameter between maximum waiting and detour time was the decisive one concerning the shareability index. In this study, no advanced booking was allowed, there existed only immediate requests where customers wanted to be picked up as soon as possible.

In a subsequent research, Bilali et al. (2019a) extended the previous framework with the allowance of advanced bookings. On the other hand, the boarding time parameter was neglected. A total of 36 scenarios were developed in total, with varying waiting, detour, and reservation time. According to the results, it was found out that introducing a reservation time improved shareability, especially for low trip density and restrictive systems where maximum waiting or detour time was low. Another finding was that the impact of detour time was generally higher than that of waiting time.

2.4.3. Simulation Platforms

Ride-hailing and ride-pooling studies focusing on the vehicle-request matching algorithms are modeled in various simulation platforms in the literature. As in general transportation studies, the main platforms used here are: (i) microscopic traffic simulations, (ii) macroscopic traffic simulations and (iii) agent-based simulations. Besides these, some studies utilized custom simulation frameworks that were developed by the authors themselves for the specific research objectives. A review of the relevant literature revealed that very few studies utilized microscopic platforms, most likely due to the high effort required to model very large areas with several on-demand vehicles and requests. Moreover, the well-known existing microsimulation tools do not include an on-demand mode to date; thus, an external control of the tool that exactly fits its data and object structure becomes inevitable, yet still cumbersome. Furthermore, these studies did not give sufficient details about the implementation in the employed microsimulation tool.

While macroscopic tools are moderately preferred for these studies, the highest number of publications has occurred in the field of agent-based simulations. Agent-based simulations do not need a high level of detail and can nonetheless deal with several vehicles and requests as agents for larger areas (Bonabeau, 2002).

In the next sections, the ride-hailing and ride-pooling studies will be investigated with respect to the utilized simulation platforms.

An overview of the used simulation platforms for the selected studies can be found in Table 2.1 and Table 2.2.

2.4.3.1. Microscopic Traffic Simulation

One of the first on-demand studies that involved microscopic traffic simulation was Cortés et al. (2005). They developed a general framework in the Paramics tool to model a flexible transit system called High-Coverage Point-to-Point Transit (HCPPT), allowing the sharing of trips. They adapted and enhanced the software to address specific requirements for transit simulation that were not typically supported by standard microsimulation tools.

Dandl et al. (2017) used the microsimulation tool Aimsun to implement an autonomous taxi system in Munich, however only with ride-hailing. They emphasized the importance of using microsimulation due to the fact that the travel times required to calculate routes and waiting times could be gathered from the network links in the microsimulation, unlike many studies in the literature that preferred to use constant travel times mostly because of their simplicity or the lack of data.

Tang and Armellini (2021) used the microsimulation tool SUMO and proposed the metaheuristic algorithm ant colony optimization to solve the DARP with time windows and capacity constraints. The requests were composed of the origin stop, destination stop, and either the earliest departure or latest arrival time. As requests were revealed one-by-one, the algorithm solved a new instance of the DARP each time and formed the updated tours of the vehicles. The objective function involved the minimization of the total travel time of vehicles. A case study was conducted in Elde region of Germany with 2 vehicles and 10 requests, where vehicles started and ended their tours in a depot, and picked up/dropped off the requests in between. The requests were generated randomly in terms of their spatial and temporal parameters. To validate the algorithm, the same constellation was simulated with an exact algorithm developed by one of the authors, and the results were compared. It was found that the proposed algorithm was able to nearly find the same routes as the exact algorithm produced, yet with a much more acceptable computational time.

Another example of using microsimulation is the study by Armellini (2021), who employed SUMO as well to model a ride-pooling system in Brunswick, Germany, but only in a static manner where all requests were known in advance. For this purpose, an external Python tool was developed that allowed communication with SUMO and calculated the vehicle routes in an optimized way. The work of Alonso-Mora et al. (2017) was taken as the base for the matching algorithm.

2.4.3.2. Macroscopic Traffic Simulation

Friedrich et al. (2018) presented a matching algorithm which could be integrated into existing macroscopic travel demand models. This algorithm was described as a greedy algorithm and did not guarantee an optimal solution for the matching problem. It focused on ride-pooling with two main stakeholders: the demanders (passengers desiring to travel in an existing car trip) and the suppliers (car drivers willing to take other demanders). The aim of the matching was to assign the demanders to suitable suppliers based on the similarities of their routes and departure times. The matching algorithm was applied to the PTV Visum macrosimulation platform. Here, every link in the Visum model was assigned to a zone. The routes of each supplier and demander were then converted from a link sequence to a zone sequence (path). If the demander's path was identical to the supplier's (full match), the matching was performed. For the remaining demanders, the second step of the matching was carried out. For this step, it was sufficient if the demander's path was only partly covered by the supplier's path

(partial match). After the matchings, the sharing capacities of the supplier and the unsatisfied ride-pooling demand were updated. The results for the simulation of the Stuttgart region showed that the entire travel demand of all motorized modes could be transported by 7% of the current car fleet with 65% of the current vehicle distance traveled, if all travelers were willing to either use ride-pooling vehicles with 6 seats or traditional rail.

Richter et al. (2019) took the algorithm of Friedrich et al (2018) as the base and adapted it again in Visum for the case where the demanders of ride-pooling were served by automated vehicles, and not by suppliers. The matchings took place according to the similarities of zone sequences of the demanders' trips. For this, firstly the paths were sorted in descending order by their length. The trip with the longest path was assigned to a vehicle. Then, the next longest trip was checked to see if its zone sequence was fully included in the zone sequence of the previous trip. In that case, assignment to the same vehicle occurred, provided that the vehicle capacity restrictions were not violated. Otherwise, another vehicle was assigned. This procedure was repeated for the whole path list of demanders. A test scenario in the Stuttgart region showed that the number of vehicles needed could be reduced to 25% of the number of cars used at that time.

Hartleb et al. (2021a) and Hartleb et al. (2021b) focused on scheduling on-demand vehicle fleets in macroscopic simulation. They built their research upon the works of Richter et al. (2019), who generated ride-pooling trips which by then had fixed origins, destinations and departure times. As a next step, these trips had to form vehicle blocks that represented the tours of the vehicles. The authors modeled the vehicle scheduling problem as a flow problem and described it on a mathematical graph, both temporally and spatially. They utilized a heuristic that went through all nodes one by one with the aim of minimizing the number of vehicles used. Even though the algorithm could not guarantee the optimal solution, it provided a solid approach for networks with a large number of zones and time intervals. The algorithm was demonstrated for two scenarios: (i) one for car sharing in Stuttgart, and (ii) another for an e-scooter sharing on the campus of the University of Stuttgart. For each scenario, the number of vehicles and empty drives were calculated.

Thomsen (2023) extended the algorithm of Friedrich et al. (2018) and Richter et al. (2019) by introducing population centers as pick-up locations, instead of traffic analysis zones. The reason for this was that her study was in a nationwide scale, and at that scale, the zones were much larger than in previous studies, which would lead to unrealistically high matching rates. In this way, the path trajectories could be better differentiated. She applied this methodology to the German national transport model (DEMO) in the Visum macrosimulation and assessed the impacts of ride-pooling on vehicle occupancy, mileage, detour times, and required fleet size.

Richter and Hess (2021) investigated the question of at what price level it would be profitable to offer a robo-taxi fleet. In the Visum macrosimulation and PTV MaaS Modeler, 500 such taxis were added to the existing transport system of Zurich. The simulation results were combined with a comprehensive cost analysis. The results indicated that with sufficient utilization, robo-taxis could establish themselves below the price level of many conventional mobility solutions.

Zubair et al. (2025) aimed to optimize the fleet size range of ride-pooling services for a given population density and service area in suburban regions. The analyses were performed in the Visum macrosimulation tool for the northern suburbs of Munich. Furthermore, the matching algorithm of the tool itself under the DRT (Demand Responsive Transit) module was utilized. In the study, nine different service areas were defined according to their size and population density. For various demand quantities and levels of service (percentage of trips served), outputs for average waiting time, average detour factor, and appropriate fleet size range were produced. Based on these results, fleet size range recommendations

were made per service area. The study overall underscored the importance of scaling fleet size proportionately to the specific demand and population density conditions to ensure an efficient ride-pooling service.

2.4.3.3. Custom Simulation Framework

In a custom simulation framework environment, Alonso-Mora et al. (2017) introduced a new batch assignment algorithm concerning the vehicle-request matching. This algorithm was applied to nearly 450,000 daily taxi trips in New York City, which were operated originally by 13,500 taxis. The taxi trips were converted to ride-pooling requests and were processed in batches of 30 seconds. Several scenarios with varying fleet sizes, vehicle capacities, maximum waiting times and maximum detour times were investigated. The results showed that only 3,000 vehicles with a capacity of 4 would be sufficient to serve 98% of the requests within acceptable waiting and detour times. The validation of the algorithm proved it to be highly scalable for other use cases as well.

Liu and Samaranayake (2022) presented two speed-up techniques, search space pruning and input/output reduction, to improve the computational performance of ride-pooling assignment algorithm of Alonso-Mora et al. (2017). Moreover, they developed a proactive rebalancing method for more efficient matching. The results of case study in Manhattan were significant: the computation time could be reduced by almost 97%, and the service rate could be raised by 5%.

Ghandeharioun and Kouvelas (2023) focused on developing a matching algorithm for on-demand ride-pooling. They formulated this algorithm as a combinatorial optimization problem. Some heuristics were used to reduce the computational complexity. A case study was carried out in Manhattan, New York using the taxi data for a typical day between 7-9 a.m. Dynamic congestion was also implemented by regularly updating the network's road segments' travel time during the simulation horizon. The objective function and the operational parameters were varied for the analyses, which used certain indicators such as the service rate, sharing rate, average waiting time, average detour time and idle time rate. The experiments demonstrated that utilizing ride-pooling with a fleet smaller than 15% of the current number of active taxis in Manhattan could accommodate 80% to 98% of the requests.

Zhou and Roncoli (2021) touched upon the issue of congested networks in ride-pooling and the selection of routes. They mentioned that most of the studies in this field used static travel times or historical travel time data, and ignored the congestion caused by ride-pooling vehicles. To assign the ride-pooling vehicles while considering network traffic, the authors proposed a framework that incorporated time-dependent (dynamic) link travel times into the vehicle-request assignment. Furthermore, they formulated an optimal assignment problem that considered multiple path options and took into account the congestion potentially caused by assigned routes. The results of the simulations on a very small-scale custom network revealed that the proposed model could reduce congestion, average passenger waiting time, and average passenger travel time compared to static approaches.

One of the few studies that incorporated the flexible roles of the vehicles is Atasoy et al. (2015). In their study, they modeled a flexible mobility on-demand (FMOD) system to serve the requests with personalized services. Contrary to the most studies, the vehicles in the system could be used flexibly as different service types: they could act as a taxi (ride-hailing), shared-taxi (ride-pooling), or mini-bus (fixed route) dynamically, i.e., their service types could change during the operation. Furthermore, passengers were also flexible in that they were provided with a menu of choices from which they could make a selection based on their preferences in terms of travel time, price, and comfort. In this custom framework, firstly a feasible product set was generated where each product consisted of a vehicle, a service type, scheduled departure and arrival time, and a price. Then, a choice-based assortment optimization was conducted in order to create an option list for the passenger, which included one option

per service type. To determine the scheduled departure and arrival times, either new schedule blocks were created where necessary or the existing blocks were modified in the case of shared-taxi and mini-bus service types. For the shared-taxi case, an insertion algorithm was utilized. The results have shown that the dynamic allocation of vehicles to different service types provided significant benefits over static allocation.

2.4.3.4. Agent-based Simulation

Most of the ride-hailing and ride-pooling studies in the literature were conducted in agent-based simulation platforms. In this section, the most widely used agent-based simulators are explored with exemplary studies in the field of ride-hailing and ride-pooling.

Von Mörner (2018) employed a self-developed agent-based simulation environment in AnyLogic. In his research, he examined the potential of ride-pooling by autonomous vehicles in rural areas as a replacement for traditional public transport. The impacts of different parameters on the pooling potential were also explored. He utilized a sequential rule-based algorithm to dispatch the vehicles. 1,588 on-demand stops were used in the simulation, the routing between them was calculated offline using OpenStreetMap. This, however, brought the disadvantage that spontaneous route changes could not always be considered since an online routing calculation was not possible. Two dispatching strategies were mainly discussed, one with high efficiency aiming to pool requests as much as possible, and the other one with high passenger comfort aiming to create as few detours as possible. Different scenarios concerning the travel demand, vehicle capacity, maximum waiting time, maximum detour ratio and the allowance of advanced booking were analyzed. The main results of the study showed that ride-pooling with autonomous vehicles in rural areas replacing classic public transport could be economically feasible, yet various other dispatching strategies had to be developed for more efficiency.

In Kucharski and Cats (2021) as well as Kucharski and Cats (2022), an agent-based simulator called MaaSSim was presented that reproduced the transport system dynamics by two kinds of main agents: travelers requesting to travel from one point to another, and drivers offering rides. Agents could make their own decisions, for instance the travelers could select the transport mode or reject an incoming offer, whereas the drivers could opt out of the system or reject incoming requests. The key contribution of these studies was described as advancing the state-of-the-art agent-based simulation models towards two-sided mobility platform systems to develop, test and simulate platform operations under various settings. In the default case of the simulator, incoming travel requests were matched with the nearest idle vehicle, yet the authors mentioned that this matching methodology might be replaced with another algorithm easily, thanks to the modular structure.

Wilkes et al. (2021b) developed an agent-based simulation framework called mobiTopp also considering the ride-pooling transport mode. According to mobiTopp, various transport modes were available for the agents, including ride-pooling. For the mode choice of agents, the parameters resulting from ride-pooling, such as waiting time and travel time, were required. To calculate these parameters, mobiTopp involved a ride-pooling sub-module using a spontaneous assignment technique, which was the focus of this paper. In this sub-module, as soon as new requests appeared in the system, they were inserted into the current tours of the vehicles. Among different insertion options over all available vehicles, the option with the least increase in vehicle travel time was selected. A prefiltering mechanism for the options was used, considering the waiting time and detour time of requests, as well as the capacity constraint of vehicles. A case study in Stuttgart was conducted with fleet sizes of 100, 200, and 400.

A similar study was conducted in Wilkes et al. (2021a). Here, the authors employed mobiTopp with a more sophisticated ride-pooling sub-module (Kagerbauer et al., 2021). In this module, a two-step assignment algorithm was introduced. In the first step, spontaneous assignment was used in order to

respond to the requests as quickly as possible. If the request actually chose the ride-pooling module, a batch assignment was applied in the second step, where several such requests within a defined time period (batch) were assigned to vehicles at the same time. The implemented objective function rewarded serving as many requests as possible while minimizing the overall service time of vehicles. The framework was applied in Eggenstein-Leopoldshafen, Germany for more than 70,000 trips in 24 hours. The available modes were public transport, car, walking, cycling, and ride-pooling. The impact of ride-pooling fare on modal split, and the impacts of the ride-pooling parameters (maximum waiting time, maximum detour ratio, and fleet size) on vehicle occupancies were investigated.

Fiedler et al. (2018) evaluated the potential of ride-pooling to reduce traffic congestion and vehicle mileage in the agent-based simulation framework AgentPolis. The matching strategy consisted of a spontaneous assignment which used an insertion heuristic algorithm. Results from a case study in Prague's morning peak traffic showed that ride-pooling significantly enhanced system efficiency, reducing total vehicle mileage by 40% compared to current private car usage. Average vehicle occupancy increased from 0.7 to 2.7 passengers per vehicle.

Using the well-known agent-based simulation environment MATSim (Horni et al., 2016), Narayan et al. (2017) aimed to analyze the competition capability of an on-demand system (ride-hailing and ride-pooling) against other existing modes such as car, walking, and conventional public transport. They developed and compared three scenarios, one with the base modes, a second one where a ride-hailing system was added to the base case and a third one where a ride-pooling system was added to the base case. They highlighted the impact of fleet size and on-demand fares on the modal split and average waiting times. The findings suggested that increasing the fleet size significantly reduced waiting times and encouraged mode shifts from personal cars and conventional public transport to on-demand systems. Additionally, higher cost ratios led to a decline in the usage of on-demand systems, with ride-pooling services maintaining better user retention compared to ride-hailing.

Again in MATSim, Narayan et al. (2022) investigated the potential scalability of on-demand systems to replace car and public transport trips. A total of six different scenarios were developed, consisting of those where ride-hailing and ride-pooling services replaced private car, public transport, and both. The fleet size of the on-demand service was taken as the system parameter and its impact for the operator perspective (service efficiency) as well as the user perspective (level of service) was elaborated. The authors found that there was a significant share of unsatisfied demand for the car trips replacement scenario, whereas this was just the opposite for the public transport trips replacement scenario: even a relatively low fleet size could handle these trips. Moreover, ride-pooling scenarios turned out to be more efficient in terms of vehicle mileage, empty drive ratio, and passenger waiting time compared to ride-hailing. However, due to the long detours in ride-pooling, the average travel time was better in ride-hailing scenarios.

Narayan et al. (2020b) developed a model in MATSim to determine the optimal fleet size for a mixed ride-hailing and ride-pooling system under elastic demand conditions. The research investigated the differences in fleet size requirements from the perspective of a transit planning authority (agency) and a service provider (operator), focusing on whether these stakeholders prioritize system-wide efficiency or profit maximization, respectively. Results indicated that the optimal fleet size for the agency was larger than what an operator would choose, as the agency aims to minimize user travel costs, while the operator focuses on profit maximization. Another important result of the study was that operating a ride-hailing service was more profitable for both the agency and operator compared to ride-pooling. An additional fare sensitivity analysis revealed that keeping ride-hailing fares comparable to public transport resulted in the most efficient system performance.

Bischoff et al. (2017) implemented a spontaneous assignment of requests to vehicles using MATSim. An insertion heuristic algorithm was used, where the pick-up and drop-off points of an incoming request could be inserted into the current stop sequence of vehicles. These insertions were applied to each vehicle, and the option leading to the least increase in vehicle service time was selected. Rejections of the requests were possible if all options violated the predefined rules of vehicle capacity, maximum waiting time, or maximum detour time. Simulations in MATSim for 12 hours, comprising more than 27,000 requests and 4,000 vehicles in Berlin, were performed. Vehicle capacities and maximum detour times were altered for each simulation run, whereas the maximum waiting time was assumed to be a constant value of 15 minutes. These runs were compared to the base scenario, which involved a ride-hailing operation, and then to each other. The authors concluded from the results that ride-pooling could reduce the total mileage by 15-20%. They also indicated the need for an operator to weigh trade-offs between passenger comfort and the fleet performance, since the utilized objective function that considered only the operator perspective led to longer travel times.

Fagnant and Kockelman (2018) introduced a ride-pooling framework in MATSim as well. They used spontaneous assignment for incoming requests and checked the vehicles in the vicinity of the request as matching potentials. They introduced five conditions, such as maximum detour ratios or earliest pick-up time of the new request, that had to be satisfied for a vehicle to get short-listed at all. All combinations of pick-ups and drop-offs were tested in this way. Finally, the nearest vehicle to the new request satisfying all the conditions was chosen as a match. A case study in Austin, USA, was conducted with different maximum detour ratios (1, 1.2, 1.3, 1.4), as well as varying demand and fleet size.

Another study that used MATSim is the work of Ruch et al. (2020). The authors compared ride-pooling with ride-hailing to evaluate between operational efficiency and service quality. The matching algorithm was based on the work of Alonso-Mora et al. (2017) which formulated the assignments as an integer linear programming problem. Results indicated that while ride-pooling reduced mileage moderately, it also increased total travel time and often resulted in low vehicle occupancy.

Besides MATSim, which has been widely used in on-demand studies, some researchers have built their own agent-based simulation tool. Engelhardt et al. (2022a) introduced such a self-developed open-source agent-based simulation tool “FleetPy” with its key features, sub-modules, and implementation techniques. This tool was developed explicitly to model ride-pooling services at a high level of detail. The model was validated in a case study conducted with Manhattan taxi trips, analyzing the total vehicle mileage and service rate. In another scenario, customers’ ability to reject the offer was modeled as well with a simple linear probabilistic distribution. This FleetPy tool was employed in many studies of the authors to analyze various aspects of ride-pooling, as elaborated in the next parts.

Engelhardt et al. (2019) aimed to quantify the benefits of ride-pooling through a case study with Munich private trips in FleetPy. Their study was based on the matching model of Alonso-Mora et al. (2017), yet they made some modifications to the system. Firstly, customers were allowed to reject the provided offer as well, based on a simple probability calculation depending on the waiting time. Secondly, the requests that were rejected by the operator did not vanish immediately from the system; instead, they tried again at the next batch period, thus had two chances in total. Thirdly, they modified the objective function and considered the minimization of the vehicle mileage, hence focusing more on the operator perspective. The results indicated that the rejection rate decreased with bigger fleet sizes and relatively higher demand. Moreover, vehicle kilometers could be reduced by up to 18%, predominantly on the major roads with higher demands. Conversely, ride-pooling had a negative effect on traffic in case of low demand, increasing vehicle kilometers. The authors proposed in their final words to employ microsimulation to model ride-pooling for further research, in order to observe the global change of traffic flow in the network with a push-pull effect (from private trips to ride-pooling).

In Engelhardt et al. (2020), the authors extended their previous work in FleetPy and added heuristic rules to improve the computation time while sacrificing some efficiency. Their heuristics were based on limiting the number of vehicles that came into question for matching a request via three different rules. Moreover, they also modeled a scenario with spontaneous assignment to compare with the batch assignment scenarios. Different from the previous study, they removed the requests rejected by the operator immediately, thus did not keep them for a second chance. Moreover, they did not allow the customers to make a rejection decision (customers always accepted the offer). It could be observed from the results that the applied heuristics could speed up the algorithm by a factor of 8, while keeping the service rate almost constant. Another finding was that the batch assignment strategy raised the number of served requests by 8%, compared to the spontaneous assignment. The authors argued that this was expected since the spontaneous assignment possessed the drawback of neglecting the benefits of global optimization and possible reassignments.

Engelhardt and Bogenberger (2021) explored the impacts of boarding locations on the system efficiency in ride-pooling. Unlike previous studies, which accepted the requests from certain network points, it was assumed that the requests came from any location in the network. The aim of the study was then to find the best boarding/alighting location for each request within a 300-meter radius. The authors introduced the concept of “imaginary request”, meaning that a real request could have several imaginary requests, i.e., combinations of the possible pick-up and drop-off points within the given radius. On the other hand, as this led to a drastic increase in computational complexity, a two-step heuristic was employed: (i) firstly, only the imaginary requests that have the closest locations to the real request were considered, and (ii) secondly, after vehicles were matched to the requests, each assigned vehicle tour was optimized with respect to possible pick-up and drop-off locations. Additionally, the objective function was modified in a way that it took both the operator and the passenger perspective into account. Results revealed that 4% more served requests and 4.8% fewer kilometers could be achieved with dynamic boarding locations, while the walk times and aggregated travel times of customers increased only marginally.

While many studies in ride-pooling assumed that there is only one entity that possesses all the vehicles in the whole system and decides the assignments alone, there can also be cases where multiple ride-pooling operators exist and compete with each other to serve customers. Engelhardt et al. (2022b) touched upon this issue and presented a model to assess the effect of competition and cooperation among multiple ride-pooling providers, and the impact of demand fragmentation. Apart from the customers and operators, they introduced the broker concept whose task was to collect trip offers from the operators and forward them to customers, acting as an interface between them. In FleetPy framework, they developed four scenarios in total where (i) a single operator got all the requests and decided alone, a so-called base case, (ii) two independent operators that did not communicate with each other received randomly half of the requests and responded, (iii) both operators made offers to each request, and users decided eventually which to choose with based on a user utility function, and (iv) both operators made offers to each request, and the broker decided alone which to choose based on a system utility function. For the matching, initially a spontaneous assignment and subsequently a batch assignment were used. The results indicated that the broker setting was proven to be beneficial for all stakeholders; it led operators to generate higher profits and less vehicle mileage, regulators to have increased pooling efficiency, and customers to obtain higher service rates. Nevertheless, the customers experienced a slight increase in waiting and travel times, yet within acceptable levels.

Engelhardt et al. (2024) focused on the rebalancing problem in ride-pooling systems. The aim of the study was to develop a model that proactively relocated unused vehicles within the operational area, in order to optimally cover the expected future demand. The assignment algorithm was based on the sampling method and employed integer linear programming. Results of the case study carried out in

FleetPy on Chicago TNC data revealed that the proposed algorithm could almost double the number of served requests compared to a service without rebalancing. In addition, it increased the pooling efficiency while reducing empty rides and passenger waiting times.

Lastly, Zwick et al. (2022) compared the two agent-based simulation approaches, FleetPy and MATSim, by coupling them with the mobiTopp mode choice simulation framework. One essential difference mentioned in this comparison was that the MATSim approach required multiple iterations. It was shown by a case study in Hamburg with scenarios of different pricing levels, that both systems were capable of generating realistic results and assessing ride-pooling schemes.

2.4.4. Matching Strategies

The core of an on-demand mobility platform is the algorithm used to match the incoming requests with vehicles. There are mainly two matching strategies employed in the previous studies which differ in terms of the execution point of time.

The first strategy is known as spontaneous assignment. In this strategy, the incoming requests are processed immediately as soon as they arrive in the system. As the requests are matched to the vehicles in the sequence of arrival, this strategy is also called sequential or first-come-first-serve strategy. Here, many vehicles are checked for one request; hence, a one-to-many relation can be observed.

The second strategy is known as batch assignment. In this strategy, the incoming requests are processed within certain time periods, which are called batches. The new requests accumulate in every batch, and a global assignment is conducted at the end of the batch. The requests are not processed immediately and have to wait until the end of the batch period for a matching. Here, many vehicles are checked for many requests, hence a many-to-many relation can be observed.

While spontaneous assignment can give quick responses to the request and is computationally less expensive, batch assignment can deliver more optimal results as assigning several requests at once often leads to better matchings.

In the literature, it is observed that some studies focused on spontaneous assignment (Fagnant & Kockelman, 2018; Kucharski & Cats, 2021; Narayan et al., 2020a; Tang & Armellini, 2021; von Mörner, 2018), whereas others were based on a batch assignment strategy (Engelhardt et al., 2019; Engelhardt & Bogenberger, 2021; Kümmel, 2016; Ruch et al., 2020). Spontaneous assignment was sometimes applied as an insertion heuristic, where the stops of the new request were inserted into the current stop sequence of the vehicles (Bischoff et al., 2017; Fiedler et al., 2018; Wilkes et al., 2021b). This approach, however, limits the search space as it claims from the beginning that the current stop sequence of the vehicles cannot be changed.

Regarding the batch assignment strategy, a substantial work has been published by Alonso-Mora et al. (2017) in the field of dynamic ride-pooling focusing on the vehicle-request matching algorithm. In this study, assignment of immediate requests was pursued under the hard constraints of (i) vehicle capacity, (ii) maximum waiting time of requests, and (iii) maximum delay time of requests. The authors claimed that traditional approaches such as that of Cordeau (2006), were not capable of solving large problem instances in real-time since modeling the network edges explicitly in an integer linear programming (ILP) would be computationally very expensive in such cases. Instead of this, they proposed a framework based on the work of Santi et al. (2014) using the concept of the shareability network. This framework was based on three important steps:

- In the first step, it was computed (i) which requests could be pairwise combined, given the hard constraints about requests, and (ii) which vehicles could serve which requests individually, given their current passengers. The resulting combination pattern was named the RV-graph.
- In the second step, the RV-graph was extended to find more groups of requests (named as trips) that could be combined and assigned to a vehicle while satisfying the constraints. At this point, a special approach was applied to ensure computational efficiency. Matching of a trip to a vehicle was only explored if all the subsets of the trip could also be matched with that vehicle. Thus, an incremental check of request groups was applied. For each trip-vehicle combination, a cost was calculated involving the delay time (waiting time plus the detour time) of the relevant passengers. The resulting combination pattern was named the RTV-graph.
- In the RTV-graph, several trips of varying size were generated that could include a particular request. Additionally, a vehicle could service more than one trip. However, in reality, the system would only allow that (i) each request exists in one assigned trip (or zero in case of rejection), and (ii) each vehicle gets maximum one trip assigned. In the third step, these requirements were assured by an assignment algorithm based on ILP. The objective function of ILP was set as minimizing the total assignment costs and the number of rejected requests. In order to keep the rejections to a minimum, a very high penalty cost was incorporated in the objective function.

As the proposed model provided sufficiently quick solutions to the matching problem, it was regarded as a neat approach for similar ride-pooling problems of a dynamic real-time nature. Many studies took the model of Alonso-Mora et al. (2017) as a base and modified, extended, or improved it, e.g., with heuristics (Armellini, 2021; Engelhardt et al., 2020, 2024; Liu & Samaranayake, 2022; Ruch et al., 2020).

Besides spontaneous and batch assignment, a hybrid approach has also been identified in some works. These theoretically used a batch assignment approach, i.e., the new requests were obtained per each batch, yet they were not processed at the same time, as would be the normal case in batch assignment. Rather, a one-by-one processing was applied, resulting in a hybrid strategy (Friedrich et al., 2018; Richter et al., 2019; Thomsen, 2023). For example, in Ghandeharioun and Kouvelas (2023), for each new request, one vehicle was pre-matched according to the objective function costs. Then, these requests were sorted according to the objective function values. Next, starting from the top of the sorted list, the requests were matched to the pre-matched vehicle if that vehicle was not pre-matched before. If that vehicle had already been pre-matched, those requests went through another iteration to be matched with the remaining vehicles. This process was repeated until all the requests in the batch found a matching vehicle or were rejected.

Lastly, a spontaneous assignment followed by a batch strategy was also observed in some studies, mostly in order to give the passenger a response as soon as possible, and then subsequently consider it in the global batch assignment if the passenger accepts the offer (Engelhardt et al., 2022a; Engelhardt et al., 2022b; Wilkes et al., 2021a).

An overview of the matching strategies for the selected studies can be found in Table 2.1 and Table 2.2.

2.4.5. Objective Function Perspectives

The objective function of the matching strategy can have two perspectives, reflecting the two main stakeholders of the on-demand mobility system. An operator's perspective ensures that the interests of the operator side are taken into account. This can be, for instance, minimizing vehicle mileage (Engelhardt et al., 2019), minimizing the total vehicle travel time (Wilkes et al., 2021a) or maximizing

the operator profit (Atasoy et al., 2015). In these cases, the passenger interests are not represented directly and may be sacrificed for the sake of the operator. Although the objective function may not include passenger perspectives, some limitations on the passenger parameters can still prevent the emergence of options with highly unfavorable outcomes for passengers. These limitations are typically the maximum waiting time, maximum detour time, or maximum detour ratio (Fiedler et al., 2018; Tang & Armellini, 2021; Wilkes et al., 2021b).

On the other hand, a passenger's perspective in the objective function ensures that the interests of the passenger side are primarily taken into account. Minimizing the waiting time, detour time, or total travel time (from pick-up until drop-off) of passengers are the typical objectives in this case (Alonso-Mora et al., 2017; Dandl et al., 2017; Narayan et al., 2022; Ruch et al., 2020).

The primary aim of ride-hailing and ride-pooling services is to serve as many passengers as possible and apply rejections only when absolutely necessary. In batch assignment, the objective functions therefore often involve a large penalty value to hold the number of rejections as low as possible, regardless of their perspective. In spontaneous assignment, this is not necessary, since the requests are processed one by one, sequentially.

Only in a few studies, both the operator and passenger interests are taken into account in the objective function (Engelhardt et al., 2022b; Engelhardt & Bogenberger, 2021; Kucharski & Cats, 2022; Kümmel, 2016). This is certainly a fair approach, as none of the stakeholders remains unconsidered in the vehicle-request matchings. Moreover, placing more emphasis on one side is also possible in this approach, by using different weightings in the formulation.

An overview of the objective function perspectives for the selected studies can be found in Table 2.1 and Table 2.2.

2.4.6. Key Performance Indicators

An extensive review of the literature revealed that the key performance indicators (KPI) to assess the developed models are numerous and diverse. These can be relevant to either the operator or the passenger side. Some notable metrics concerning the passenger side are the mean waiting time (Kucharski & Cats, 2022), mean detour time (Thomsen, 2023), mean detour ratio (Zubair et al., 2025) and mean in-vehicle travel time (Narayan et al., 2020b). In contrast, for the operator side, the mean mileage (Wilkes et al., 2021b), total mileage (Richter et al., 2019), mean occupancy (Fiedler et al., 2018), total vehicle time (Tang & Armellini, 2021), empty mileage ratio (Narayan et al., 2020a), operator profit (Atasoy et al., 2015) and mileage index (Bischoff et al., 2017) are the most frequently used KPIs. Mileage index is calculated as the ratio of benefits to costs, i.e., the sum of the direct trip distances of served passengers (benefit) divided by the total mileage (cost) (Liebchen et al., 2020).

A typical KPI used in many studies is the service rate, which shows the ratio of the number of served requests to the total number of requests. This metric is essential, as serving all requests in on-demand systems cannot be guaranteed, and some requests may need to be rejected. Nevertheless, the operation strives to provide a high service rate (Armellini, 2021; Engelhardt et al., 2022a; Friedrich et al., 2018; Liu & Samaranayake, 2022; von Mörner, 2018).

Sharing rate is another metric used in certain studies. It is calculated as the number of requests that share a ride, divided by the total number of served requests, thus representing the shareability of the model in ride-pooling cases (Alonso-Mora et al., 2017; Fagnant & Kockelman, 2018; Ghandeharioun & Kouvelas, 2023; Ruch et al., 2020). Furthermore, modal split has also been observed as a metric,

especially in the works where a modal choice of the passengers was possible (Atasoy et al., 2015; Narayan et al., 2017, 2020a, 2020b; Richter et al., 2019).

An overview of the key performance indicators for the selected studies can be found in Table 2.1 and Table 2.2.

2.5. Summary

The comprehensive literature review conducted in this chapter provides valuable insights for the modeling of an on-demand ride-pooling system in a microsimulation environment.

With the emergence of innovations in transportation systems, on-demand mobility systems have gained a crucial role in today's passenger transport concepts. Different types of these on-demand systems were explored in detail in Section 2.1. Among those, ride-hailing and ride-pooling have been identified as the most relevant for this thesis.

It is evident that for ride-hailing and ride-pooling, calculating the shortest routes between different network points is very important since vehicles have flexible routes and can change their routes at any time (e.g., due to detours in pooling). In Section 2.2, various well-known algorithms concerning shortest route calculation were addressed and explained in detail. The Dijkstra's algorithm was selected for the application in this thesis, since the distance and path of the shortest routes can be obtained easily for all vertex pairs. More details on this application in the microsimulation can be found in Section 4.2.2.3.

One of the aims of this thesis is to develop algorithms for an on-demand system where requests are revealed in the system in real-time and require immediate transportation. According to the definition of problems and their classifications addressed in Section 2.3.1, such a system can be regarded as a dynamic-deterministic DARP. Moreover, the classical static DARP, as presented in Section 2.3.5 appears to be suitable for the modeling of vehicle-request matchings in batch assignments; however, its applicability is limited primarily to small problem instances and static problems, and should therefore be considered with caution.

The review of simulation platforms for ride-hailing and ride-pooling studies in Section 2.4.3 sheds light on the infrequent use of microsimulation. The main reasons for this are the significant effort required to model larger areas with multiple on-demand vehicles and requests, as well as the technical limitations of well-known existing microsimulation tools. This has been identified as a research gap and forms one of the main pillars of this thesis, particularly through the use of the Vissim microsimulation tool. It should also be noted that the case study area in this work is limited to the University of the Bundeswehr Munich campus (see Section 4.1), which reduces the effort required to build such a model in microsimulation. In contrast, many studies in the literature dealt with larger networks such as the cities of Munich, Stuttgart, Berlin, Amsterdam, Chicago, or New York.

Concerning the matching strategies discussed in Section 2.4.4, the algorithms can be broadly categorized into two types: (i) spontaneous assignment and (ii) batch assignment. Spontaneous assignment is sometimes implemented as an insertion heuristic, where the stops of a new request are inserted into the current stop sequence of vehicles. However, this approach limits the search space by assuming from the beginning that the existing stop sequence cannot be rearranged. A more general approach would allow all stop permutations, enabling the current vehicle stop sequence to be modified. Regarding batch assignment, the algorithm introduced by Alonso-Mora et al. (2017) was frequently cited and adopted by several studies, and is also considered suitable for application in this thesis, with certain modifications.

The objective function perspectives outlined in Section 2.4.5 revealed that very few studies considered both operator and passenger interests. However, incorporating both perspectives is a fair and balanced method, as it ensures that no stakeholder is neglected in the vehicle-request matching process. This approach also allows for flexibility, as greater emphasis can still be placed on one side by applying different weightings in the formulation. For these reasons, accounting for both perspectives in the objective functions of the matching algorithms proposed in this thesis is considered appropriate.

Lastly, Section 2.4.6 highlighted that a wide range of key performance indicators (KPIs) has been used in previous studies, depending on each study's objective. The service rate is definitely the most important metric, as on-demand systems aim to fulfill as many requests as possible. In addition, this thesis employs both vehicle-oriented KPIs, such as mileage, occupancy, empty mileage ratio, and mileage index, as well as passenger-oriented KPIs including waiting time, detour time, and sharing rate.

Table 2.1 Overview of ride-hailing and ride-pooling studies involving matching strategies.

Paper	Simulation Platform	Matching Strategy	Objective Function Perspective	Key Performance Indicators (KPI)
Atasoy et al. (2015)	Custom framework	Spontaneous	Operator	Service rate, Operator profit, Modal split
Kümmel (2016)	Custom framework	Batch	Passenger & Operator	Service rate, $\emptyset$ waiting time, $\emptyset$ mileage, $\emptyset$ taxi profit
Alonso-Mora et al. (2017)	Custom framework	Batch	Passenger	Service rate, Sharing rate, $\emptyset$ mileage, $\emptyset$ waiting time, $\emptyset$ detour time
Bischoff et al. (2017)	Agent-based (MATSim)	Spontaneous	Operator	Service rate, Total mileage, Mileage index, $\emptyset$ waiting time, $\emptyset$ in-vehicle travel time
Dandl et al. (2017)	Microsim. (Aimsun)	Spontaneous	Passenger	Total mileage, Empty mileage ratio, $\emptyset$ waiting time
Narayan et al. (2017)	Agent-based (MATSim)	Spontaneous	Passenger	$\emptyset$ waiting time, Modal split
Fagnant and Kockelman (2018)	Agent-based (MATSim)	Spontaneous	Passenger	Total mileage, Sharing rate, $\emptyset$ waiting time
Fiedler et al. (2018)	Agent-based (AgentPolis)	Spontaneous	Operator	Total mileage, $\emptyset$ occupancy
Friedrich et al. (2018)	Macrosim. (Visum)	Hybrid	Not specified	Service rate, Total mileage, Veh. occupancy
Von Mörner (2018)	Agent-based (AnyLogic)	Spontaneous	2 separate scenarios: (i) Operator (ii) Passenger	Service rate, Total mileage, Empty mileage ratio, $\emptyset$ occupancy
Engelhardt et al. (2019)	Agent-based (FleetPy)	Batch	Operator	Service rate, Mileage index
Richter et al. (2019)	Macrosim. (Visum)	Hybrid	Not specified	Total mileage, Modal split
Engelhardt et al. (2020)	Agent-based (FleetPy)	Batch	Operator	Service rate, Mileage index
Narayan et al. (2020a)	Agent-based (MATSim)	Spontaneous	Passenger	$\emptyset$ waiting time, Modal split, Empty mileage ratio
Narayan et al. (2020b)	Agent-based (MATSim)	Spontaneous	2 separate scenarios: (i) Operator (ii) Passenger	$\emptyset$ waiting time, $\emptyset$ in-vehicle travel time, Modal split, Total operator cost, Total passenger cost

Table 2.2 Overview of ride-hailing and ride-pooling studies involving matching strategies (cont.).

Paper	Simulation Platform	Matching Strategy	Objective Function Perspective	Key Performance Indicators (KPI)
Ruch et al. (2020)	Agent-based (MATSim)	Batch	Passenger	Sharing rate, Total mileage, Veh. occupancy
Armellini (2021)	Microsim. (SUMO)	Batch	Operator	Service rate, Total waiting time, Total vehicle time
Engelhardt and Bogenberger (2021)	Agent-based (FleetPy)	Batch	Passenger & Operator	Service rate, Mileage index, $\emptyset$ walking time $\emptyset$ passenger total travel time
Tang and Armellini (2021)	Microsim. (SUMO)	Spontaneous	Operator	Total vehicle time, Optimality gap
Wilkes et al. (2021a)	Agent-based (mobiTopp)	Spontaneous, then Batch	Operator	$\emptyset$ mileage, Veh. occupancy, $\emptyset$ detour time, $\emptyset$ served requests per veh.
Wilkes et al. (2021b)	Agent-based (mobiTopp)	Spontaneous	Operator	$\emptyset$ mileage, Veh. occupancy, $\emptyset$ detour time, $\emptyset$ served requests per veh.
Engelhardt et al. (2022a)	Agent-based (FleetPy)	Spontaneous, then Batch	Operator	Service rate, Total mileage
Engelhardt et al. (2022b)	Agent-based (FleetPy)	Spontaneous, then Batch	Passenger & Operator	Service rate, Mileage index, $\emptyset$ waiting time, $\emptyset$ detour ratio
Kucharski and Cats (2022)	Agent-based (MaaSSim)	Spontaneous	Passenger & Operator	$\emptyset$ waiting time, $\emptyset$ mileage
Liu and Samaranyake (2022)	Custom framework	Batch	Passenger	Service rate, $\emptyset$ waiting time, $\emptyset$ total delay
Narayan et al. (2022)	Agent-based (MATSim)	Spontaneous	Passenger	Total mileage, Empty mileage ratio, $\emptyset$ passenger travel time
Ghandeharioun and Kouvelas (2023)	Custom framework	Hybrid	Passenger	Service rate, Sharing rate, $\emptyset$ waiting time, $\emptyset$ detour time, Idle time rate
Thomsen (2023)	Macrosim. (Visum)	Hybrid	Not specified	$\emptyset$ mileage, $\emptyset$ detour time, $\emptyset$ occupancy
Engelhardt et al. (2024)	Agent-based (FleetPy)	Batch	Operator	Service rate, Mileage index, $\emptyset$ waiting time Veh. occupancy, Empty mileage ratio
Zubair et al. (2025)	Macrosim. (Visum)	Not specified	Not specified	Service rate, $\emptyset$ waiting time, $\emptyset$ detour ratio

3. Matching Methodologies

In a central on-demand ride-hailing or ride-pooling system, when requests arrive, a central decision must be made to identify appropriate vehicles that can pick up these requests from their origins and drop them off to their destinations. This decision can be challenging when there are multiple vehicles in the system. At that point, the main question becomes: **Which requests should be assigned to which vehicles?**

The core of such an on-demand system is an intelligent matching algorithm, where vehicles and passenger requests are matched in an optimized way.

Before diving deeper into the topic, it is important to note that the two methodologies elaborated in this chapter were developed under certain assumptions. First of all, the requests are not known in advance: they are revealed in the system in real-time and cannot be planned beforehand. From this perspective, the system can be categorized as a dynamic-deterministic DARP (see Section 2.3.1 and Section 2.5). Moreover, the requests are for immediate demand, so no advanced booking is considered.

Furthermore, the system is stop-based, i.e., vehicles can pick up and drop off passengers only at predefined stops. A door-to-door service is not explored. In addition, the walking times of passengers to the stops are not considered; a passenger sending a request is assumed to be already at the origin stop.

Customers whose requests are accepted by the central system, and who receive an offer accordingly, accept this offer. A decision-making mechanism for customers, such as the ability to reject the offer under certain conditions, is not addressed. Customers who do not receive an offer due to the lack of an appropriate vehicle in the system are rejected. These customers then leave the system and do not attempt to submit the same request later again. No vehicle rebalancing strategies are applied, and lastly, every request is assumed to be for one passenger. Groups of passengers who wish to travel together are not taken into account.

The primary components of this system consist of three stakeholders: customers, vehicles, and a central operator (Figure 3.1).

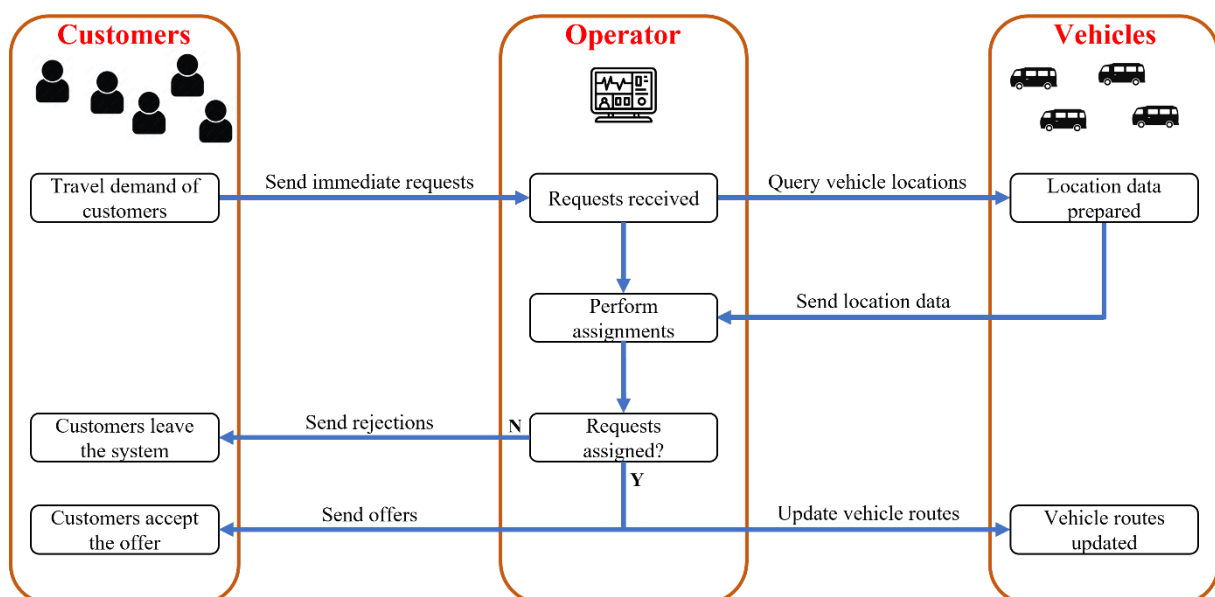


Figure 3.1: Components of an on-demand ride-hailing/ride-pooling system.

In this system, customers send immediate requests, indicating that they want to travel as soon as possible. After receiving these requests, the operator queries the vehicles for their current locations and collects their location data. Based on this information, an assignment procedure is conducted by the operator. At the end of this procedure, requests are either assigned or rejected. In the case of a rejection, customers are informed and leave the system. In the case of an assignment, offers are sent to customers, who then accept the offer. On the other side, vehicle routes are updated accordingly in the meantime.

The findings presented in this chapter form an essential input for subsequent chapters, as the developed assignment strategies are later applied to the microsimulation environment for an on-demand shuttle system simulation.

3.1. Comparison of Spontaneous and Batch Assignment

There are two main matching strategies, which differ in terms of the execution time (Figure 3.2):

- In **spontaneous assignment**, incoming requests are processed immediately as soon as they arrive in the system. This method is discussed in Section 3.2.
- In **batch assignment**, incoming requests are processed within defined time periods, referred to as batches. New requests accumulate during each batch, and a global assignment is conducted at the end of the batch. This method is discussed in Section 3.3.

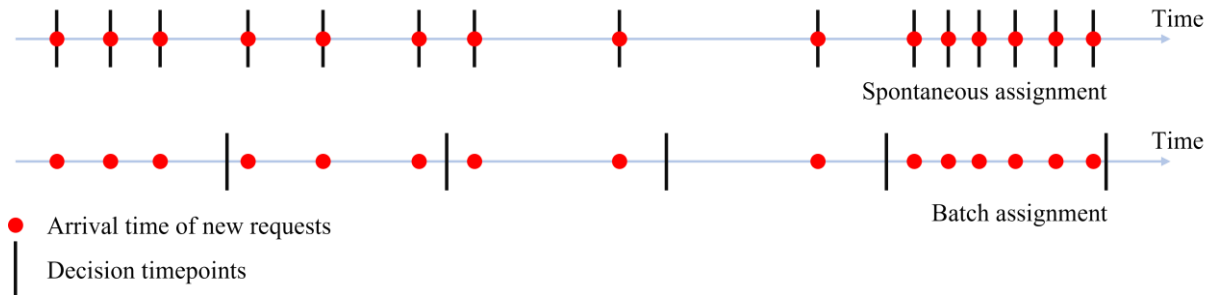


Figure 3.2: Spontaneous assignment versus batch assignment.

Batch assignment has advantages over spontaneous assignment, in that a more global optimization becomes possible, as multiple requests are handled concurrently and their synergies can be better exploited. This effect is especially noticeable in systems with high request densities.

This can be demonstrated through the following simple example: Imagine that a request arrives in the system, and in spontaneous assignment, the most optimal vehicle is assigned to it. This vehicle picks up the request and travels toward its drop-off point. While on the way, another request, from the same pick-up point as the previous one and with a very similar drop-off point, arrives in the system. This second request could have been a good candidate for sharing a trip with the previous one. However, the vehicle has already left the stop with the first request, and it is no longer optimal to travel back and pick up the second request as well. Therefore, in spontaneous assignment, a second vehicle has to be assigned to the second request, and the opportunity of pooling is unfortunately missed.

On the other hand, in batch assignment, both requests may fall into the same batch period, and if that is the case, they are assigned to vehicles at the same time. Here, serving both requests with a single vehicle is clearly an optimum solution, so both are assigned to the same vehicle. In this case, pooling is ensured.

This example can be extended to additional scenarios in systems with high request densities; the likelihood of more efficient pooling is greater in batch assignment compared to the spontaneous assignment.

Despite these advantages, there are also some disadvantages of the batch approach. For example, the response times to requests can be somewhat longer in the batch case, since the request is not processed immediately, as in the spontaneous case. Therefore, the passenger submits the request, but must wait until the end of the batch period to receive a response from the central system. In an unlucky situation, the passenger may have submitted the request at the very beginning of the batch period, meaning the response time would be equal to the entire batch period duration. On the other hand, in cases where the batch period is very short, this impact will be minimal.

Another disadvantage of the batch approach can be the computational effort required to solve the optimization problem. While in spontaneous assignment, processing only one request at a time is ensured; in batch case, several requests may be processed concurrently. Although a more global optimization is thus envisaged, this comes at the cost of the increased computational complexity. Such dial-a-ride problems (DARP) are categorized as NP-hard, and the computational time can rise to very high levels very quickly when the problem size expands (Engelhardt et al., 2020; Ho et al., 2018).

3.2. Spontaneous Assignment of Requests

In this section, the methodology developed for spontaneous assignment of requests to vehicles is scrutinized. By spontaneous assignment, it is meant that the requests which arrive in the system are immediately assigned to an available vehicle upon arrival. Moreover, the performed assignment of a request is final and will not be changed later. If there is no suitable vehicle available, the request is rejected immediately. The rejected request is then removed from the system. Therefore, every request is processed only once, at the moment it arrives in the system.

In the case of an incoming request, the dispatching algorithm has to decide which vehicle will handle it. However, even if there is only one vehicle in the system, there may be multiple possibilities to serve this request. These possibilities (or, more precisely, options) depend on the previous state of the vehicle, i.e., whether any other requests were assigned to that vehicle in previous periods (Arslan & Hoffmann, 2024).

3.2.1. Creating Options

For a better understanding of how options are created, this part will be elaborated separately for the ride-hailing and ride-pooling cases.

3.2.1.1. Ride-hailing Case

In a ride-hailing system, requests are delivered by a vehicle one after another, pooling is not considered. Therefore, ride-hailing can be treated as a special case of ride-pooling, with vehicle capacities of 1.

For a ride-hailing vehicle that has no assigned requests (idle mode), there is only one insertion possibility of the new request. If a new request from passenger X arrives in the system at a given moment, the only option for an idle vehicle is as follows (P: Pick-up, D: Drop-off):

- It firstly picks X up, then drops X off:

$$X_P \rightarrow X_D$$

For a vehicle that already has a passenger on board, there is also only one insertion possibility of the new request. Since no pooling is allowed in ride-hailing, the vehicle can deliver the new request only after it has brought the passenger on board to his drop-off point. For instance, a vehicle has just picked up passenger A, and shortly after, a new request from passenger X arrives in the system. The only insertion possibility is as follows:

- It firstly drops A off, then picks X up, and at last drops X off: $A_D \rightarrow X_P \rightarrow X_D$

In the case that a new request arrives while the vehicle is already traveling to pick up request A, there can be two alternatives. Either the vehicle firstly picks up and drops off the existing request A, and then serves the new one; or it firstly picks up and drops off the new request, and then serves the existing one. The latter is especially suitable in cases where both the pick-up and drop-off of the new request lie along the vehicle's route to the existing request's pick-up location. The two insertion possibilities are as follows:

- It picks A up, then drops A off, picks X up, and at last drops X off: $A_P \rightarrow A_D \rightarrow X_P \rightarrow X_D$
- It picks X up, then drops X off, picks A up, and at last drops A off: $X_P \rightarrow X_D \rightarrow A_P \rightarrow A_D$

3.2.1.2. Ride-pooling Case

In the case of ride-pooling, sharing a vehicle trip among different requests is possible. Therefore, a vehicle does not have to wait to drop off one request to pick up another one. Depending on its capacity, a vehicle can already have multiple requests on board. Moreover, the vehicle can also be assigned to additional requests that have not yet been picked up. The ride-pooling case can thus be addressed in four sub-cases:

- if an idle vehicle gets a new request ($\rightarrow$ Trivial solution, only one option possible),
- if a vehicle that has in-vehicle passenger(s) gets a new request,
- if a vehicle that is on its way to pick-up passenger(s) gets a new request,
- both last cases combined.

Ride-pooling case with in-vehicle passenger(s)

In this sub-case, a vehicle has just picked up passenger A and shortly after, a new request from passenger X arrives in the system. If this vehicle is to pick up this passenger as well, it has exactly 3 options:

- It firstly picks X up, then drops A off, and at last drops X off: $X_P \rightarrow A_D \rightarrow X_D$
- It firstly picks X up, then drops X off, and at last drops A off: $X_P \rightarrow X_D \rightarrow A_D$
- It firstly drops A off, then picks X up, and at last drops X off: $A_D \rightarrow X_P \rightarrow X_D$

As the number of in-vehicle passengers increases, the number of options increases accordingly. To generalize this in a formula, assuming that i is the number of in-vehicle passengers in a vehicle, the number of options to transport the in-vehicle passengers along with the new request can be expressed as follows:

$$\text{Total number of options} = \frac{(i + 2)!}{2} \quad (3.1)$$

There are i drop-off points, one new pick-up point, and one new drop-off point. Therefore, the permutation of all these points leads to $(i + 2)!$ options. In half of these options, the new pick-up point comes before the drop-off, and in the other half, it is the other way around. Consequently, the total value is divided by 2.

According to Equation (3.1), small i values lead to a low number of options (e.g., 3 options for $i = 1$) but somewhat larger i values result in a higher number of options (e.g., 60 options for $i = 3$). Moreover, even larger values will lead to a huge number of options (e.g., 2,520 options for $i = 5$).

Ride-pooling case with to-be-picked-up passenger(s)

In this sub-case, a vehicle has just been assigned to a request A and is on the way to pick him up. At that moment, a new request from passenger X arrives in the system. If the vehicle is to pick up this passenger as well, it has exactly 6 options:

- It picks A up, then picks X up, drops A off, and at last drops X off: $A_P \rightarrow X_P \rightarrow A_D \rightarrow X_D$
- It picks A up, then picks X up, drops X off, and at last drops A off: $A_P \rightarrow X_P \rightarrow X_D \rightarrow A_D$
- It picks A up, then drops A off, picks X up, and at last drops X off: $A_P \rightarrow A_D \rightarrow X_P \rightarrow X_D$
- It picks X up, then picks A up, drops A off, and at last drops X off: $X_P \rightarrow A_P \rightarrow A_D \rightarrow X_D$
- It picks X up, then picks A up, drops X off, and at last drops A off: $X_P \rightarrow A_P \rightarrow X_D \rightarrow A_D$
- It picks X up, then drops X off, picks A up, and at last drops A off: $X_P \rightarrow X_D \rightarrow A_P \rightarrow A_D$

Just as in the case of in-vehicle passengers, increasing the number of to-be-picked-up passengers will also increase the number of options. To generalize this in a formula, assuming that t is the number of requests to be picked up by a vehicle, the number of options to transport the to-be-picked-up passengers along with the new request can be expressed as follows:

$$\text{Total number of options} = \frac{(2t + 2)!}{2^{t+1}} \quad (3.2)$$

There are t pick-up points, t drop-off points, one new pick-up point, and one new drop-off point in total. Therefore, the permutation of all these points lead to $(2t + 2)!$ options. For each $(t + 1)$ pick-up – drop-off pair, the pick-up points come before the drop-off in half of the cases. In the other half, it is the other way around. Consequently, the total value is divided by 2^{t+1} .

According to Equation (3.2), small t values lead to a low number of options (e.g., 6 options for $t = 1$), but larger t values will rapidly result in a huge number of options (e.g., 2,520 options for $t = 3$).

Ride-pooling case with in-vehicle and to-be-picked-up passengers

In this sub-case, a vehicle has an in-vehicle passenger A and, in the meantime, is also assigned to a passenger B (to-be-picked-up passenger). At that moment, a new request from passenger X arrives in the system. This case can be regarded as a combination of both previous sub-cases. Since the number of options is high even for small values, a full listing of options is avoided here. Assuming that i is the number of in-vehicle passengers of a vehicle and t is the number of requests to be picked up by a vehicle, a general formula for the number of options to transport the in-vehicle passengers, to-be-picked-up passengers, as well as the new request can be expressed as follows:

$$\text{Total number of options} = \frac{(2t + i + 2)!}{2^{t+1}} \quad (3.3)$$

There are t pick-up points and drop-off points, i drop-off points, one new pick-up point, and one new drop-off point in total. Therefore, the permutation of all these points lead to $(2t + i + 2)!$ options. For each $(t + 1)$ pick-up – drop-off pair, the pick-up points come before the drop-off in half of the cases. In the other half, it is the other way around. Consequently, the total value is divided by 2^{t+1} .

Compared to an insertion heuristic, where the stops of the new request are solely inserted into the current stop sequence of the vehicles, as applied in various studies (Bischoff et al., 2017; Fiedler et al., 2018; Wilkes et al., 2021a); this method of creating options is much more optimal, since the search space is

larger, and even the current stop sequence of the vehicles can be changed if a better solution is found. An insertion heuristic was used in this methodology only in exceptional cases, when the total number of stop points in a case exceeded eight, which required excessive computational time to generate all the options.

3.2.2. Filtering out Options

The generated options ensure a route legitimacy, i.e., that the drop-off point of a request does not come before the pick-up point of that request. However, additional checks must also be performed in order to obtain valid options for the on-demand operation.

The first check is conducted regarding the vehicle capacity. In none of the options should the vehicle capacity be exceeded during the pick-up events. This check is necessary to avoid any capacity overflow, as vehicles might exploit their capacity values by incorporating the new request into their travel plans. Here, if the number of passengers inside the vehicle exceeds its capacity at any point in time, the concerned option is omitted. For example, a vehicle with a capacity of 3 has two in-vehicle passengers (A and B) and one to-be-picked-up passenger (C), when a new request X arrives in the system. The option $C_P \rightarrow X_P \rightarrow B_D \rightarrow X_D \rightarrow C_D \rightarrow A_D$ cannot be valid, as the vehicle occupancy rises to 4 after the pick-up of X (X_P), thereby exceeding the capacity of 3.

The second check is conducted for the waiting times of passengers. Here, a maximum waiting time (MWT), which is an important operational parameter, is applied, as too high waiting times are not acceptable for passengers. The maximum waiting time refers to the maximum acceptable duration between when a passenger submits its request and when picked up. If an option exceeds the maximum waiting time for any involved passenger, it is omitted.

The final check is conducted for the detour ratio of passengers. This applies only to ride-pooling cases, since in ride-hailing, detours do not occur. A maximum detour ratio (MDR), which is another key operational parameter, is employed here, as too high detour ratios are also unacceptable for passengers. The detour ratio is defined as the ratio of the actual travel time (increased due to pooling) to the direct travel time. If an option exceeds the maximum detour ratio for any involved passenger, it is also omitted.

3.2.3. Determination of the Best Option

The procedure mentioned above is applied to all vehicles in the system in order to create a pool of valid vehicle-option combinations. The aim is then to find the best solution from this pool. For this, both the passenger's and the operator's perspectives are analyzed. While operators aim to diminish the vehicle tour times, passengers experience two main sources of inconvenience in ride-pooling: i) waiting time for the vehicle, and ii) detour time due to pooling, defined as the in-vehicle time minus the direct travel time. While the in-vehicle passenger is affected only by the latter, the new request and the to-be-picked-up passenger are impacted by both, depending on the situation. A new assignment causes a change in the route and stop sequence of the vehicle. This affects the remaining tour time of the vehicle, as well as the waiting and detour times of the passengers currently planned in the tour.

For each option, it is checked how much extra cost it adds to the system. On one side, the tour time of the vehicle involved in that option may change due to the new assignment (operator's perspective). On the other side, the waiting times and in-vehicle travel times of the passengers involved in that option may also change (passenger's perspective). For both sides, weightings can be incorporated to place more emphasis on either perspective. An overall score (cost function) of a valid option can therefore be formulated as follows:

$$W_o(tt_{new} - tt_{cur}) + W_p \sum_{r \in req} (wt_{new}^r - wt_{cur}^r) + (ivtt_{new}^r - ivtt_{cur}^r) \quad (3.4)$$

where

- tt is the planned tour time of the vehicle (for current and new system states),
- wt^r is the waiting time of request r (for current and new system states),
- $ivtt^r$ is the in-vehicle travel time of request r (for current and new system states),
- req is the set of requests that are involved in that particular vehicle,
- W_o and W_p are the weightings for the operator and passenger side, respectively.

Here, the valid option with the lowest overall score is used for the assignment, i.e., the new request will be matched to the vehicle associated with that option.

The explained algorithm with each single step is demonstrated in Algorithm 3.1 below.

Algorithm 3.1 Spontaneous Assignment

```

1  Input: Vehicles  $V$  and New Request  $r$ 
2  Output: Vehicle-option tuple  $Selection$ 
3  set  $ValidOptions = \emptyset$ 
4  for  $v \in V$  do
5      set  $CurStC$  = current-state costs
6      set  $O$  = options created by assigning  $r$  to  $v$ 
7      for  $o \in O$  do
8          if operational constraints are not violated then
9              set  $NewStC$  = new-state costs
10             set  $OverallScore = NewStC - CurStC$ 
11             add  $(v, o, OverallScore)$  to  $ValidOptions$ 
12  if  $ValidOptions = \emptyset$  then
13      return (rejection of  $r$ )
14  set  $Selection$  = the vehicle-option tuple in  $ValidOptions$  with min  $OverallScore$ 
15  return  $Selection$ 

```

3.2.4. Illustration of an Example

The logic of spontaneous assignment can be best explained through an exemplary situation, illustrated in Figure 3.3, where a square grid network is depicted. Each edge of the grid has a length of 1 unit and can be traversed in 1 minute by vehicles. The vehicle icons represent the current positions of the three vehicles. The circles and squares on the grid show the pick-up (P) and drop-off (D) points of the requests, respectively. The green square (I_d) represents the drop-off point of the request that is currently travelling in the green vehicle. Furthermore, the red circle (T_p) and red square (T_d) indicate the pick-up and drop-off points of the request that has already been assigned to the red vehicle, but not yet picked up (to-be-picked-up passenger).

To summarize, the yellow vehicle is in idle mode and waiting, the green vehicle is on its way to the drop-off point of the green request, and the red vehicle is en route to the pick-up point of the red request.

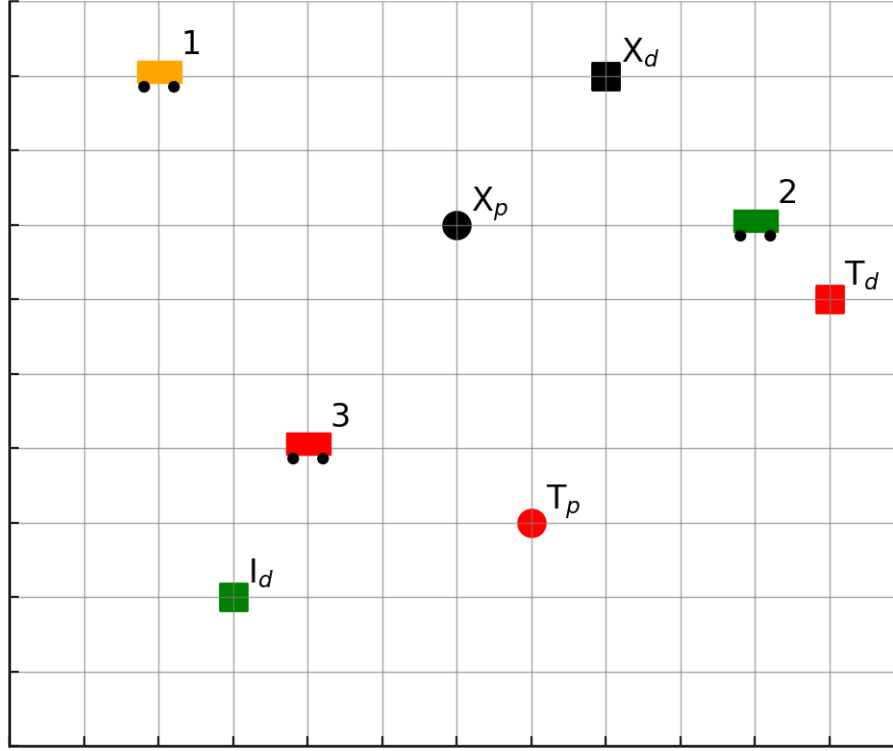


Figure 3.3: Illustration of an exemplary situation on a square grid network (spontaneous assignment).

At the moment when the current situation is as illustrated in the figure above, a new request X arrives in the system, demanding to travel from X_p to X_d . In this constellation, there are a total of 10 options to be checked: vehicle 1 has 1, vehicle 2 has 3, and vehicle 3 has 6 options in total. These options, along with the cost changes for both the vehicle as well as passenger perspectives, and the overall score (OS), are summarized in Table 3.1. To keep the example simple, both weightings W_o and W_p in Equation (3.4) are taken as 1, and boarding/alighting times are ignored. The superscript r stands for the already assigned passenger to the vehicle, and x stands for the new incoming request.

An inspection of the third option (vehicle 2 with X_p - X_d - I_d) in Table 3.1 reveals the following: The tour in the previous state of vehicle 2 takes 12 minutes (the vehicle would travel directly to I_d), whereas the new tour takes 20 minutes ($4+4+12$); hence, Δ_{veh} is 8. The in-vehicle passenger has no waiting times anymore, so Δ_{wt}^r is 0. However, its in-vehicle travel time increases from 12 to 20, resulting in 8 for Δ_{ivtt}^r . The new request will have both a waiting time and in-vehicle travel time of 4 in the new state, and since this request did not exist in the previous state, Δ_{wt}^x and Δ_{ivtt}^x are 4. For this option, the passenger-related costs are 16 in total ($\Delta_{pas} = 0+8+4+4$). The overall score (OS) is then the sum of Δ_{veh} and Δ_{pas} , which amounts to 24.

After calculating the overall score for all 10 options in the same manner, it turns out that the first option with vehicle 1 is the best, and therefore, vehicle 1 with this option is matched to the new request. This occurs even though vehicle 1 is the farthest vehicle from the pick-up point of the new request. To compare, in some other studies such as Fagnant and Kockelman (2018), the nearest vehicle is chosen for assignment; in that case, it would have been vehicle 2 with one of its two options.

For this example, no filtering-out of options was considered (see Section 3.2.2). However, if the maximum waiting time had been set to 5 minutes, the first option would have had to be disregarded due to the violation of this rule, with 6 minutes of waiting time. In that case, the next option with the lowest overall score would be selected, which is option 3.

Table 3.1: Possible options for the exemplary situation (spontaneous assignment).

Veh. No.	Option	Δveh	Δwt^r	Δivtt^r	Δwt^x	Δivtt^x	Δpas	OS
1	X_p-X_d	10	0	0	6	4	10	20
2	$X_p-I_d-X_d$	12	0	0	4	20	24	36
2	$X_p-X_d-I_d$	8	0	8	4	4	16	24
2	$I_d-X_p-X_d$	12	0	0	20	4	24	36
3	$T_p-X_p-T_d-X_d$	10	0	4	9	12	25	35
3	$T_p-X_p-X_d-T_d$	8	0	8	9	4	21	29
3	$T_p-T_d-X_p-X_d$	10	0	0	17	4	21	31
3	$X_p-T_p-T_d-X_d$	12	6	0	5	18	29	41
3	$X_p-T_p-X_d-T_d$	12	6	6	5	12	29	41
3	$X_p-X_d-T_p-T_d$	12	12	0	5	4	21	33

3.3. Batch Assignment of Requests

In this section, the methodology developed for the batch assignment of requests to vehicles is explored. In batch assignment, unlike the spontaneous assignment, requests that arrive in the system are not immediately assigned to an available vehicle. Instead, the assignments are made at the end of predefined time periods, which are called batches, for the requests that were involved in that batch.

The batch assignment algorithm introduced in this thesis is based on the work of Alonso-Mora et al. (2017) and has been modified where necessary. In this algorithm, incoming requests are collected during batch periods and processed together at the end of each period. At that point, as many requests as possible are assigned to vehicles, and the paths of the vehicles are updated accordingly. Requests that cannot be assigned to any vehicle (due to operational constraints) are rejected and removed from the system immediately.

The idea of this approach is to build group of requests (“request-sets”) that are assignable to individual vehicles. This is done in an intelligent manner; starting from single requests, then gradually building upon them by incrementally adding another request to the request-set each time. At every step, the minimum cost of the vehicle/request-set combination is calculated. At the end of this process, a list of vehicle/request-set combinations with their associated costs is obtained. Subsequently, an integer linear programming (ILP) model is applied to ensure that each vehicle is assigned to at most one request-set, and that each request appears in at most one assigned request-set.

Let $V = \{v_1, v_2, \dots, v_x\}$ be the set of vehicles and $R = \{r_1, r_2, \dots, r_y\}$ the set of requests involved in the batch period. For each $v \in V$, it is first checked whether v can be assigned to r_1 . Here, if v is idle, it has only one possibility: it has to pick up r_1 , and then drop off r_1 . If v is not idle, e.g., it already has an in-vehicle passenger at that moment, then there are three possible options (see Section 3.2.1.2 for the formulation).

In the next step, the options are checked to determine whether any operational constraints (vehicle capacity, maximum waiting time, and maximum detour ratio) are exceeded (see Section 3.2.2 for a more detailed explanation of option filtering). Options that violate any of these constraints are removed, the remaining options are considered valid.

For each valid option, it is checked how much extra cost it adds to the system. On one side, the tour time of the vehicle involved in that option may change due to the new assignment (operator’s perspective).

On the other side, the waiting times and in-vehicle travel times of the passengers involved in that option may also change (passenger's perspective). For both sides, weightings can be incorporated to place more emphasis on either perspective. Therefore, the cost of the option is calculated using the following formula:

$$W_o(tt_{new} - tt_{cur}) + W_p \sum_{r \in req} (wt_{new}^r - wt_{cur}^r) + (ivtt_{new}^r - ivtt_{cur}^r) \quad (3.5)$$

where

- tt is the planned tour time of the vehicle (for current and new system states),
- wt^r is the waiting time of request r (for current and new system states),
- $ivtt^r$ is the in-vehicle travel time of request r (for current and new system states),
- req is the set of requests that are involved in that particular vehicle,
- W_o and W_p are the weightings for the operator and passenger side, respectively.

In our case so far, for example, req consists of the request r_l and the in-vehicle passenger in vehicle v . Among the valid options, the one with the lowest cost is selected for this assignment. As a result, the assignment cost of request r_l to vehicle v can be expressed as $c(r_l, v)$, and is written into the list of vehicle/request-set combinations.

In the same manner, vehicle v will be further checked to see whether it can be assigned to the individual requests $r_2, r_3, \dots, r_y$ one by one. For the valid options, their associated costs are calculated using Equation (3.5), then the option with the lowest cost within one assignment is written into the list of vehicle/request-set combinations. This part described so far is referred to as the first level of combinations.

In the next step, the second level of combinations is worked out. Here, the request-set consists of two requests, such as (r_l, r_2) . The main point of the algorithm is that, a second level combination can only exist if both requests separately resulted in a valid option in the previous level. The reason for this is as follows: If a vehicle cannot transport a request alone, it will also not be able to transport that request together with another extra one. The additional request cannot improve the situation, if anything, it can only make it worse. Therefore, not all combinations need to be checked in this level. For example, if only the requests r_l, r_2 and r_4 resulted in valid options at the first level, then only the combinations (r_l, r_2) , (r_l, r_4) , and (r_2, r_4) need to be considered in the second level. When e.g. (r_l, r_2) is checked, the valid options of one of these requests in the previous level are taken as the base, and the other request is inserted into these options.

Again, only the options that meet the operational constraints are labeled as valid. In the next step, it is calculated for each valid option how much extra cost it adds to the system, according to the Equation (3.5). The option with the lowest cost is selected per assignment. Finally, the selected options with their associated costs are written into the list of vehicle/request-set combinations, which, in this case, represent the second level of combinations.

The algorithm continues to increment the request-sets by one additional request at each level. In the third level, the subsets of the request-set from the previous level are checked again, for example, for (r_l, r_2, r_3) , the existence of (r_l, r_2) , (r_l, r_3) and (r_2, r_3) is verified. The cheapest options are written into the list per assignment, and the process continues with the fourth level, fifth level, and so on. Whenever no valid options can be generated at a level, the algorithm stops, since it would no longer be meaningful to continue searching for valid combinations further in the next levels. Otherwise, the algorithm stops after the request-set equal to R is processed.

The algorithm described above is performed for every vehicle in the system. When all vehicles have been processed, the result is a list of vehicle/request-set combinations along with their associated costs. This list, however, contains several combinations with varying request-set sizes and various vehicles; and the problem is now transformed into an optimization problem with the objective of minimizing the overall cost of the entire assignment process.

To solve this optimization problem, the following integer linear programming (ILP) formulation is employed:

$$\min \sum_{(i,j) \in L} c_{ij}x_{ij} + M \sum_{r \in R} p_r \quad (3.6)$$

$$\sum_{(i,j) \in L(j)} x_{ij} \leq 1 \quad \forall j \in V \quad (3.7)$$

$$\left(\sum_{(i,j) \in L(r)} x_{ij} \right) + p_r = 1 \quad \forall r \in R \quad (3.8)$$

The two variables of the formulation are listed as below:

- x_{ij} are the binary variables that take the value 1 if the vehicle/request-set combination for request-set i and vehicle j is used in the optimal solution, and 0 otherwise,
- p_r are the binary variables that take the value 1 if the request r is not served by any vehicle (i.e., rejected), and 0 otherwise.

The parameters and sets of the formulation are as follows:

- c_{ij} is the cost of the vehicle/request-set combination for request-set i and vehicle j (based on Equation (3.5)),
- M is a very large value to penalize rejected requests,
- L is the list of all vehicle/request-set combinations,
- $L(j)$ is the list of all vehicle/request-set combinations that include vehicle j ,
- $L(r)$ is the list of all vehicle/request-set combinations that include request r ,
- V is the set of vehicles,
- R is the set of requests.

Equation (3.6) is the objective function of the problem, aiming to minimize the total cost of the assignment. The first part of the equation represents the costs of the vehicle/request-set combinations selected by the ILP. The second part is a penalty component used to penalize the requests that had to be rejected by the ILP. The very large M value ensures that as few requests as possible are rejected, and highlights that the primary objective of the assignment is to assign several requests.

Constraints (3.7) ensure that each vehicle is assigned to at most one request-set. It is also possible for a vehicle to remain unassigned after the assignment process; therefore, a “less than or equal to” sign is used instead of an equality.

Constraints (3.8) ensure that each request is either assigned to one vehicle, or rejected. If the very left-hand side of the constraints is equal to 1, this means the request is assigned to a vehicle, resulting in p_r taking the value 0, thus no penalty is applied to that request in the objective function. On the other hand, if the left-hand side of the constraints is equal to 0, implying that the request is not assigned to any vehicle, p_r takes the value 1, thus a penalty is applied to that request in the objective function.

To solve the ILP, the formulation above was modeled in Python using the PuLP package (PuLP, 2025) and the Gurobi optimizer (Gurobi Optimization LLC, 2025).

The explained algorithm with each single step is demonstrated in Algorithm 3.2 below.

Algorithm 3.2 Batch Assignment

```

1  Input: Vehicles  $V$  and Requests  $R$ 
2  Output: Selected vehicle/request-set combinations  $SelectedVRSC$ 
3  set  $VRSC = \emptyset$  (list of vehicle/request-set combinations)
4  for  $v \in V$  do
5      set  $CurStC$  = current-state costs
6      for  $n = 1; n \leq |R|; n++$  do
7          set  $RS$  = request-sets composed of  $n$  requests
8          for  $i \in RS$  do
9              if  $n = 1$  or (all subsets of  $i$  from the previous level exist in  $VRSC$ ) then
10                 set  $O$  = options created by assigning  $i$  to  $v$ 
11                 set  $ValidOptions = \emptyset$ 
12                 for  $o \in O$  do
13                     if operational constraints are not violated then
14                         set  $NewStC$  = new-state costs
15                         set  $FinalC = NewStC - CurStC$ 
16                         add  $(o, v, FinalC)$  to  $ValidOptions$ 
17                 set  $SelectedOption$  = the option  $o$  in  $ValidOptions$  with min  $FinalC$ 
18                 set  $c(i, v) = FinalC$  of the  $SelectedOption$ 
19                 add  $SelectedOption$  and  $c(i, v)$  to  $VRSC$ 
20  solve ILP with the inputs  $(V, R, VRSC)$ 
21  ILP Output:  $SelectedVRSC (\subseteq VRSC)$ 
22  return  $SelectedVRSC$ 

```

3.3.1. Illustration of an Example

For a better grasp and understanding of the batch assignment methodology, the following example can be used. An exemplary situation is illustrated in Figure 3.4, where a square grid network is depicted. Each edge of the grid has a length of 1 unit and can be traversed in 1 minute by vehicles. The vehicle icons (I and II) represent the current positions of the two vehicles. The circles and squares on the grid indicate the pick-up (P) and drop-off (D) points of the requests, respectively. In this example, there are 2 vehicles and 4 requests in total. Vehicles can move only along the grid edges, for instance, the travel time from P3 to D3 is 7 minutes. It is assumed that both vehicles move at the same speed and have a capacity of 4. To keep the example simple, the vehicles are in idle mode at the beginning of the problem, with no passengers. The maximum waiting time (MWT) of the requests is set to 10 minutes, and the maximum detour ratio (MDR) to 1.6. Furthermore, both weightings W_o and W_p in Equation (3.5) are taken as 1, and boarding/alighting times are ignored.

In the following part, the algorithm will be outlined step by step. The algorithm firstly explores vehicle I, starting from the first level, then vehicle II in the same manner, subsequently generates the full list of vehicle/request-set combinations, and finally solves the ILP for the most optimal assignment.

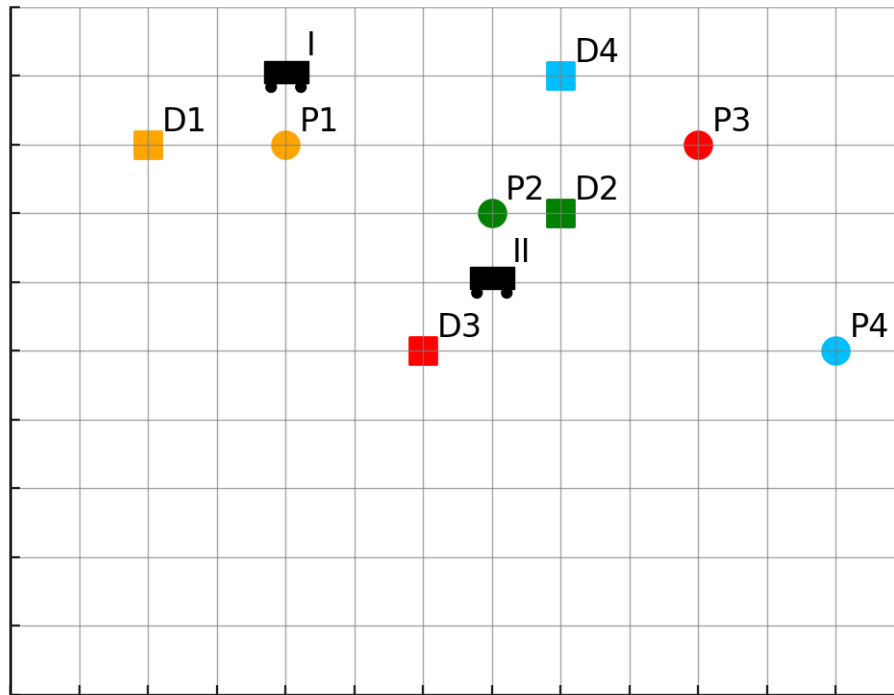


Figure 3.4: Illustration of an exemplary situation on a square grid network (batch assignment).

Vehicle I

The first level of combinations for vehicle I is as follows:

- Vehicle I – Request-Set (1):** The only option for vehicle I to serve request set (1) is $P1 \rightarrow D1$. It takes **1** minute for the vehicle to pick up request 1, and **2** additional minutes to drop him off. In this case, the vehicle travel time is **3** minutes, the waiting time of the request is **1** minute, and the in-vehicle travel time of the request is **2** minutes. Therefore, the cost of the assignment amounts to **6** ($3+1+2$). It should be noted that the vehicle was in idle mode beforehand; that is why the new state cost is equal to the assignment cost. Additionally, none of the operational constraints are violated, hence this matching (as the only option) is added to the list of vehicle/request-set combinations.
- Vehicle I – Request-Set (2):** The only option for vehicle I to serve request set (2) is $P2 \rightarrow D2$. It takes **5** minutes for the vehicle to pick up the request, and **1** additional minute to drop him off. In this case, the vehicle travel time is **6** minutes, the waiting time of the request is **5** minutes, and the in-vehicle travel time of the request is **1** minute. Therefore, the cost of the assignment amounts to **12** ($6+5+1$). As there is no violation of operational constraints, this matching is also added to the list of vehicle/request-set combinations.
- Vehicle I – Request-Set (3):** The only option for vehicle I to serve request set (3) is $P3 \rightarrow D3$. It takes **7** minutes for the vehicle to pick up the request, and **7** additional minutes to drop him off. In this case, the vehicle travel time is **14** minutes, the waiting time of the request is **7** minutes, and the in-vehicle travel time of the request is **7** minutes. Therefore, the cost of the assignment amounts to **28** ($14+7+7$). As there is no violation of operational constraints, this matching is also added to the list of vehicle/request-set combinations.
- Vehicle I – Request-Set (4):** The only option for vehicle I to serve request set (4) is $P4 \rightarrow D4$. It takes **12** minutes for the vehicle to pick up the request, and **8** additional minutes to drop him off. In this case, one of the operational constraints, the maximum waiting time (MWT) of **10**

minutes, is violated, as the waiting time of the request is **12** minutes. Therefore, this matching is disregarded and not added to the list of vehicle/request-set combinations.

The first level of combinations for vehicle I is thereby completed. The algorithm now continues with the second level of combinations for vehicle I:

- **Vehicle I – Request-Set (1,2):** In this case, there are six possible options for vehicle I to transport request set (1,2). These six options need to be checked in sequence:
 - **P1 → P2 → D1 → D2:** In this option, the vehicle travels **1** minute to pick up request 1, then **4** additional minutes to pick up request 2, then **6** additional minutes to drop off request 1, and finally **7** additional minutes to drop off request 2. However, the in-vehicle travel time of request 1 amounts to **10** minutes. Comparing this with his direct travel time, which is **2** minutes, this corresponds to a detour ratio of **5**. As the MDR is **1.6**, consequently, this option is discarded. The same applies to request 2: with an in-vehicle travel time of **13** minutes and direct travel time of **1** minute, his MDR is also exceeded. In any case, one violation is already sufficient for removing the option.
 - **P1 → P2 → D2 → D1:** In this option, the vehicle travels **1** minute to pick up request 1, then **4** additional minutes to pick up request 2, then **1** additional minute to drop off request 2, and finally **7** additional minutes to drop off request 1. Again, the MDR is violated, as request 1 has a detour ratio of **6** ($12/2$). This option is therefore also discarded.
 - **P1 → D1 → P2 → D2:** In this option, the vehicle travels **1** minute to pick up request 1, then **2** additional minutes to drop off request 1, then **6** additional minutes to pick up request 2, and finally **1** additional minute to drop off request 2. No constraint is violated here: the waiting times for the requests are **1** and **9** minutes, respectively, which are below the MWT threshold of **10** minutes. In addition, the detour ratios of both requests are **1**, as they do not experience any detour in this case. Regarding the cost of the option, tt_{new} is **10**, wt_{new}^1 is **1**, $ivtt_{new}^1$ is **2**, wt_{new}^2 is **9**, and $ivtt_{new}^2$ is **1** minute. The total cost of the option hence amounts to **23**, and this matching is a candidate for the list of vehicle/request-set combinations.
 - **P2 → P1 → D2 → D1:** This option must be discarded, as the detour ratio of request 1 amounts to $(5+7)/2 = 6$, exceeding the MDR.
 - **P2 → P1 → D1 → D2:** This option must be discarded, as the detour ratio of request 2 amounts to $(4+2+7)/1 = 13$, exceeding the MDR.
 - **P2 → D2 → P1 → D1:** This option must be discarded, as the waiting time of request 1 amounts to **11** minutes ($5+1+5$), exceeding the MWT.

To sum up, for the combination **Vehicle I – Request-Set (1,2)**, there is only one valid option (**P1 → D1 → P2 → D2**). Therefore, this option is added to the list of vehicle/request-set combinations.

- **Vehicle I – Request-Set (1,3):** In this case, there are again six possible options for vehicle I to transport request set (1,3). These six options are listed below; however, the explanations will be presented more concisely:
 - **P1 → P3 → D1 → D3:** Discarded, as the MDR is violated for both requests.
 - **P1 → P3 → D3 → D1:** Discarded, as the MDR is violated for request 1.
 - **P1 → D1 → P3 → D3:** Discarded, as the MWT is violated for request 3.
 - **P3 → P1 → D3 → D1:** Discarded, as both the MWT and MDR are violated for request 1.

- **P3 → P1 → D1 → D3:** Discarded, as the MWT is violated for request 1, and the MDR is violated for request 3.
- **P3 → D3 → P1 → D1:** Discarded, as the MWT is violated for request 1.

Therefore, there are no valid options for this vehicle/request-set combination, and nothing is added to the list of vehicle/request-set combinations.

- **Vehicle I – Request-Set (1,4):** There is no need to check this combination, as it was shown in the first level that request 4 could not be served by vehicle I. According to the algorithm, all subsets of the request-set from the previous level must exist in the list of vehicle/request-set combinations. However, this is not the case here. Therefore, this combination is discarded immediately.
- **Vehicle I – Request-Set (2,3):** The six options for this combination are listed as follows:
 - **P2 → P3 → D2 → D3:** Discarded, as the MDR is violated for request 2.
 - **P2 → P3 → D3 → D2:** Discarded, as the MDR is violated for request 2.
 - **P2 → D2 → P3 → D3:** This option is valid, as no constraints are violated. Regarding the cost of the option, tt_{new} is **16**, wt_{new}^2 is **5**, $ivtt_{new}^2$ is **1**, wt_{new}^3 is **9**, and $ivtt_{new}^3$ is **7** minutes. The total cost of the option hence amounts to **38**, and this matching is a candidate for the list of vehicle/request-set combinations.
 - **P3 → P2 → D3 → D2:** Discarded, as both the MWT and MDR are violated for request 2.
 - **P3 → P2 → D2 → D3:** Discarded, as the MWT is violated for request 2.
 - **P3 → D3 → P2 → D2:** Discarded, as the MWT is violated for request 2.

To sum up, for the combination **Vehicle I – Request-Set (2,3)**, there is only one valid option (**P2 → D2 → P3 → D3**). Therefore, this option is added to the list of vehicle/request-set combinations.

- **Vehicle I – Request-Set (2,4):** There is no need to check this combination, as it was shown in the first level that request 4 could not be served by vehicle I. Therefore, this combination is discarded immediately.
- **Vehicle I – Request-Set (3,4):** There is no need to check this combination, as it was shown in the first level that request 4 could not be served by vehicle I. Therefore, this combination is discarded immediately.

The second level of combinations for vehicle I is thereby completed. The algorithm would normally continue with the third level; however, there is no need to proceed, as only the request-sets (1,2) and (2,3) could be added to the list in the previous level. Thus, no request-sets with three requests can be considered feasible (Request-set (1,2,3) could have been checked if (1,3) had existed as well.).

As no third level combinations were available, the algorithm does not need to check the fourth level, which would normally be the last. The algorithm therefore terminates for vehicle I.

Vehicle II

The first level of combinations for vehicle II is as follows:

- **Vehicle II – Request-Set (1):** The only option $P1 \rightarrow D1$ is valid, with a cost of **14**.
- **Vehicle II – Request-Set (2):** The only option $P2 \rightarrow D2$ is valid, with a cost of **4**.
- **Vehicle II – Request-Set (3):** The only option $P3 \rightarrow D3$ is valid, with a cost of **24**.

- **Vehicle II – Request-Set (4):** The only option $P4 \rightarrow D4$ is valid, with a cost of **28**.

The second level of combinations for vehicle II is as follows:

- **Vehicle II – Request-Set (1,2):** The only valid option is $(P2 \rightarrow D2 \rightarrow P1 \rightarrow D1)$, with a cost of **20**.
- **Vehicle II – Request-Set (1,3):** There are no valid options for this vehicle/request-set combination, and nothing is added to the list of vehicle/request-set combination.
- **Vehicle II – Request-Set (1,4):** There are no valid options for this vehicle/request-set combination, and nothing is added to the list of vehicle/request-set combination.
- **Vehicle II – Request-Set (2,3):** There are two valid options for this vehicle/request-set combination:
 - $P3 \rightarrow P2 \rightarrow D2 \rightarrow D3$: The option has a cost of **38**.
 - $P2 \rightarrow D2 \rightarrow P3 \rightarrow D3$: The option has a cost of **26**.

The option with the lowest cost is selected for this combination, which is $(P2 \rightarrow D2 \rightarrow P3 \rightarrow D3)$.

- **Vehicle II – Request-Set (2,4):** The only valid option is $(P2 \rightarrow D2 \rightarrow P4 \rightarrow D4)$, with a cost of **34**.
- **Vehicle II – Request-Set (3,4):** The only valid option is $(P3 \rightarrow P4 \rightarrow D3 \rightarrow D4)$, with a cost of **60**.

The second level of combinations for vehicle II is thereby completed. The algorithm then continues with the third level:

- **Vehicle II – Request-Set (1,2,3):** There is no need to check this combination, as request-set (1,3) could not be served in the second level. Therefore, this combination is discarded immediately.
- **Vehicle II – Request-Set (1,2,4):** There is no need to check this combination, as request-set (1,4) could not be served in the second level. Therefore, this combination is discarded immediately.
- **Vehicle II – Request-Set (1,3,4):** There is no need to check this combination, as request-set (1,3) and (1,4) could not be served in the second level. Therefore, this combination is discarded immediately.
- **Vehicle II – Request-Set (2,3,4):** The only valid option for this combination is as follows:
 - $P2 \rightarrow D2 \rightarrow P3 \rightarrow P4 \rightarrow D3 \rightarrow D4$: With tt_{new} **22**, wt_{new}^2 **1**, $ivtt_{new}^2$ **1**, wt_{new}^3 **5**, $ivtt_{new}^3$ **11**, wt_{new}^4 **10**, $ivtt_{new}^4$ **12**, the total cost is **62**.

The third level of combinations for vehicle II is thereby completed. The algorithm would normally continue with the fourth level; however, there is no need to proceed, as only the request-set (2,3,4) was added to the list in the previous level. Thus, no request-sets with four requests can be feasible. The algorithm therefore terminates for vehicle II.

ILP

At the end of the exploration of the different combinations, a full list of vehicle/request-set combinations is obtained. The goal of the ILP is now to determine the most optimal assignments based on the objective function in Equation (3.6), and the constraints in (3.7) and (3.8).

The list of vehicle/request-set combinations with their associated costs is illustrated in Figure 3.5. The linking bars in bold indicate the combinations selected by the ILP.

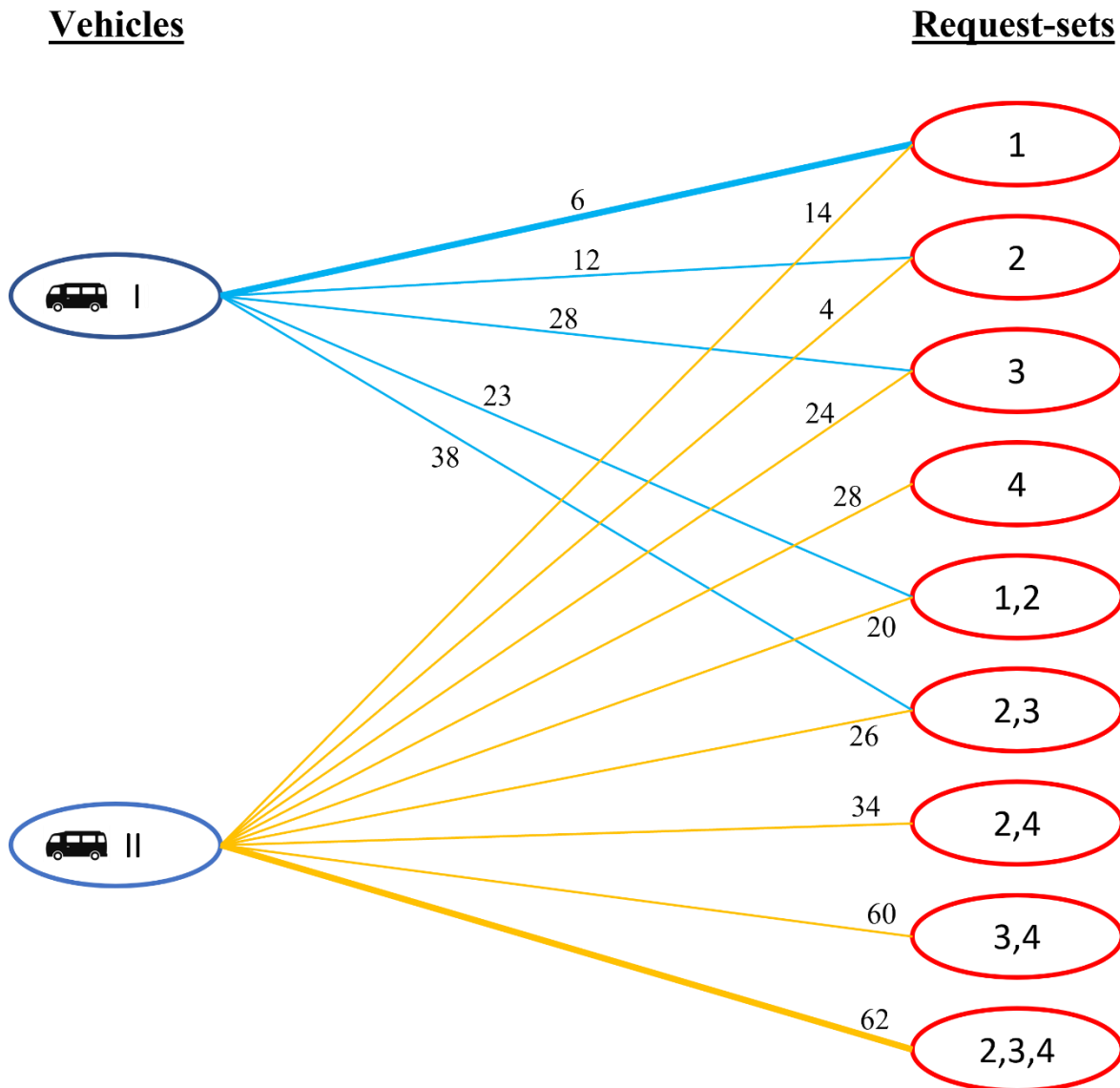


Figure 3.5: List of the vehicle/request-set combinations with the associated costs per combination.

The solution to the problem shows that vehicle I is matched with request 1, and vehicle II is matched with requests 2, 3, and 4. The total cost of the solution amounts to $6+62 = 68$. A quick check of the solution confirms that both constraints (3.7) and (3.8) hold: each vehicle is assigned to at most one request-set, and each request is either assigned to exactly one vehicle, or rejected.

Another important aspect of the solution is that all requests were successfully assigned, and no rejections occurred. The penalty component with a high penalty cost in the objective function ensures this. Otherwise, for instance, a matching for I-(1,2) and II-(3) might have appeared more optimal, as the total cost would be $23+24 = 47$. However, this would result in the rejection of request 4, and thanks to the high penalty cost, this solution is ultimately worse than the optimum solution found.

The final solution includes the vehicle tours that were found as best options within the combinations. These vehicle tours are as follows:

- **Veh. I** : $P1 \rightarrow D1$
- **Veh. II** : $P2 \rightarrow D2 \rightarrow P3 \rightarrow P4 \rightarrow D3 \rightarrow D4$

3.4. Summary

This chapter introduces and examines two algorithmic strategies for vehicle-request assignment in an on-demand ride-hailing and ride-pooling system: **spontaneous assignment** and **batch assignment**. Both strategies were developed under the assumption that requests arrive dynamically in real-time and must be served immediately, without advanced booking. The system is modeled as stop-based, and the operational logic assumes that passengers either accept the assigned offer or are rejected without further resubmission.

In the spontaneous assignment strategy, each incoming request is matched immediately to the most suitable vehicle. This method is computationally lightweight and ensures fast user response times, but it lacks global optimization potential, particularly in high-demand settings. The algorithm explores all feasible options for each available vehicle and evaluates them using a cost function that incorporates both operator's and passenger's perspectives. The methodology is illustrated with a detailed numerical example using a grid network.

The batch assignment strategy aggregates incoming requests over a defined batch period and performs a global optimization at the end of each period. This enables a more holistic assignment by leveraging synergies between similar trips. The chapter presents an incremental approach, where request-sets are built progressively and combined with vehicle options. Each vehicle/request-set combination is scored based on a cost function incorporating both operator's and passenger's perspectives, and an integer linear programming (ILP) model is used to select the optimal set of assignments. The methodology is further illustrated with a detailed numerical example using a grid network, showing the step-by-step development of feasible combinations and the final ILP-based solution.

This chapter directly addresses **Research Question 2** of this thesis: “**How can requests be matched with vehicles in an optimized way, taking both operator's and passenger's perspectives into account?**”. The developed algorithms serve as a critical input for the simulation experiments discussed in later chapters.

4. Application in Microsimulation Environment

This chapter provides insights into the application of an on-demand shuttle system in the microsimulation tool Vissim. The chapter begins with a description of the selected case study area. Then, the general simulation setup is explained, including the data collection methodologies, the integration of this data into the simulation, and specific characteristics of the university campus. Next, the setup particularly for the on-demand traffic is elaborated, including extensions to the standard simulation tool. After explaining the calibration process carried out in the simulation, the methodology for generating request pools for on-demand shuttles is presented. The chapter concludes with an overview of the entire workflow in the simulation.

4.1. Case Study Area

The case study of this thesis is conducted on the campus of the University of the Bundeswehr Munich. Located in the south of Munich Metropolitan Area, in the town of Neubiberg, the campus covers an area of 140 hectares and consists of many facilities serving various purposes, such as various research buildings, administrative buildings, student dormitories, a university library, a post office, sports facilities, sanitary facilities, childcare centers, and restaurants. As of the end of 2024, the campus hosts nearly 5,500 members, with students (65%), academic staff (20%) and non-academic staff (15%) (UniBw Careers, 2024). Most of the students live in the campus dormitories and leave the campus at the weekend to visit home, whereas the academic and non-academic staff commute between home and the campus at working days.

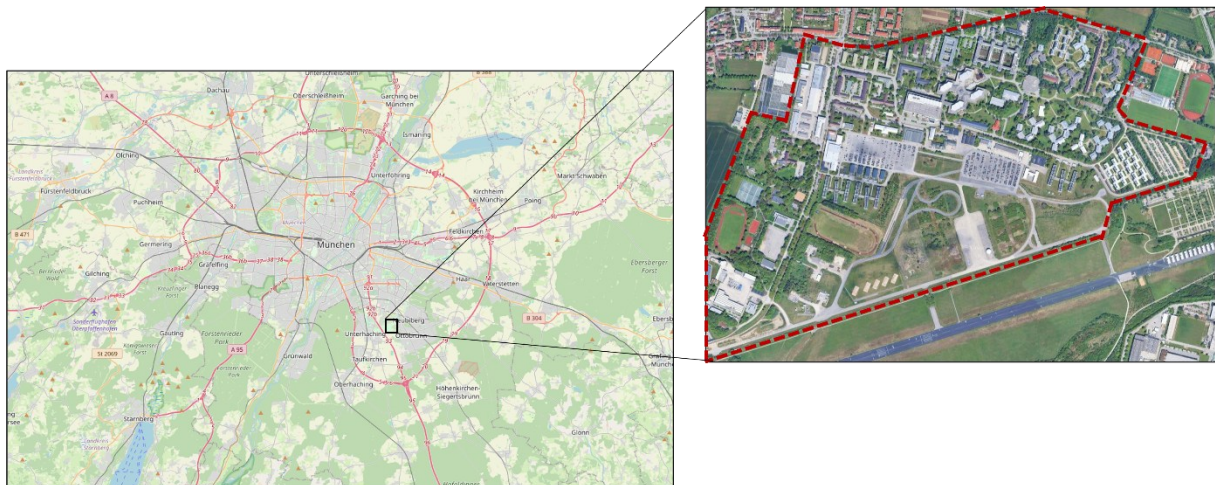


Figure 4.1: Location of the campus in Munich Metropolitan Area (OpenStreetMap and GoogleMaps).

The campus is segregated from the neighboring areas, and a smart card is used by the university members for a controlled access at designated entry points. Visitors, on the other hand, must use the main gate to identify themselves to campus security. Regarding the entry points, there are three vehicle barriers for the motorized and bicycle traffic (main gate, north gate, and east gate). Next to these barriers, there exist turnstiles for the pedestrian access as well. In addition, there are three more entry points for

pedestrians and bicycles only, located in the northern and eastern parts of the campus. The campus map with the important locations and mentioned entry points is depicted in Figure 4.2.

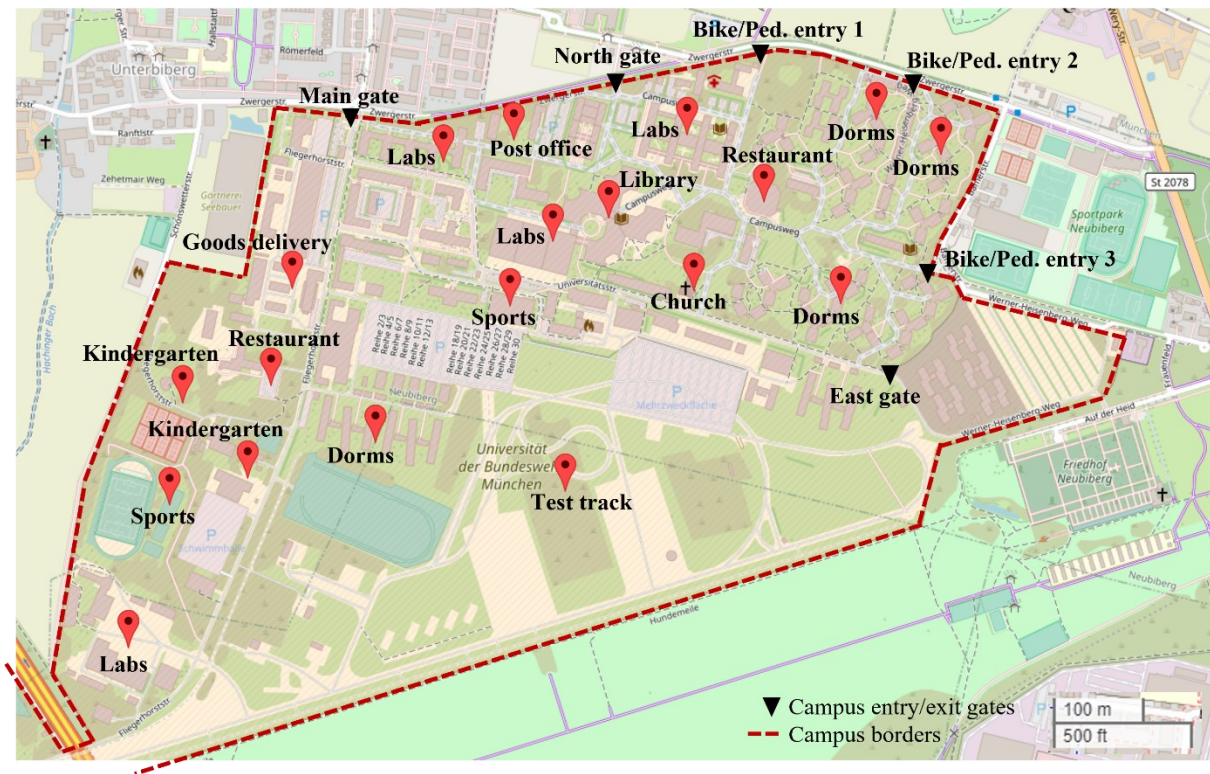


Figure 4.2: Campus map of the University of the Bundeswehr Munich (OpenStreetMap).

4.2. Modeling in Simulation

In this thesis, the campus traffic simulation is performed in the microscopic traffic simulation software Vissim (Version 2024) from the company PTV AG (PTV, 2024), incorporating an on-demand shuttle system. Vissim is a globally recognized microsimulation tool, widely used in traffic engineering for assessing and planning road infrastructure, evaluating traffic management measures, as well as testing new technologies and systems (Daamen et al., 2015).

The campus exhibits a well-balanced multimodal traffic system. Motorized traffic primarily uses the main roads and has extensive parking availability throughout the campus. The main roads have a speed limit of 30 km/h, whereas shared spaces, where motorized and non-motorized users share the roadway, must be traversed at walking speed. Due to short distances between destinations, the campus is very bicycle-friendly, and bicycles are heavily used. They mostly share the roads with motorized traffic. In certain parts of the campus, however, some road sections are used exclusively by bicycles and pedestrians. Pedestrians either use the sidewalks designated for them, or share the road with other traffic participants, such as in shared spaces. In general, the campus offers good infrastructure for walking, and since the distances are relatively small, walking is a very frequently used mode within the campus. Moreover, the micromobility sharing system of the campus, called MORE Sharing, allows university members to book e-scooters, e-bikes, e-cargo bikes, city bikes, and e-mopeds for their trips, with a total fleet of 100 vehicles (Pobudzei & Hoffmann, 2024).

The simulation setup is mainly divided into two sections. In the first part, the most important points of the general simulation setup will be scrutinized; the second part will focus more on the on-demand

features of the simulation, where the prevalent microsimulation tools offer limited standard functionalities. Lastly, calibration of the simulation will be addressed.

4.2.1. General Simulation Setup

In a standard Vissim microsimulation, vehicles enter the network at points where “vehicle input” objects are located. These vehicle inputs possess two main parameters:

- the number of vehicles to be generated, which is a numerical value,
- the vehicle composition assigned, which determines the ratio of different vehicle types (modal mix) within the input.

Each vehicle input operates on pre-defined time intervals, and these two parameters can change for each time interval. This gives the user a certain flexibility in feeding the simulation with vehicles. Vehicle compositions have three parameters per vehicle type:

- the vehicle type,
- the distribution of desired speed,
- the relative flow, which is a numerical value.

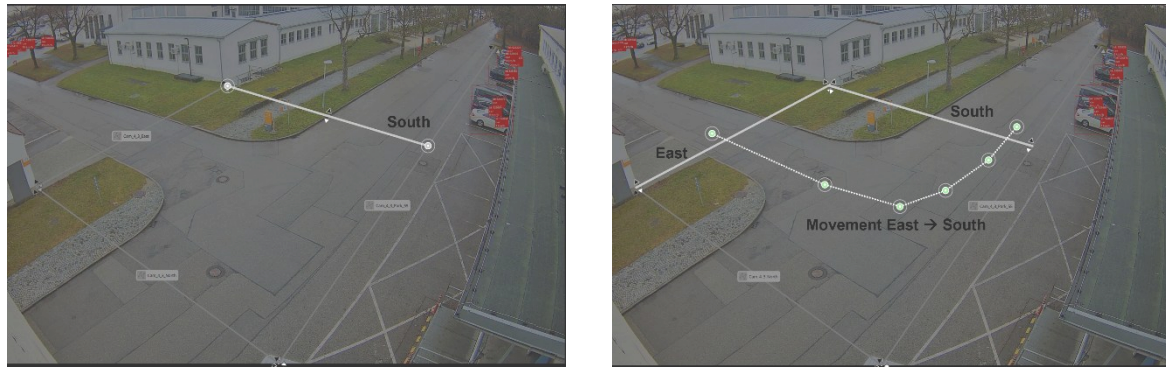
In this way, each vehicle composition object ensures a modal mix among different vehicle types.

After vehicles start moving in the simulation from their input points, they need to make decisions about where to travel when they arrive at road junctions. In a four-arm intersection, for example, an approaching vehicle can have three possible directions to travel. This decision point is modeled in Vissim via routing decisions. A routing decision can be defined for a specific vehicle type, so that only certain vehicle types are affected. Routing decisions in Vissim possess single routes assigned to them. These single routes represent the direction of travel, e.g., northbound, eastbound, and westbound for an intersection approach from the south. Routes have two main parameters: the direction of travel, which is specified by selecting the corresponding exit arm of the intersection in the simulation network, and the relative flow, which is a numerical value. Each route operates on pre-defined time intervals, and the relative flow parameter can change for each time interval. Thereby, various turning ratios in routing decisions can be modeled for different simulation time intervals.

4.2.1.1. Data Collection via Traffic Cameras

In this thesis, a novel approach is employed to feed the simulation with vehicle inputs and turning ratios. The idea is to make use of traffic data collection cameras that were installed at different locations on the university campus, especially at important intersections (MORE Mobility Research, 2024). These cameras can detect different vehicle types and can provide historical count data around-the-clock. An interface was built to gather the relevant data from the cameras and import it into the Vissim simulation.

The workflow can be described as follows: The traffic data collection cameras send the image data via LAN/WLAN to a primary server (FLOW), where operators can set virtual gates on the camera images via a client. Gates detect the vehicles passing over them along with the corresponding vehicle types. The pre-defined vehicle types are: bicycle, bus, car, caravan, e-cart, heavy, light, motorcycle, scooter, tractor, tuk-tuk, van, and pedestrian. In this way, the vehicle counts per type can be produced for every previous minute. While one single gate allows to obtain the vehicle count at a specific cross-section of the road, a two-gate configuration enables description of vehicle movements such as turning at an intersection. A one-gate arrangement is therefore used for vehicle inputs and vehicle compositions, whereas the two-gate arrangement is used for routing decisions and routes (Figure 4.3).



a) Single gate at the south of an intersection, resulting in the counts at that cross-section.

b) Two gates at the east and south of an intersection, resulting in the counts of the movement east to south.

Figure 4.3: Setting the virtual gates on the camera images in FLOW client.

The vehicle counts produced by the gates are then sent to a central back-end server via webhooks, where all campus traffic data are stored. The historical data for a specific date and time is queried and imported into the simulation via the Component Object Model (COM) interface of Vissim. The COM interface enables access to the objects, data, and object parameters in Vissim, as well as their modification. Communication with the server API and the updating of simulation object parameters is implemented in a Python script (see details in Figure 4.5). An overview of the data flow is illustrated in Figure 4.4.

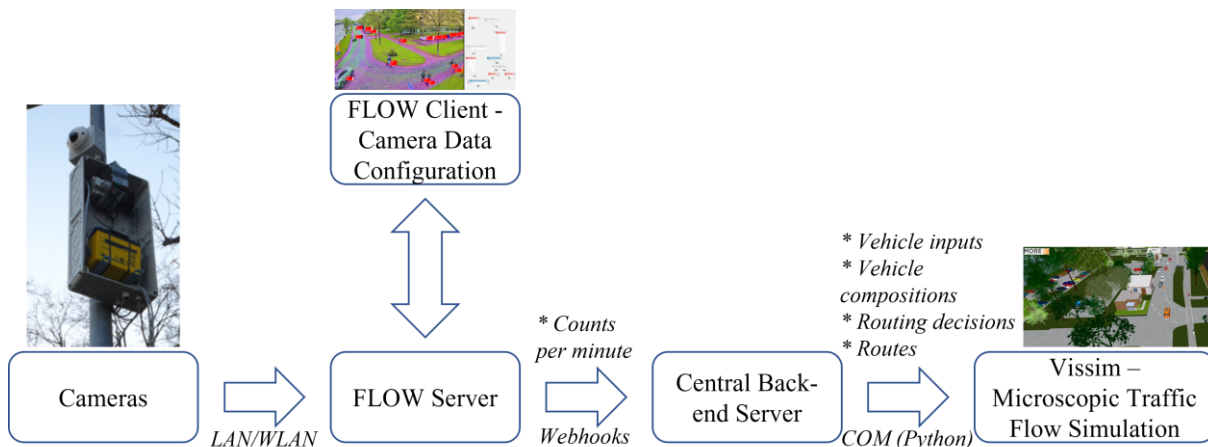


Figure 4.4: Data flow diagram to import camera data into the simulation.

The objects in Vissim whose parameters are updated via camera data are listed as follows:

- **Vehicle Inputs:** Starting point of vehicles. Include number of vehicles and vehicle compositions assigned, both per time interval.
- **Vehicle Compositions:** Modal split of vehicle input. Include vehicle types, desired speed distributions, and relative flow values. Used in vehicle inputs.
- **Routing Decisions:** Decision points for vehicles at intersections. Include vehicle types and single routes.
- **Routes:** Turning directions at intersections. Include direction of travel and relative flow values per time interval.

It is possible to enter the simulation date, simulation time, and simulation period in the graphical user interface of the simulation. These parameters are decisive in determining which data will be retrieved

from the back-end server. In this way, simulation of specific days and time windows from the past is possible (see Figure 4.5).

First, an initialization is carried out for the parameters of the objects mentioned above. This step is necessary because some parameters may remain from the previous simulation run (for a different date/time) when a new simulation is started, and this can cause faulty parameter values. For the vehicle inputs, the ‘number of vehicles’ parameter is initialized to 0 for all time intervals. For the vehicle compositions, the relative flow value is initialized to 1 for “car” vehicle type, and all entries with other vehicle types are deleted. For the routing decisions, all the ones other than “car” and “pedestrian” vehicle type are deleted. Finally, for the routes, all relative flow values for all time intervals and all directions are initialized to 1.

Second, a query is sent to the server API for the selected time period, which returns one-minute count data. Then, the count data is aggregated into 15-minute intervals, a value decided to be the time intervals of the simulation.

Next, a differentiation is made between the objects concerning pedestrians and non-pedestrians (motorized vehicles and bicycles). For the non-pedestrian part, the total counts without pedestrians at each gate for all time intervals are assigned to the corresponding vehicle input. Additionally, the modal split at that gate is assigned to the relative flow values in vehicle compositions, taking vehicle types into account. In terms of routing decisions, the default routing decisions (for car vehicle type) are duplicated for the other vehicle types. Here, it is checked which modes are involved at an intersection approach. The appearing vehicle types receive their own routing decisions by duplication.

The idea here is to be able to set different turning ratios for different vehicle types. For the remaining vehicle types, though, a common routing decision is produced, with all relative flow values set to 1, in case these vehicle types appear due to the vehicle inputs upstream (without any routing decision, those vehicles would receive no routes downstream, which would lead to undesired effects on the simulation). After that, total counts for each direction of travel (each movement) are assigned to the corresponding relative flow values of routes for each time interval.

For the pedestrian part, the methodology is more or less the same, but some steps are skipped. In this part, after 15-minute aggregation, the number of pedestrians at each gate is assigned to the vehicle input for each time interval. These vehicle inputs represent pedestrian generation in the simulation (pedestrians are modeled in Vissim as vehicles, which is why the term “vehicle input” is used on purpose) and are located, for instance, at pedestrian sidewalks, hence separated from the vehicle inputs of non-pedestrians. The vehicle compositions are not modified here and left as the default “all pedestrian”, since no other vehicle type is in use. Regarding the routing decisions, the original routing decision is associated with the pedestrian vehicle type, and there is no need for its duplication since this is the only vehicle type considered. Finally, total counts for each direction of travel (each movement) and time interval are assigned to the corresponding relative flow values of routes.

The interaction between the simulation, the back-end server, and the Python script in terms of data exchange, as well as the individual steps in the Python script, can be summarized as depicted in Figure 4.5.

This approach proves to be very useful because it enables an automatic connection between a data collection technology and the simulation itself, which simplifies the essential data entry in the simulation (see Research Question 3 in Section 1.2). Furthermore, since historical data is available, a specific day and time (for example, the morning peak on weekdays of a specific week, or even an entire day over 24 hours) can be simulated with the corresponding vehicle data.

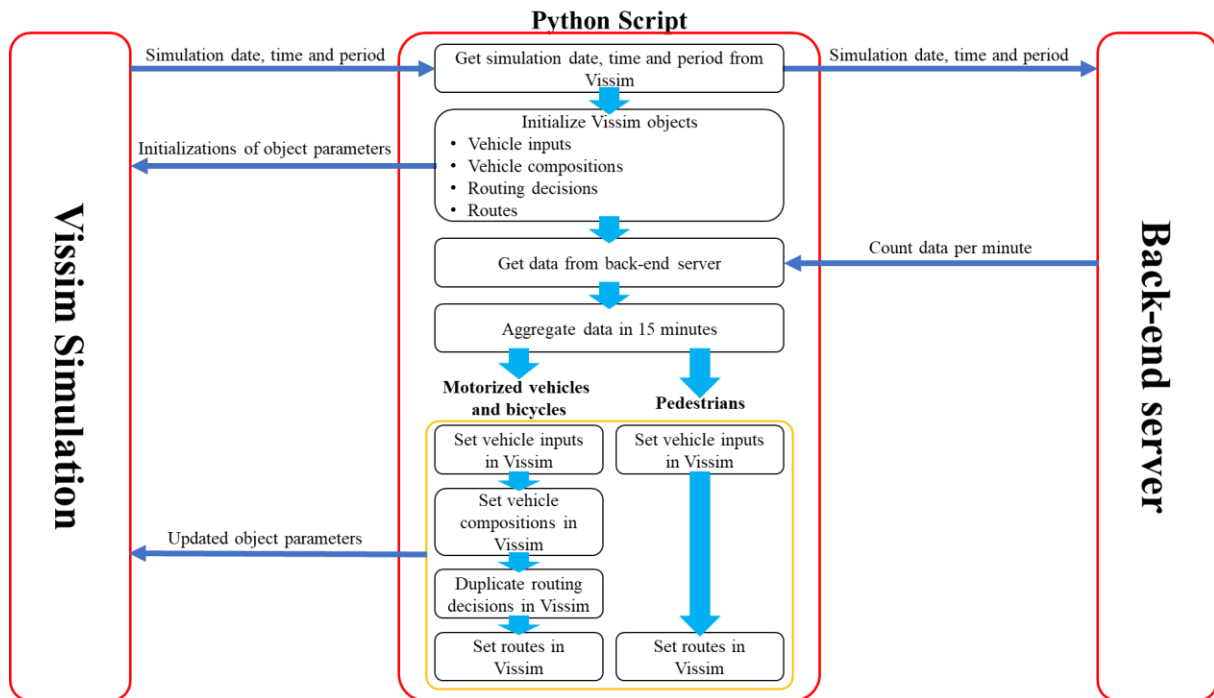


Figure 4.5: Interaction between the simulation, back-end server, and Python script.

Even though it is not within the scope of this thesis, a real-time simulation could be pondered as well, which would update the simulation objects as soon as the data becomes available in the back-end system.

4.2.1.2. Data Collection via Traffic Surveys

Since there was not sufficient coverage of traffic cameras across the whole campus, it has been inevitable to conduct manual traffic surveys. These surveys were carried out at various important intersections and entry points of the campus, making up a total of 11 spots. While the intersection surveys served to feed the routes in the simulation, the entry point surveys were used for the vehicle inputs. The surveys were performed for motorized traffic, bicycles, and pedestrians over five different time periods throughout the day: 7:00-9:00, 9:00-11:30, 11:30-14:00, 14:00-16:00, and 16:00-18:00. For the survey day, always a Tuesday, Wednesday or Thursday was chosen because these days best represented the campus traffic.

As it was logistically demanding to cover all the spots in the surveys for all time periods (55 slots for 11 spots and 5 time periods), some slots were pre-selected. Surveys were ultimately performed at 16 slots, and the remaining ones, where no survey data existed, were filled by extrapolations and assumptions.

Regarding the vehicle inputs, the following procedure was used: Firstly, the camera data from the main gate of the campus (incoming traffic) was analyzed for the weekdays of Tuesday, Wednesday, and Thursday for June 2024. Through this analysis, the distribution of vehicle volumes within one day was obtained (Table 4.1). Here, it can be observed that the campus entries were highest during the morning peak, declined gradually during the day, and were lowest at the evening time. This distribution was then used to extrapolate the missing data for the other campus entry points, under the assumption that the vehicle volumes entering from the other entry points would have the same distribution as at the main gate. The distribution was calculated separately for motorized vehicles, bicycles, and pedestrians.

In this way, the vehicle volumes for a time period could be calculated. Yet, for the distribution within the time period (a time interval of 15 minutes is necessary for the simulation, as decided earlier), another extrapolation was performed: the existing survey data from another spot within that time period was taken as a base, and its distribution was transferred to the spot in question.

Table 4.1: Daily distribution of vehicle volumes at the main gate of the campus (incoming traffic).

	Motorized vehicle (MV)	Bicycle	Pedestrian
07:00-09:00	30 %	40 %	35 %
09:00-11:30	25 %	25 %	25 %
11:30-14:00	20 %	15 %	20 %
14:00-16:00	15 %	15 %	10 %
16:00-18:00	10 %	5 %	10 %
SUM	100 %	100 %	100 %

Regarding the routes, since the transferability from one intersection to another did not make any sense, each intersection was investigated individually. As the relevant value here was the (relative) turning flows, it was assumed that the turning ratios remained constant for the time periods where no survey data existed, and these constant ratios were calculated from the time periods at the same intersection where the survey data was available.

Finally, values of all vehicle inputs and routes that were prepared in a spreadsheet document were copied into the simulation. An overview of the campus intersections covered either by cameras or traffic surveys, as well as the vehicle inputs based on the camera data or traffic surveys, is depicted in Figure 4.6 for clarity.

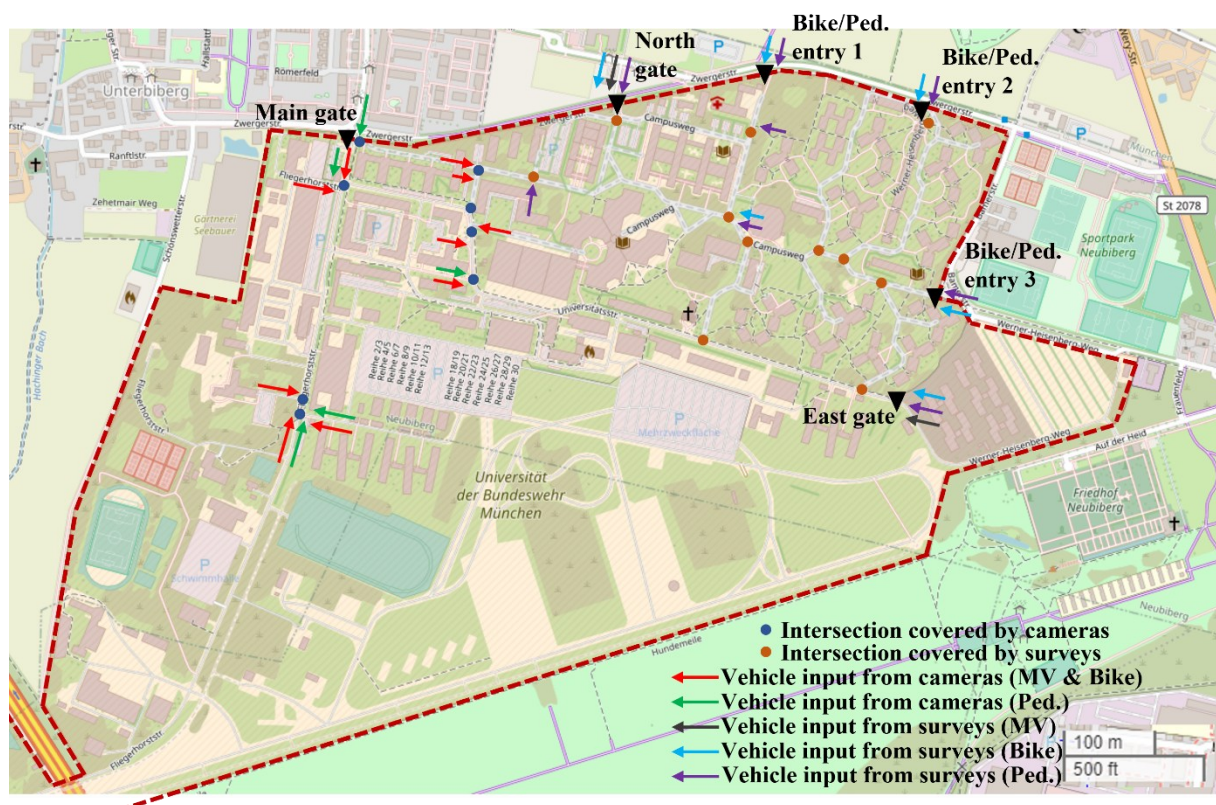


Figure 4.6: Overview of the vehicle inputs and routes used in the simulation (OpenStreetMap).

4.2.1.3. Usage of "Priority Rules" Object at Bottlenecks

In some parts of the shared space on the campus, there are bottlenecks where vehicles must wait for each other if an encounter occurs. In reality, this limitation applies only to situations when two motorized vehicles meet at the bottleneck. The width of the bottleneck does not allow two motorized vehicles to pass at the same time, but the situation is different if cyclists, scooter drivers, or pedestrians are involved.

Since their width is not as large as that of a motorized vehicle, encounters involving these traffic participants do not cause any limitations at the bottlenecks. For example, a cyclist can pass through the bottleneck while a car is passing from the opposite direction.

The first approach to model this situation would be to use the "conflict areas" object in Vissim. However, in conflict areas, a differentiation by vehicle type is not possible. If a conflict area is set at the bottleneck, it applies to all traffic participants. Secondly, in conflict areas, one direction must always be prioritized over the other, even if the arrival times are the other way around. That is why the "priority rules" object is more appropriate for such modeling. Priority rules regulate the priorities between different vehicles in Vissim, when a network section is commonly used. It has four main attributes:

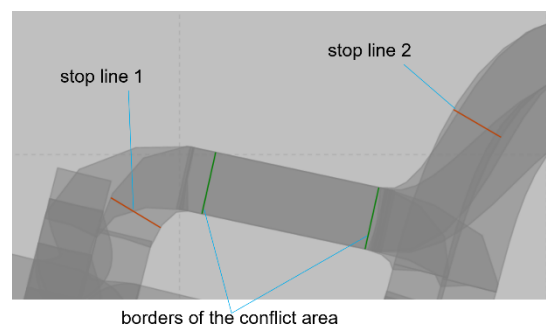
- a conflict area to determine the section of the network where the conflict takes place,
- a stop line indicating exactly where the vehicle has to wait if the conflict area is occupied by the opposing vehicle,
- the vehicle types for which the priority rule is active,
- the vehicle types considered for the opposite direction.

Two priority rules are placed for one bottleneck situation, one for each direction. The conflict area is defined as the bottleneck, and the stop line is placed at the point where the vehicle has to stop if the conflict area is occupied. The vehicle types are then chosen as motorized vehicles for both directions. In this way, if two motorized vehicles encounter each other at the bottleneck, the last-arriving vehicle waits for the clearance of the other one at the stop line before the bottleneck. After the clearance, it can pass through the bottleneck as well.

Four such bottlenecks were identified in the campus network, and they were modeled with eight priority rules in the simulation. One of these is shown as an example in Figure 4.7. A road width of 3.5 meters does not allow two motorized vehicles to pass through the bottleneck at the same time, as seen in the left figure. For this reason, two priority rule objects were created, one for the first and the other for the second direction. The borders of the conflict area, as well as the stop lines for both directions, were modeled in the simulation as illustrated in the right figure.



a) A bottleneck in the campus (orange area) with a road width of 3.5 m.



b) Modeling the bottleneck in the simulation with priority rules.

Figure 4.7: Usage of priority rules for bottlenecks in the campus.

Apart from the special configurations in the simulation mentioned in the previous part, there are other relevant objects used for the modeling of the campus, which are typical in similar simulation models. These will not be explained here in detail, but a selection of the most important model objects can be

listed as follows: “**Reduced speed areas**” to force vehicles to reduce speed at curves, “**Conflict areas**” to regulate the priorities between crossing traffic streams at intersections, “**Desired speed decisions**” to set speed limits for different parts of the campus, and “**Stop signs**” to model the vehicle barriers at the entry/exit gates of the campus, as well as the inner-section barrier (to regulate access to the inner section of the campus) enabling entrance/exit of one vehicle at a time. Since there are no signalized intersections on the campus, no signal controls and signal heads were used.

4.2.2. Simulation Setup for the On-Demand Shuttle Traffic

An on-demand shuttle system is based on two important aspects: flexible routes and flexible departure times. Flexible routing means that the vehicles do not have fixed routes, as typically the case in conventional public transport; rather, the routes are calculated according to the incoming trip requests, leading to a different route each time. In addition, a vehicle’s route may also change during the trip, for example, if a new request needs to be picked up with a detour as in the case of ride-pooling. Therefore, a dynamic aspect must be available for the on-demand implementation. However, in Vissim, the standard usage is that vehicles are assigned static routes along the network, typically at the approaches of an intersection. The arrival point of a vehicle also emerges stochastically according to the turning ratios at intersections and the random seed used. To overcome this issue, the feature “*dynamic assignment*” was utilized in the simulation.

4.2.2.1. Dynamic Assignment

Dynamic assignment normally incorporates the origin-destination (OD) matrices between zones to distribute the traffic across different possible paths in the network, i.e., to enable path selection among alternative routes. Here, the vehicles do not receive static route decisions; rather, they are assigned a path immediately after their emergence in the simulation, which they follow for their entire trip. Paths are particularly useful for the on-demand case, because vehicles must be able to be sent from an arbitrary point to another during the simulation. For example, if a vehicle wants to travel from point A to B in order to pick up a request at B, it needs a direct (and likely the shortest) path from A to B. This is possible with dynamic routing. In contrast, with static routing, the vehicle would need a vehicle-specific static route at each decision point, e.g., at each intersection between A and B, which is highly impractical.

Moreover, with the dynamic assignment feature, it is possible to change the path of a vehicle “on the fly”. This is also very useful for the on-demand case; an appropriate example for this can be given in ride-pooling: A vehicle is travelling from point A to B on a path X, and receives on its path a new pick-up event for an incoming request. Now, instead of going to B, with a small detour, the new destination is point C, which requires a path change during the original trip. As mentioned above, this requirement can be fulfilled using dynamic routing.

To employ dynamic assignment in Vissim, at least one OD matrix has to be activated. Since the distribution of the traffic within the road network was not foreseen, and only dynamic routing features were required, a dummy OD matrix with placeholder zones was created and activated in the model.

Dynamic and static routing can be used simultaneously in the same model, which is very advantageous, because in this way, while on-demand shuttles receive dynamic routes, the rest of the campus traffic can still be routed statically (see Section 4.2.1).

4.2.2.2. Usage of COM Interface

Apart from dynamic assignment, which was elaborated in the previous section, there are also other aspects specific to on-demand simulation, such as the algorithm for matching requests with vehicles,

generating requests, assigning new paths to vehicles, pick-up and drop-off events at stops, or updating the attributes of vehicles and requests. The standard microsimulation functionalities are not sufficient to handle such peculiarities. This makes the use of an external programming in on-demand simulations inevitable.

Vissim does indeed offer an external control of the simulation via the COM interface (see also Section 4.2.1.1). COM provides access to data and functions contained in other programs. For example, objects, data, and object attributes in Vissim can be accessed via the COM interface. Conversely, object attributes can be modified via the COM interface, too. For special research applications, it is highly recommended by the software provider to use this interface to enhance the capabilities of the simulation.

For this reason, the COM interface was used extensively in this thesis. The external programming was conducted using a Python code. A detailed description of this code and the object attributes involved in the COM interface is explained more in detail in Section 4.2.3.

Furthermore, COM provides some very useful default methods for on-demand simulation. These are:

- **AddPathForVehicle:** Adds a “*path*” object to Vissim for a specific vehicle from its origin to a given destination, continuing from the current vehicle position along the given node sequence. This method requires three parameters: i) the numeric ID of the specific vehicle, ii) the destination parking lot, iii) the node sequence of the path.
- **AssignPath:** Assigns a specific path to a specific vehicle. This is mostly used right after the *AddPathForVehicle* method. This method requires two parameters: i) the numeric ID of the specific vehicle, ii) the *path* object.
- **AddVehicleInParkingLot:** Creates a new vehicle with a specified vehicle type in the specified parking lot. This method is used for the initialization of the simulation with shuttles, and requires two parameters: i) the vehicle type, ii) the parking lot where the vehicle will be created.

4.2.2.3. Nodes, Edges and Shortest Path Calculation

Another prerequisite for using dynamic assignment in the simulation is equipping the network with nodes and edges, which can be related to graph theory in mathematics. As mentioned before, the paths of a vehicle are created between two parking lots, but there can be various path options if alternatives exist in the network. Each path alternative is represented by a node sequence showing along which checkpoints a vehicle has to travel when a path is assigned to it. Nodes can be regarded as decision points in the network (e.g., intersections) and edges are the connections between them.

When a vehicle has to travel from point A to B, the system must be able to create a path starting from A and ending at B, including the node sequence in between. This node sequence, and hence the path itself, can be calculated using a shortest route algorithm, such as Dijkstra’s.

The conducted procedure was as follows: Firstly, nodes were created in the network where necessary, including the decision points (intersections), shuttle stops, and the depot locations, with the option “*use for dynamic assignment*” activated. Vissim then generated the edges between the nodes automatically. At the beginning of the simulation, these nodes and edges were exported from Vissim to gather the graph of the network. This graph is basically a matrix of $n \times n$, where n is the number of nodes. The (a,b) cell of the matrix represents, for example, the cost of travelling between two neighboring nodes a and b , typically the distance between the two.

Secondly, Dijkstra’s shortest route algorithm was applied to this graph to compute the shortest paths between each and every node. Two matrices of $n \times n$ were returned as a result:

- a cost matrix C with cells (a,b) representing the cost (distance) of the path from a to b ,
- a path matrix P with cells (a,b) representing the penultimate node of the path from a to b .

In order to obtain the node sequence of the entire path, thirdly, a backtracking was applied to the matrix P . As a result, both the cost and the node sequence of every single path could be determined, and imported back into Vissim (see Section 2.2.2 for more details on the Dijkstra's algorithm). An overview of these steps is demonstrated in Figure 4.8. Integration of this process into the entire simulation control is explained in Section 4.2.3 more in detail.

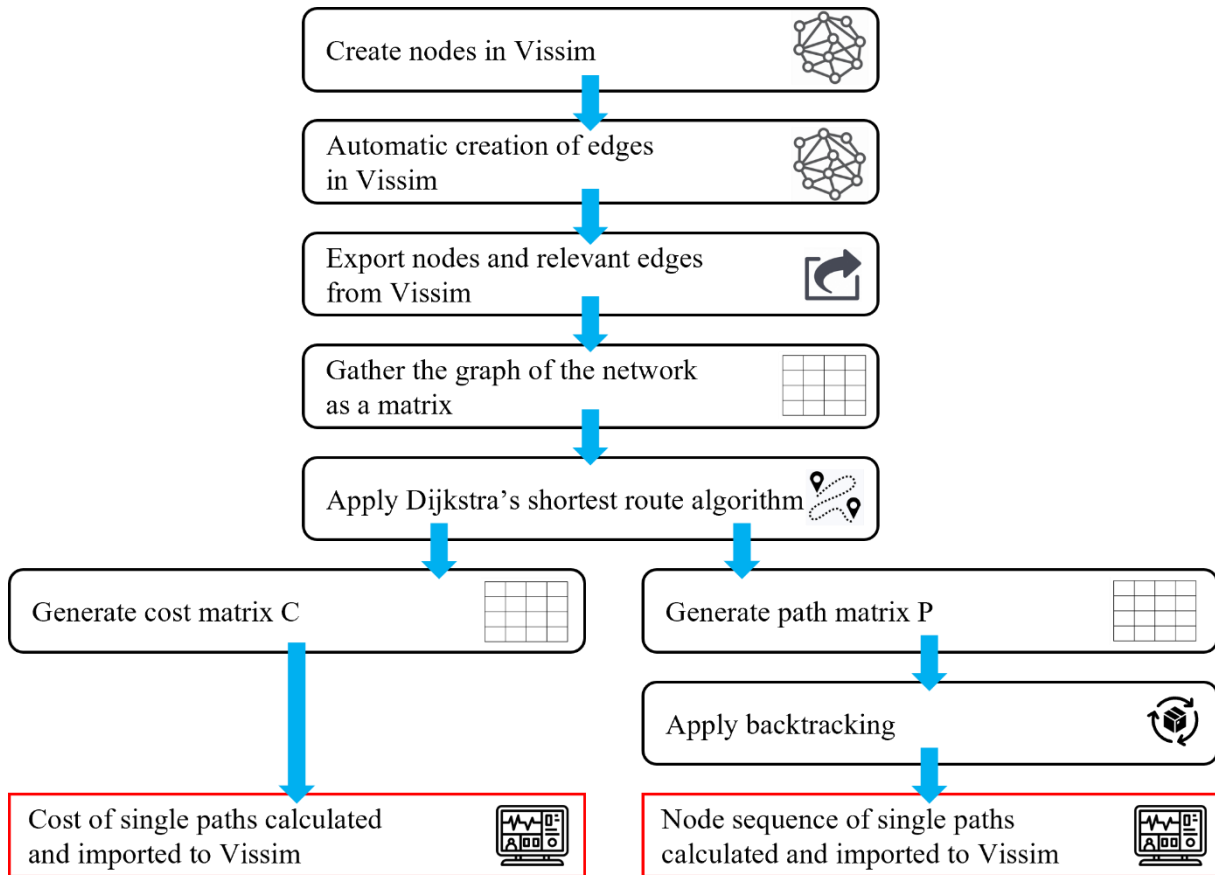


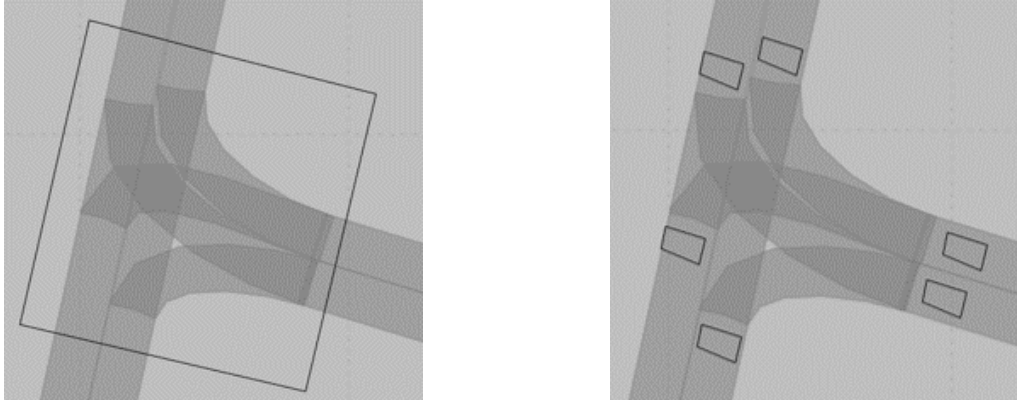
Figure 4.8: Steps to calculate the cost and node sequence of Vissim paths using Dijkstra's shortest route algorithm.

Since the on-demand shuttles did not use the whole network of the simulation (for example, sidewalks or small road sections where shuttles were not supposed to drive at all), it was necessary to differentiate the edges in terms of their usage by the on-demand vehicles. This was done by labelling the network nodes as relevant or non-relevant, and only the edges between relevant nodes were exported. In this way, the paths were created using only the relevant road sections.

In the simulation, if a vehicle was located at a node at the moment of path generation, then the starting point of the path for the path calculation was trivially taken as this node. On the other hand, if a vehicle was on its way and currently not located at a node (rather positioned on an edge), then its next node in the current path was taken as the starting point of the new path.

In standard cases of dynamic assignment, for one intersection in Vissim, one single node is created. However, this leads to problems in on-demand routing if a vehicle has to be rerouted during its trip. For instance, if a moving vehicle receives a pick-up request from a stop placed in the opposite direction of travel, then a path will be created beginning from its current next node to the destination stop. However,

if the vehicle does not have a chance to make a U-turn within the next node, then the created path will be one that cannot be followed by the vehicle. This causes an abortion of the simulation due to the inconsistencies. To prevent this, the node structure was fundamentally modified: each approach and exit of the intersection was assigned a separate node instead. In this way, since the nodes were differentiated for the directions, the created paths automatically considered the turning possibilities, i.e., whether the exit and the approach node were linked by an edge. The comparison of the two different node structures is illustrated in Figure 4.9.



a) Conventional node structure in Vissim with one node per intersection.

b) Modified node structure in Vissim for on-demand operations with one node per approach and exit of an intersection.

Figure 4.9: Comparison of two different node structures in Vissim.

4.2.2.4. Shuttle Stops

The on-demand system in the simulation is a stop-based system, meaning that the shuttles can pick up and drop off requests only at designated shuttle stops. Requests travel from one stop to another; in reality, these would be virtual stops that are displayed in the on-demand smartphone app. In the simulation, these stops were modeled as the Vissim object “*parking lots*”, since vehicle paths can only be created between two parking lots that are active for dynamic assignment. It is possible to activate a parking lot in dynamic assignment by simply assigning the parking lot to a dummy zone.

Vehicles with an assigned path travel to the destination stop, which is actually a *parking lot* object in Vissim. For this parking lot, a virtual “*parking routing decision*” object is created internally during the simulation, located very close to the parking lot itself. Even though the vehicles know their path (and thus the destination parking lot) for the entire course of their trip, the stopping event of the vehicle begins only after passing over this *parking routing decision*. Then, the vehicle state changes to “*driving to parking space*” and the vehicle subsequently stops at the destination.

A problem, however, occurs when the parking lot is already occupied, for example by another vehicle. When the second vehicle approaching the destination stop observes that the stop is occupied, it waits behind the occupying vehicle until the stop becomes free again. But, if this vehicle passes over the “*parking routing decision*” object before the stop is freed, and waits after that, the vehicle ignores its stopping event at the destination and continues to travel in the network without any path or route, which eventually leads to a malfunction in the simulation. This situation can occur in the on-demand simulation, for instance, when a shuttle is currently picking up a passenger and occupying the shuttle stop, while another shuttle approaches the same stop to drop off any other passenger.

To avoid this situation, the Vissim object “*priority rules*” was utilized. As mentioned in Section 4.2.1.3, the *priority rules* object regulates the priorities between different vehicles in Vissim, in case a network section is commonly used. From its four main attributes, two were applied here; a conflict area to determine the section of the network where the conflict (occupancy) takes place, and a stop line indicating exactly where the second vehicle has to wait if the conflict area is occupied. In the on-demand case, this was useful because the aim was to ensure that the second shuttle waits “before” the *parking routing decision* in case of an occupied stop.

For the implementation, the following was applied: The conflict area was defined as the shuttle stop, and the stop line was placed right before the *parking routing decision*. An illustration of this is shown in Figure 4.10. Here, the blue rectangle is the shuttle stop, the purple bar is the *parking routing decision*, the area marked with a green bar and green triangle represents the conflict area, and finally the orange bar shows the stop line. Additionally, the arrows on the link show the direction of travel. With this setup, provided that the shuttle stop is occupied by a shuttle, it is ensured that the next arriving shuttle waits right before the *parking route decision*, thereby not losing its path information. This was applied to all shuttle stops in the simulation.

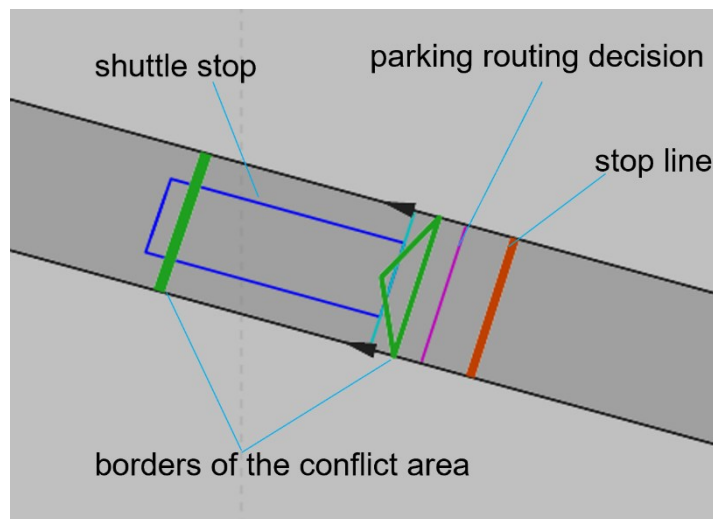


Figure 4.10: Implementation of priority rules at shuttle stops.

Based on the campus entry/exit points, the most important campus locations, and the surveys conducted among university members, a total of 16 shuttle stops were defined on the campus (see Figure 4.13). The stops were configured as stop bays within the simulation to avoid obstructing the main traffic flow.

4.2.2.5. New Attributes for Vissim Objects

In order to enable a functioning on-demand simulation and the generation of essential KPIs, new object attributes were created in Vissim. These attributes were used within the external Python code, and their values were modified when necessary.

There are two main parties in an on-demand system: vehicles (shuttles) and passengers (requests). Since there exists no object for passengers in Vissim, passengers were defined under the vehicle object. This is actually not very different from common usage, because pedestrians are also defined as vehicles in Vissim. Although all the new attributes addressed in this section are therefore under the “*vehicle*” object of Vissim, they are categorized under two main headings (vehicles and passengers) to ensure clarity for the reader. Lists of new vehicle and passenger attributes can be found in Table 4.2 and Table 4.3, respectively.

Table 4.2: New attributes for the vehicles in Vissim.

Vehicle Attribute	Description of the Attribute
Shuttle ID	Numeric ID of the shuttle vehicle.
Shuttle Status	Shows the status of the shuttle during the simulation: Idle / Empty-drive / Occupied / Picking up a passenger / Dropping off a passenger.
Idle Time	Time-counter for periods when vehicle was idle.
Empty Drive Time	Time-counter for periods when vehicle was empty driving.
Occupied Time	Time-counter for periods when vehicle was occupied.
Time with Occupancy x	Different attributes for different x values. Time-counter for periods when vehicle occupancy was x .
Empty Drive Kilometer	Km-counter for periods when vehicle was empty driving.
Occupied Kilometer	Km-counter for periods when vehicle was occupied.
Previous Mileage	Total distance traveled until the previous simulation time step (to calculate how much distance was covered in the last time step).
Passenger-Kilometer	Passenger-km of the vehicle. Calculated cumulatively by occupancy*mileage.
Mean Occupancy	Mean occupancy of the vehicle. Calculated by passenger-km / mileage.
Passengers in Shuttle	List of the requests that are currently on board in the shuttle.
Passengers to-be-Picked-up	List of the requests that are assigned to a shuttle, but not yet picked up.
Assigned Option	The option assigned to the vehicle from the last matching procedure. Here, the sequence of pick-up/drop-off events is stored in a list. The list is updated after every pick-up/drop-off event or new matchings.
Planned Tour	The sequence of shuttle stops whom vehicle is supposed to visit according to the last matching procedure. Here, the sequence is stored in a list. The list is updated after every pick-up/drop-off event or new matchings.
Distance Checkpoint	Total distance traveled until the last stopping event (e.g., pick-up, drop-off). A more detailed explanation is given below.

The vehicle attribute “*Distance Checkpoint*” requires further elaboration here. When a new request arrives in the system, it is necessary to calculate the distance from the vehicle’s position to the pick-up point of the new request. This can easily be calculated by the Dijkstra’s shortest route algorithm if the vehicle is currently on a node. However, if the vehicle is not on a node, but rather somewhere between two nodes, then the Dijkstra’s algorithm will not be able to return the exact distance, i.e., the distance from the vehicle to its next node will remain unconsidered.

To address this, a temporary path is created for the vehicle, expecting to start from its position and end at the request pick-up point. The problem is that paths in Vissim are generated solely from one parking lot to another parking lot, that is why the generated path also includes the past segment of the vehicle’s path since its last movement start. For this reason, this portion has to be deducted from the temporary path distance.

To overcome this issue, the attribute “*Distance Checkpoint*” was created that stores the total mileage of the vehicle until the last stopping event (the total mileage of a vehicle is already stored in the standard Vissim attribute “*Total distance traveled*”). Then, if a new distance needs to be calculated for a vehicle that is on its way, the difference between the total mileage and “*Distance Checkpoint*” is subtracted from the distance of the temporary path (see Equation (4.1)).

Table 4.3: New attributes for the passengers in Vissim.

Passenger Attribute	Description of the Attribute
Passenger ID	Numeric ID of the request.
Passenger Status	Shows the status of the passenger during the simulation: Not yet assigned / Assigned&Waiting / On board / Dropped off / Denied.
Passenger Origin	Origin (start stop) of the passenger.
Passenger Destination	Destination (end stop) of the passenger.
Direct Trip Distance	Trip distance if the trip were carried out directly, i.e., without detours.
Assigned Shuttle	The shuttle to whom the passenger is assigned.
Is Shared	Shows if the passenger shares its trip with another passenger, or not.
Request Timestamp	Point of time when the request is generated.
Assignment Timestamp	Point of time when the passenger is assigned to a shuttle.
Pick-up Start Timestamp	Point of time when the passenger is picked up (start time).
Pick-up End Timestamp	Point of time when the passenger is picked up (end time).
Drop-off Start Timestamp	Point of time when the passenger is dropped off (start time).
Drop-off End Timestamp	Point of time when the passenger is dropped off (end time).
Denial Timestamp	Point of time when the passenger is denied.

$$\text{New Distance} = \text{Distance of the Temporary Path} - (\text{Total Mileage} - \text{Distance Checkpoint}) \quad (4.1)$$

4.2.3. Description of the External Python Code

The external Python code serves to capture the peculiarities of the on-demand simulation. The code has three principal functions: **init**, **main**, and **export** (see Figure 4.11). The **init** function is run only at the beginning of the simulation to perform initializations, the **main** function is the main body and is called during the simulation at each simulation time step, lastly the **export** function is run only at the end of the simulation and is responsible for exporting the KPIs and operational parameters to .csv files.

In the **init** function, firstly the operational parameters are set. These include the on-demand fleet size, shuttle capacity, maximum waiting time, and maximum detour ratio. Moreover, the request pool is imported from a .csv file. Then, the shuttles are initialized at the depot locations, with occupancies of 0 and appropriate shuttle IDs. Next, the graph of the network is retrieved as a matrix. Finally, Dijkstra's shortest route algorithm is applied to this graph to obtain the shortest paths and their costs between each and every node (see Section 4.2.2.3).

The key part of the code is located in the **main** function. Here, at each simulation time step, firstly the shuttle attributes are updated. These include, for instance, the attributes of *idle time*, *occupied time*, *empty drive kilometer*, *previous mileage*, or *shuttle status*. Then, for each assignment phase (The assignment phase differs depending on the type of the assignment algorithm: in spontaneous assignment, it is each point in time when a request arrives; whereas in batch assignment, it is the end of each batch period.), the current positions of the shuttles are obtained. Parallel to that, new passengers are created in Vissim as *vehicle* object, with several attributes such as the *passenger ID*, *origin*, *destination*, and *direct trip distance*. Next, for each shuttle, the current state regarding the vehicle and passenger side is identified. The vehicle side consists of the duration of the current vehicle tour, and the passenger side consists of the waiting and in-vehicle travel times of the passengers currently assigned.

As the next step, options are created according to the applied matching methodology. From these options, the ones violating the operation rules (maximum waiting time and maximum detour ratio) and shuttle capacity requirements are filtered out. Furthermore, if a shuttle is about to arrive at a stop for pick-up/drop-off and has passed over the “*parking routing decision*”, this stop event has to be realized first and the shuttle should not be sent to any other stop. Likewise, if a shuttle is currently picking up/dropping off a passenger, this event should be completed first, before it is sent to any other stop. Therefore, in both of these cases, the options that initially send the shuttle to another stop are filtered out as well. After these filtering steps, the remaining options can be considered as the feasible options.

Subsequently, for each feasible option, the new state regarding the vehicle and passenger side is identified. Here, similar to the current state specified above, the vehicle side consists of the duration of the vehicle tour in case the option is accepted, and the passenger side consists of the waiting and in-vehicle travel times of all passengers in the option, also assuming the option is accepted.

Among all feasible options, the best match is determined depending on the assignment algorithm. While in the spontaneous algorithm, one single request is assigned to one single shuttle with the best feasible option; in batch algorithm, several requests are assigned to (possibly) several shuttles with the best combinations.

After the assignments are finalized, the attributes of shuttles and passengers are updated, e.g., the passenger status is changed to *assigned&waiting* or *denied*, and the planned tour of the shuttles are updated. In addition, the newly emerging paths are assigned to the corresponding shuttles, leading to possible new movements of the shuttles in the next simulation time steps.

The next block of the **main** function is dedicated to the pick-up/drop-off (PUDO) events of the shuttles. For this, each shuttle is checked at every simulation time step to determine whether it has just completed a pick-up or drop-off event, i.e., it has stopped at a shuttle stop and dwelled for the duration of the boarding/alighting time. If this is the case, then primarily the shuttle attributes “*passengers in shuttle*” and “*passengers to-be-picked-up*” are updated depending on whether the event is a pick-up or a drop-off. Then, certain attributes of shuttles and passengers are updated, for example the “*occupancy*” and “*distance checkpoint*” attributes of shuttles; as well as the “*passenger status*”, “*is shared*” and “*drop-off/pick-up end timestamp*” attributes of passengers.

In the next step, the attributes “*planned tour*” and “*assigned option*” are updated by removing the first list-elements, since the first event has just been completed. Afterwards, the shuttle is sent to the subsequent stop for the next event, if there is one. If the next event also takes place at the current stop (for instance, a pick-up event right after a drop-off event at the same stop), then the shuttle stays at that stop for the next boarding/alighting time. In case there is no other next stop in the planned tour, then the shuttle remains at that stop in idle mode, until it receives a new assignment.

While a shuttle waits at a stop in idle mode, it occupies the stop and prevents the other shuttles from using it. To avoid this, the stop needs to be freed again. Here, an approach is used which involves sending the idle shuttle to the depot whenever another shuttle is about to arrive. The idle shuttle therefore makes an empty drive to its depot point, and the next shuttle can use the stop for its next pick-up/drop-off event.

In the **export** function of the code, important system KPIs, including service rate, sharing rate, total mileage, mean waiting time, mean occupancy, or empty mileage ratio, are exported to .csv files along with the operational parameters used. The **export** function is activated only at the very end of each simulation run.

An overview of the external Python code with the aforementioned functions is illustrated in Figure 4.11.

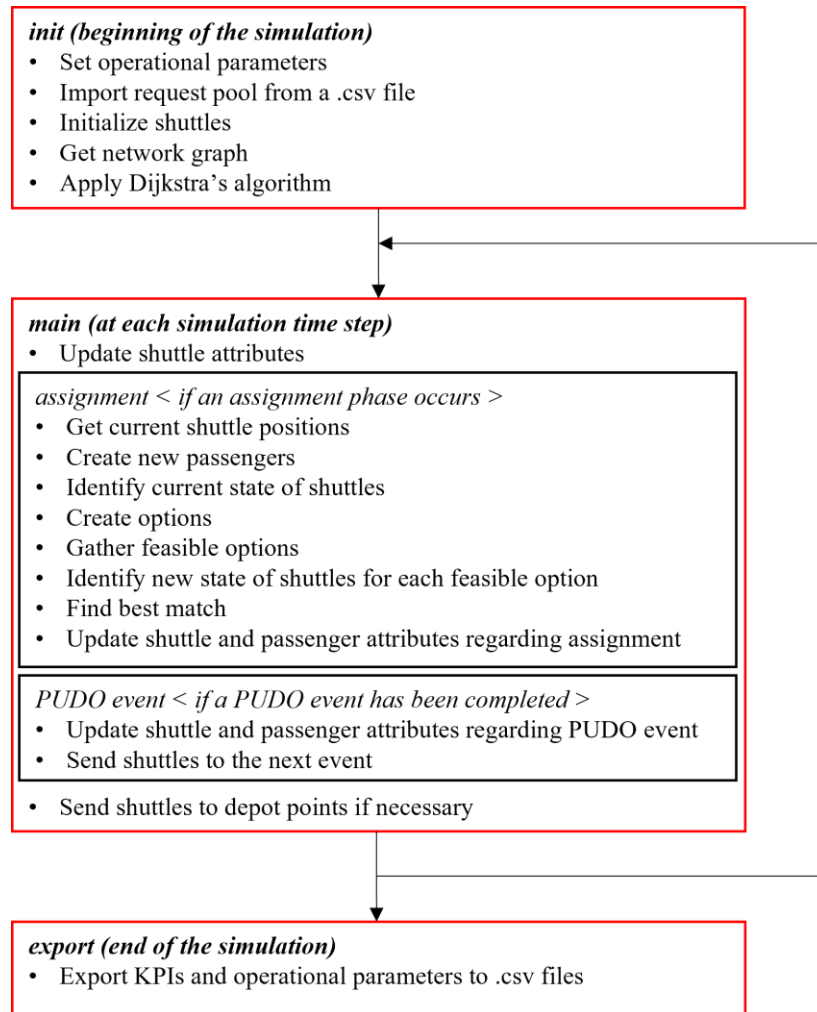


Figure 4.11: Overview of the external Python code.

4.2.4. Calibration of the Simulation

To ensure the accuracy and realism of the simulation model, a calibration process is required. This process involves aligning the simulated system behavior with real-world observations by adjusting key model parameters. During calibration, the differences between the simulated and observed variables are minimized by finding an optimum set of parameters (Daamen et al., 2015).

The calibration of the simulation was performed in two steps. In the first step, the travel times were calibrated by conducting test trips in the real world and sending probe vehicles into the simulation. These values were compared for two main routes on the campus, in both directions. Both routes pass the inner-section barrier that regulates access to the inner section of the campus. As no congestion generally prevails on the campus, this is the point where most of the delays occur. The incoming barrier is opened either by a smart card, which the university members hold to a device, or by campus security remotely if the user has been authorized manually in advance. In both cases, opening of the barrier takes a while and causes delays, especially when multiple vehicles want to go through. The outgoing barrier is opened automatically as soon as a vehicle is detected nearby, and this opening takes relatively less time due to minimal human intervention in the process. According to the results of the test trips, the stoppage times at the barriers were then adopted in the simulation. These stoppage times were applied to the other vehicle barriers at the main gate, north gate, and east gate as well, since there is almost no difference among the barriers regarding driver behavior or the working principle of the barriers.

As a second step, vehicle speeds were calibrated using camera data. For this, four important street sections were identified where camera coverage was possible. In both directions, at eight spots in total, observed car speeds were recorded in .csv format for 7:00-9:00 a.m. morning peak time on a typical weekday. For each spot, a cumulative speed distribution was created in the simulation. These distributions were then incorporated into the “*desired speed decisions*” object in the simulation, upstream of that spot, to compel motorized vehicles to obtain a speed probabilistically, as reflected in the speed distribution.

In parallel, *data collection points* were set in the simulation for the eight spots to generate car speed data. The simulated data and the observed camera data were compared using independent sample t-tests to determine whether significant differences existed, with a significance level 0.05. During the calibration process, the speed distributions in the simulation were fine-tuned to achieve a good fit. At the end of the calibration process, it was confirmed by the statistical t-tests that both datasets matched for all eight spots, and calibration was therefore completed.

Two such cumulative speed distributions used in the simulation are shown in Figure 4.12. The left-hand figure corresponds to a shared space spot, and the right-hand one to the main road of the campus. The observed mean speed at the shared space spot is 21 km/h, although in shared spaces, a speed of 8-12 km/h would be expected. Similarly, the observed mean speed at the main road spot is 38 km/h, where a speed limit of 30 km/h is typically enforced. It can therefore be inferred from these analyses that motorized vehicles tend to drive faster than the compulsory speed limits on the campus.

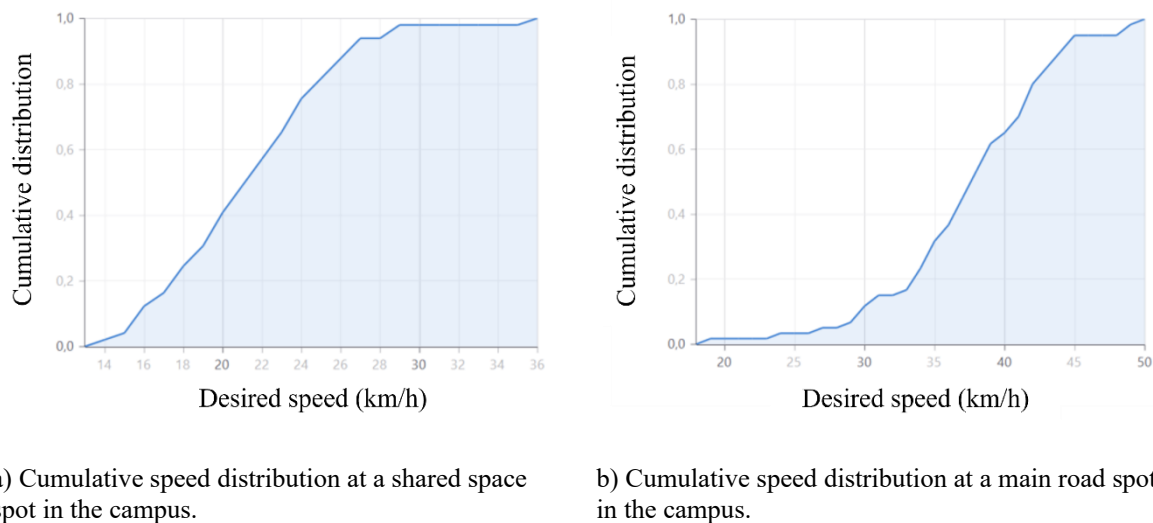


Figure 4.12: Cumulative speed distributions used at two spots in the simulation.

4.3. Generation of a Request Pool

Another essential point in the on-demand shuttle simulation is generating a request pool. In order to mimic the on-demand travel demand of passengers within the campus, the microsimulation had to be fed with trip requests. Since the campus has its own peculiarities, e.g., certain origin-destination pairs are more prevalent depending on the time of the day, the use of a realistic dataset was deemed necessary.

Comparable ride-pooling studies have mostly used either historical taxi data (Alonso-Mora et al., 2017; Engelhardt et al., 2022b; Liu & Samaranayake, 2022), TNC data (Engelhardt et al., 2024) or a portion

of private trips (Engelhardt & Bogenberger, 2021) for the expected demand. Unfortunately, such data, or any other travel demand model is not available for the campus; furthermore, many people use transport modes other than private cars (bicycle, walking, micromobility).

On the other hand, the campus has a micromobility sharing system that provides mobility solutions for university members. This platform, known as MORE Sharing, is operated as a free-floating system, where registered users can book or reserve (up to 15 minutes before the trip) e-scooters, e-bikes, e-cargo bikes, city bikes, and e-mopeds for their trips via a smartphone app (Pobudzei et al., 2023). The service is available not only within the campus, but also outside of it, if the booking starts on campus. In any case, the booking must end on campus. However, parking outside the campus is possible. For parking events, the users have to enable parking via the app and disable it when they want to continue using the vehicle. In addition, the entire system is free of charge to promote micromobility usage and generate valuable mobility data for the campus. Therefore, for this thesis, the trip base of the MORE Sharing system for the year 2023 was utilized, a similar approach to that used in Engelhardt et al. (2022a) regarding the Manhattan taxi trips. The data analysis was conducted using Python *pandas* package.

The MORE Sharing back-end system stored all the data concerning bookings within the data table “*reservations*”. Each booking had the attributes *start_time*, *end_time*, *start_coordinates*, *end_coordinates*, *driven_distance* and *park_time*. In cases where a booking was transacted but a trip was not performed eventually, *driven_distance* was assigned the value 0. These data entries were filtered out from the dataset.

Normally, the following data would suffice to gather the travel demand of on-demand ride-pooling: *start_time* as the request time, *start_coordinates* as the starting point, and *end_coordinates* as the end point of the request. However, a deeper look into the dataset revealed that many bookings involved parking events. This occurred frequently in the sharing system, mostly because the system was free of charge, which meant a user would prefer to park the vehicle instead of ending the booking, knowing there would be no surcharge. Such reservations would then be composed of multiple trips. For example, a sharing trip starting from A travels to B, and the vehicle is parked there for 10 minutes, then with the same vehicle (and the same reservation), travels back to A. In the sharing data (in *reservations* table), this is considered a single trip, starting and ending at A. However, for the on-demand shuttle system, this would represent two different trip requests: one from A to B and one for the return trip. In order to take this aspect into account as well, other relevant tables needed to be investigated, too, which will be elaborated in the following parts.

Parking data were available in the data table “*logs*”. Here, all the operational logs concerning the vehicles were stored, such as locking the vehicle, starting parking, sending feedback, restoring GPS connection, etc. However, the information was not very clear, and the data needed to be preprocessed. An example of log data containing five fields is given as follows:

```
{
  "logId": "9632538",
  "timestamp": "2023-10-09 14:03:16",
  "eventName": "getIdleTime",
  "logData": "[\"2023-10-Mon 13:27:06 - START\", \"2023-10-Mon 13:30:41 - Parking ON Idle Time: 0 (+0s)\", \"2023-10-Mon 13:43:44 - Parking OFF Idle Time: 783 (+783s)\", \"2023-10-Mon 13:44:52 - Parking ON Idle Time: 783 (+0s)\", \"2023-10-Mon 13:50:43 - Parking OFF Idle Time: 1134 (+351s)\", \"2023-10-Mon 13:56:50 - Parking ON Idle Time: 1134 (+0s)\", \"2023-10-Mon 13:56:53 - Parking OFF Idle Time: 1137 (+3s)\", \"2023-10-Mon 14:03:16 - TOTAL IDLE TIME: 1137\"]",
  "logType": "2"
}
```

In this data, the field “*timestamp*” shows the end time of the relevant booking. The useful information to be processed for parking events is contained in the field “*logData*”. The idea is to obtain the end of the parking event, which represents both the end of the first trip and the start of the following trip. The content “*Parking OFF*” provides this information. In the example above, there are three such contents, which implies that the vehicle was parked three times during the same booking. Indeed, a vehicle could be parked multiple times during a booking, resulting in a trip chain, and this chain might include more than two trips.

The following steps were applied in the given order, to clean the data:

- **Filter out the rows without parking information in the “*logData*” field:** Only the relevant data containing *Parking OFF* remained.
- **Remove all information except *Parking OFF* in the “*logData*” field:** The exemplary log data appears then as follows:

```
{
  "logId": "9632538",
  "timestamp": "2023-10-09 14:03:16",
  "eventName": "getIdleTime",
  "logData": "[\"\", \"2023-10-Mon 13:43:44 - Parking OFF Idle Time: 783 (+783', \"\", \"2023-10-Mon 13:50:43 - Parking OFF Idle Time: 1134 (+351', \"\", \"2023-10-Mon 13:56:53 - Parking OFF Idle Time: 1137 (+3']\",
  "logType": "2"
}
```

- **Within one “*logData*” entry, filter out parking events with a duration less than 60 seconds:** Since some parking events were too short to be identified as actual parking, a minimum parking duration threshold of 60 seconds was applied. All parking events shorter than that were ignored, and the relevant part of the booking was considered as a complete trip. In the exemplary log data, the parking events lasted 783, 351, and 3 seconds, respectively. The last parking event is removed since it is below the threshold:

```
{
  "logId": "9632538",
  "timestamp": "2023-10-09 14:03:16",
  "eventName": "getIdleTime",
  "logData": "[\"\", \"2023-10-Mon 13:43:44 - Parking OFF Idle Time: 783 (+783', \"\", \"2023-10-Mon 13:50:43 - Parking OFF Idle Time: 1134 (+351']\",
  "logType": "2"
}
```

- **Extend the “*logs*” table with new fields for each parking event:** For each parking event in a row, new fields were added to the table. In the exemplary data, two fields were added since there are two parking events. In fact, the table was extended with as many fields as the maximum number of parking events in any row of the table. In the entire dataset, this corresponded to 26 parking events; hence, the table was extended with 26 fields. However, for demonstration purposes, only the relevant two fields of the exemplary data are shown here (the remaining fields will contain *None* values):

```
{
  "logId": "9632538",
  "timestamp": "2023-10-09 14:03:16",
  "Park_0": \"\", \"2023-10-Mon 13:43:44 - Parking OFF Idle Time: 783 (+783\",
  "Park_1": \"\", \"2023-10-Mon 13:50:43 - Parking OFF Idle Time: 1134 (+351\"
}
```

```
}
```

- **Leave only the date and time information in the “*Park_x*” columns:** The remaining text was removed from the fields:

```
{
  "logId": "9632538",
  "timestamp": "2023-10-09 14:03:16",
  "Park_0": "2023-10-Mon 13:43:44",
  "Park_1": "2023-10-Mon 13:50:43"
}
```

- **Convert the weekday text in the “*Park_x*” columns to a numerical value:** The raw data for parking events had numerical values for year and month, but the day value was given as a three-letter text, such as “Mon” in the above example. For the conversion, a simple logic was used: Since the “*timestamp*” field shows the end time of a booking, and parking events must occur before that, the first matching weekday prior to the end time was identified. In the given example, it corresponds to the ninth day of the month:

```
{
  "logId": "9632538",
  "timestamp": "2023-10-09 14:03:16",
  "Park_0": "2023-10-09 13:43:44",
  "Park_1": "2023-10-09 13:50:43"
}
```

Although the log data was prepared ready for parking events, an important deficiency of the data table was that it did not include the booking ID explicitly. Therefore, the bookings and parking events could not be linked directly to each other. To overcome this issue, the *timestamp* field was used. As mentioned before, this field showed the end time of the relevant booking. Using this field and the *end_time* field of the *reservations* data table, a merge became possible. Nevertheless, merging became problematic and could produce errors if two bookings had the same end times. For this reason, a manual check of such cases and data correction to guarantee a correct merge was required. After merging the two tables, the resulting exemplary data appeared as follows:

```
{
  "reservationId": "74866",
  "startTime": "2023-10-09 13:26:34",
  "endTime": "2023-10-09 14:03:16",
  "startLocation": "(11.63372, 48.08065)",
  "endLocation": "(11.63352, 48.08024)",
  "logId": "9632538",
  "Park_0": "2023-10-09 13:43:44",
  "Park_1": "2023-10-09 13:50:43"
}
```

The next step was to find out the parking locations, since the data above involved only parking timestamps. In order to transfer this information to the location, another data table, “*geolocations*”, had to be utilized. The *geolocations* table included the geolocation data sent to the central system by vehicles at certain intervals, and was composed of the fields *carId* (vehicle ID), *reservationId* (booking ID), *timestamp* as well as *location_coordinates*. It was therefore possible to form the trajectory of a vehicle (or a booking) and fetch its coordinates at a specific time of day.

Firstly, the rows in the data table without *reservationId* were removed, since only the locations of vehicles during a booking were of interest. Upon closer inspection, it was observed that some of the coordinate data were faulty, the *location_coordinates* of a vehicle sometimes jumped abruptly from one location to another, and then jumped back to the original location. This occurred frequently at the end of a parking event. In order to avoid false analysis at the parking events, a logic was introduced to remove the faulty data: For each *reservationId*, the dataset was sorted in ascending order of timestamps. If there was a pattern like A, A, B, A for *location_coordinates* within the timestamp sequence, then the row involving B was removed. Tests conducted on the dataset have shown that this proved to be an appropriate window check.

An exemplary situation is illustrated in Table 4.4. Here, vehicle 25, serving reservation 1118, sends geolocation data approximately every 10 seconds, whereas the third location jumps to another coordinate and then jumps back. In this case, the third row is dropped, and it is ensured that the location coordinates remain consistent.

Table 4.4: An example of faulty data (third row) in the geolocations data table.

carId	reservationId	location_coordinates	timestamp
25	1118	(48.080217, 11.632621)	2023-03-07 18:27:03
25	1118	(48.080217, 11.632621)	2023-03-07 18:27:13
25	1118	(48.121564, 11.611996)	2023-03-07 18:27:25
25	1118	(48.080217, 11.632621)	2023-03-07 18:27:32

At this point, using the geolocations data made it possible to obtain the location of a vehicle (or a booking) at a specific point of time, especially during parking events. For this purpose, another merge was performed between the data table created after the previous merging and *geolocations*. The resulting exemplary data after the merging process is as follows:

```
{
  "reservationId": "74866",
  "startTime": "2023-10-09 13:26:34",
  "endTime": "2023-10-09 14:03:16",
  "startLocation": "(11.63372, 48.08065)",
  "endLocation": "(11.63352, 48.08024)",
  "logId": "9632538",
  "Park_0": "2023-10-09 13:43:44",
  "Park_1": "2023-10-09 13:50:43",
  "Location_0": "(11.64186, 48.0976)",
  "Location_1": "(11.6377, 48.0815)"
}
```

Consequently, it could be inferred for every booking where the parking event took place and when the parking ended. If there were multiple parking events, they could also be observed in the dataset. For example, the data above involves two parking events, hence three trips. Subsequently, a single trip dataset was created taking the multiple trips of the bookings into account. This dataset consisted of the *reservationId*, *start_time*, *start_location* and *end_location*. The resulting exemplary data containing three trips is displayed as follows:

```

{
  {
    "reservationId": "74866",
    "startTime": "2023-10-09 13:26:34",
    "startLocation": "(11.63372, 48.08065)",
    "endLocation": "(11.64186, 48.0976)",
  }
  {
    "reservationId": "74866",
    "startTime": "2023-10-09 13:43:44",
    "startLocation": "(11.64186, 48.0976)",
    "endLocation": "(11.6377, 48.0815)",
  }
  {
    "reservationId": "74866",
    "startTime": "2023-10-09 13:50:43",
    "startLocation": "(11.6377, 48.0815)",
    "endLocation": "(11.63352, 48.08024)",
  }
}

```

Since the on-demand shuttle system had 16 stops, as explained in Section 4.2.2.4, the campus was divided into 16 regions, where each region represents the catchment area of a stop, i.e., if a MORE Sharing trip originates from a coordinate within a certain region, it would mean that a shuttle request will occur from the corresponding stop (the same applies for the destination as well). Therefore, a matching was applied to the *start_location* and *end_location* fields of the single-trip dataset. The regions were determined based on the ease of accessibility to individual stops, such as walking possibilities, and are displayed in Figure 4.13. The regions were created in Google Earth and generated in Python as polygons with their coordinates, in order to check whether any location data fell within one of the regions.

Ultimately, a single database consisting of trips, which were divided by the parking events, was created. This dataset included trips with start time, start stop and end stop. Two filters were applied to this database:

- only the trips starting and ending on the campus were considered,
- trips with the same start and end stop were omitted, since that would imply a demand from and to the same stop in the shuttle system, which would not be meaningful.

The MORE Sharing data was analyzed from the system launch (06.03.2023) until 05.11.2023, for an 8-month period. For this period, 24,425 trips were identified which occurred within the campus and between different campus regions. The number of trips per calendar day is shown in Figure 4.14a. Here, one can observe the low demand during the weekends, and generally higher demand in spring compared to the other seasons. For the on-demand shuttle system demand, the day with the highest number of trip requests was taken as the base, which is 21.03.2023. Since the travel demand on that day before 7 a.m. was very scarce, only trips after 7 a.m. until midnight were considered. This resulted in a simulation run of 17 hours in total, covering 385 trip requests.

The demand profile, displaying how the demand varies over the hours of the highest-demand day is shown in Figure 4.14b. The number of trips is high during the day, and especially higher in the early afternoon, which is likely due to activities such as attending lectures or traveling to and from the restaurants at lunchtime.

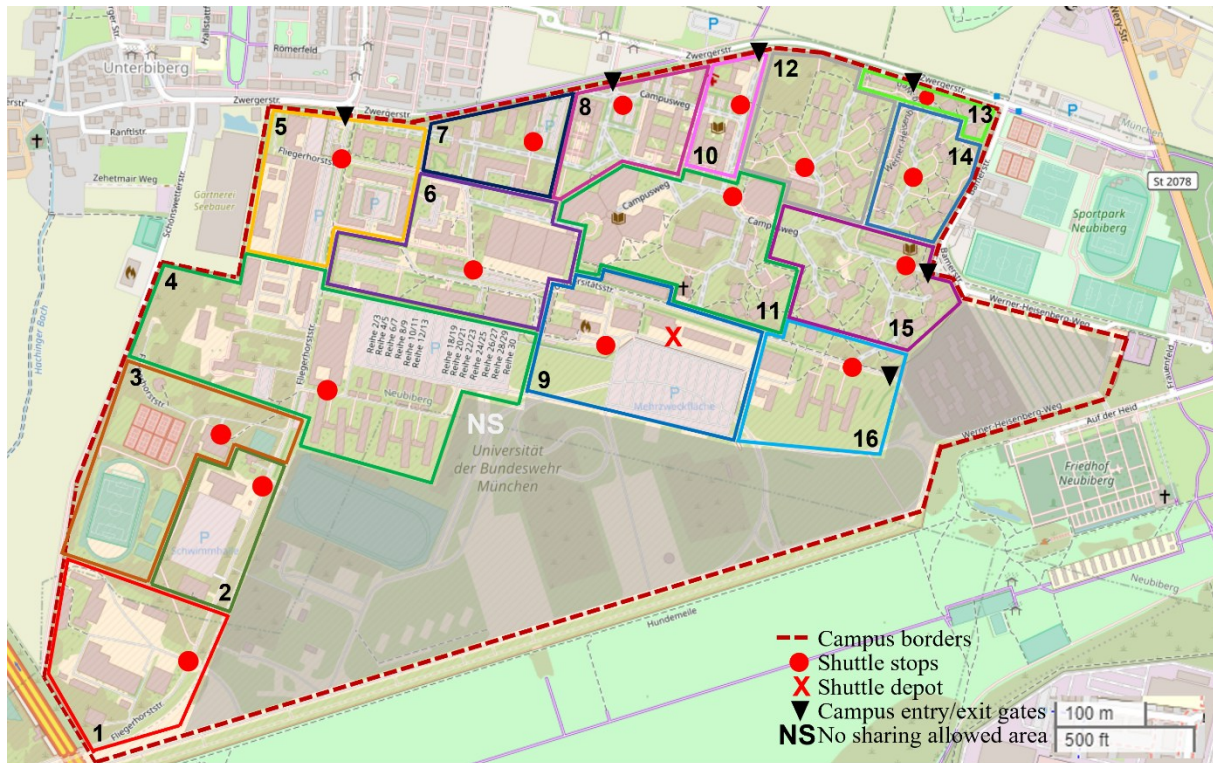
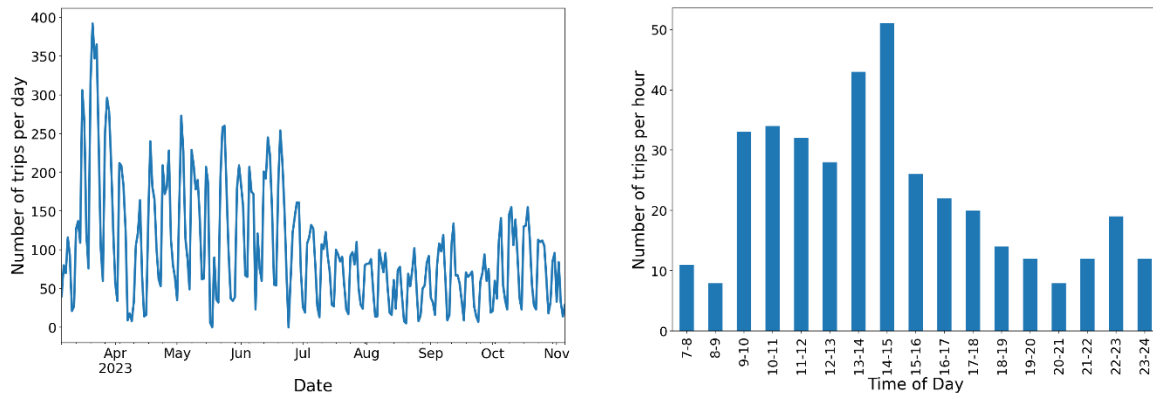


Figure 4.13: 16 regions in the campus and corresponding shuttle stops (OpenStreetMap).



a) Number of MORE Sharing trips inside the campus per calendar day. b) Demand profile of the highest-demand day.

Figure 4.14: Distribution of MORE Sharing trips.

For the simulation experiments involving batch assignment, a more complex operation was envisaged (see Section 5.3). Therefore, the demand was increased: instead of using a single day with the highest number of trip requests, the three highest demand days were selected, resulting in almost tripling the on-demand travel demand. The details of this implementation are provided in Section 5.3.

With this methodology, a request pool was generated for the daily simulation of on-demand shuttles. The start time of the MORE Sharing trip corresponded to the request time in the on-demand system, as well as the start stop and end stop of the sharing trip corresponded to the origin and destination of the request, respectively. An overview of the processes explained in this section is depicted in Figure 4.15.

One drawback of this approach is that the start point of the sharing reservations does not necessarily translate to the request start point. In MORE Sharing, users check for the nearest vehicle, walk to this

vehicle, and start their trip from there. This implies that the user may have initially been somewhere other than the booking start point. However, MORE Sharing data involves only the booking start point, and not the point from which the users send their request. On the other hand, users tend to have short walking distances to reach a vehicle; therefore, it can be assumed that they were most of the time close to the vehicle, or at least in the same region.

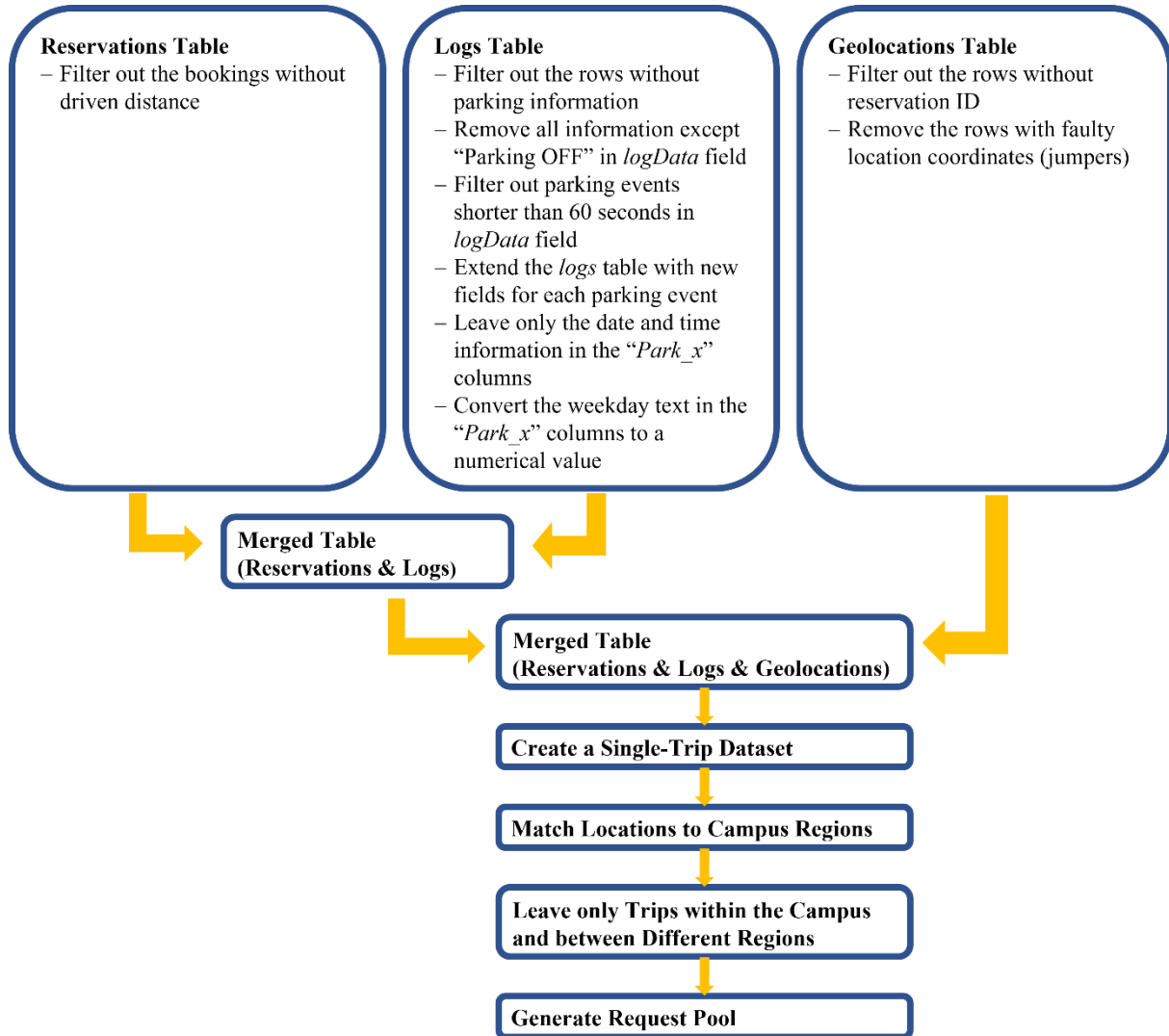


Figure 4.15: Overview of the processes to generate a request pool.

4.4. Simulation Workflow

Figure 4.16 shows the six essential steps for building a multimodal microscopic traffic simulation that incorporates an on-demand ride-pooling system, an approach that can be replicated in similar future applications. After setting up the simulation for both regular traffic and on-demand shuttle operations, trip requests for the on-demand component are generated, and request-based matching algorithms are developed. The simulation is then executed using these inputs, during which requests are assigned to shuttles and shuttle routes are updated accordingly. Finally, simulation outputs are collected and analyzed.

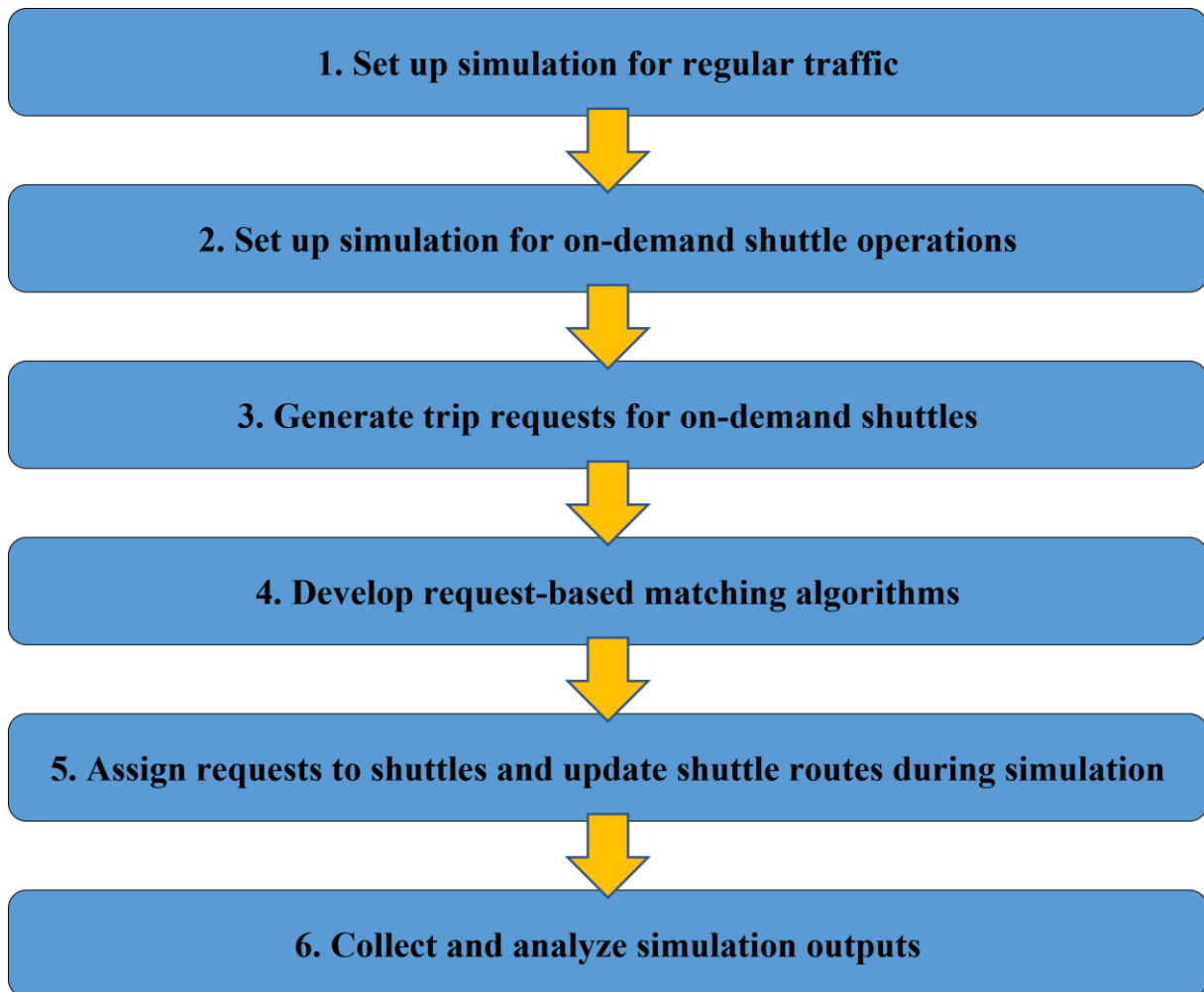


Figure 4.16: Steps for building a multimodal microscopic traffic simulation with on-demand ride-pooling.

The entire workflow of the simulation utilized in this research is illustrated in Figure 4.17, which also serves as a summary of this chapter. As the first block, the pre-processed camera data feeds the simulation with vehicle inputs, vehicle compositions, routing decisions, and routes. This also addresses **Research Question 3** of this thesis: “**How to build an interface between a camera-based traffic data collection system and the microsimulation model, to automate the vehicle data feed?**”.

Secondly, the simulation is set up using standard Vissim objects, as in other microsimulation models. An essential addition here is the representation of the network as a graph, equipped with nodes and edges. In this way, shortest route algorithms such as Dijkstra’s can be applied to the network.

The matching algorithms, which were explained in detail in the previous chapter, are programmed externally. Their development and integration into the simulation together with the dispatching algorithm form the third block of the workflow.

Lastly, the generation of a request pool to mimic the travel demand of the on-demand shuttle system forms the fourth block. Based on the MORE Sharing data, the dataset is processed and trips are extracted, which then feed the simulation as shuttle requests.

All these four blocks interface with the microsimulation via the COM interface, underlining its importance for the entire modeling of the campus.

In addition, there are two supplementary data inputs to the simulation. The first is the traffic survey data for areas of the campus with insufficient coverage from traffic cameras. Here, all values related to vehicle inputs and routes are prepared in a spreadsheet and entered manually into the simulation. The second input is the vehicle speed distributions, which are used for calibrating the simulation.

The microsimulation then runs with all these inputs, shuttles are dispatched to different locations according to their assignments, and finally, the simulation delivers outputs for the system KPIs used in the analyses. Overall, this workflow addresses **Research Question 1** of this thesis: “**How can an on-demand ride-pooling system be implemented in a microsimulation model, specifically in Vissim?**”

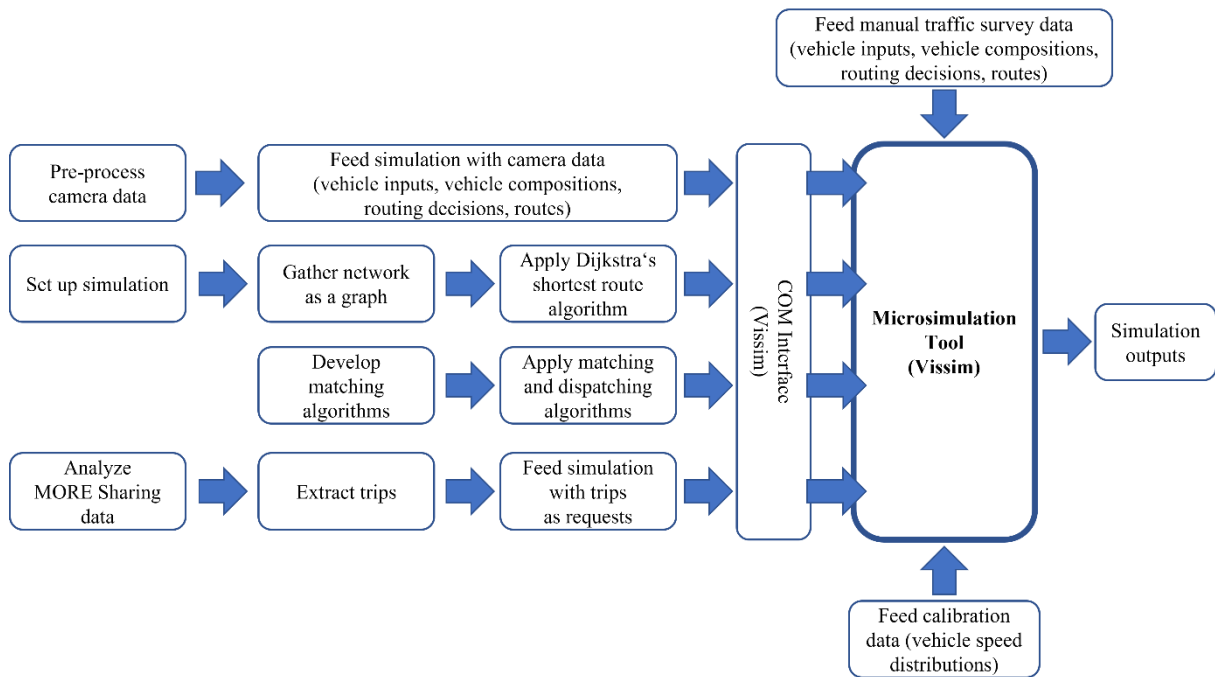


Figure 4.17: The workflow of the simulation utilized in this research.

5. Simulation Experiments and Results

Chapter 3 outlines two matching methodologies for an on-demand shuttle system, while Chapter 4 details the implementation of this system within a multimodal microsimulation environment of the university campus. Building on these foundations, this chapter presents the simulation experiments and discusses the results.

The chapter begins with a description of the assumptions made for the simulations, followed by the determination of the required number of simulation runs to ensure statistical significance. The results of the spontaneous assignment approach are then presented, focusing on the effects of varying operational parameters. Next, system KPIs are analyzed under different circumstances using the batch assignment approach, including a comparative evaluation of both assignment methodologies. The chapter concludes with a summary of the key findings from the simulation experiments.

5.1. Initial Setup

As explained in Section 4.1, the simulations were performed on the campus of the University of the Bundeswehr Munich. The Vissim simulation was set up for both on-demand mode and other transport modes, following the detailed guidelines described in Section 4.2.

Prior to initiating the simulation runs, it was necessary to feed the simulation model with camera data for all transport modes other than on-demand. A representative weekday during the university lecture period was required, ensuring realistic vehicle and pedestrian movement patterns, along with complete camera data available for that day. Based on these criteria, Thursday, 13 March 2025, was selected as the reference day.

The maximum speed for the on-demand shuttles was set to 15 km/h, reflecting the future implementation of autonomous shuttle services on campus. Furthermore, the boarding and alighting time of passengers were both set to 30 seconds. In case of consecutive boarding and/or alighting events at a stop, 30 seconds were applied to each individual event.

The simulation resolution was defined as 3 time steps per simulation second, and the total simulation period covered 17 hours, from 7:00 a.m. to 11:59 p.m., as outlined in Section 4.3.

The cost functions of both matching methodologies involve weighting parameters both for operator's and passenger's perspective, allowing to place more emphasis on either side (see Equation (3.4) and (3.5)). In the simulation experiments, these weightings were set equally (default case) to reflect a balanced perspective. Only in Section 5.3.3, these values were varied in order to analyze the impact of different weightings.

5.1.1. Required Number of Simulation Runs

The external code developed for the on-demand shuttle simulation controls the movement of shuttles and performs shuttle-request matchings in a rule-based approach. As a result, the simulation runs involving only shuttles remain the same even when different random seeds are applied. However, the simulation of other transport modes involves stochastic elements, such as the timing of vehicle entries into the network, turning decisions at intersections, or selection of desired speeds. These random

variations influence the dynamics of the overall system, including interactions with the on-demand shuttles.

To adequately capture this randomness and ensure the statistical robustness of the results in the entire simulation, multiple simulation runs with different random seeds had to be conducted. The required number of runs was determined using the methodology outlined in the German guidelines for microscopic traffic simulation studies (FGSV, 2006). The corresponding formula for estimating the necessary sample size is given as follows:

$$n \geq \frac{t(\alpha, n-1)^2 \cdot s^2}{CI^2} \quad (5.1)$$

where

- n is the required number of simulation runs,
- α is the desired confidence level,
- $t(\alpha, n-1)$ is the value from the Student's t-distribution table,
- s is the standard deviation of the KPI,
- CI is the desired confidence interval of the KPI.

A series of 10 simulation runs were initially conducted using constant parameters while varying only the random seed (see Table 5.1). From these runs, three important KPIs (service rate, mean waiting time, and total mileage) were selected to determine the required number of simulations. The desired confidence level (α) was defined as 0.95. The desired confidence intervals (CI) for the selected KPIs were set to 0.01, 5 seconds, and 10 kilometers, respectively. As $t(\alpha, n-1)$ depends on the n value, an iterative process was applied to satisfy the Equation (5.1).

Table 5.1: Determination of the required number of simulation runs.

	Service Rate	Mean Waiting Time (s)	Total Mileage (km)
Run #1	0.884	292.87	761.26
Run #2	0.872	294.52	771.45
Run #3	0.864	294.25	767.72
Run #4	0.872	292.05	767.03
Run #5	0.879	289.64	770.57
Run #6	0.875	300.00	763.25
Run #7	0.870	295.17	773.05
Run #8	0.876	296.73	773.12
Run #9	0.863	295.63	764.87
Run #10	0.879	290.76	756.87
Std. dev. (s)	0.006	2.87	5.11
Confidence Interval (CI)	0.01	5	10
Right-hand side of Eqn. (5.1) with n = 3	3.38	2.81	2.23
Right-hand side of Eqn. (5.1) with n = 4	2.19	1.82	1.44

The initial trial with $n=3$ proved insufficient, as the right-hand side of Equation (5.1) for the KPI “service rate” resulted in a value of 3.38, which is greater than 3. However, a subsequent iteration with $n=4$ fulfilled the condition, as the right-hand side values of Equation (5.1) for all KPIs are smaller than 4.

Consequently, it was concluded that a minimum of 4 simulation runs would be statistically sufficient. To ensure greater reliability, all experiments were ultimately conducted using 5 simulation runs.

5.2. Results of Spontaneous Assignment

This section presents the results of the simulation experiments performed using the spontaneous assignment. The details of the spontaneous matching methodology were explored in Section 3.2. As described in Section 4.3, the request pool comprised 385 requests distributed over a 17-hour period (7:00–24:00).

The objective of this section is to examine the influence of key operational parameters on certain system KPIs. Four operational parameters were considered in the analysis: Fleet size, Shuttle capacity, Maximum waiting time, and Maximum detour time (see the **init** function in Section 4.2.3). In each scenario, one parameter was varied while keeping the others constant. In this way, the impacts of a single parameter could be observed. The operational parameters and their values across the scenarios are summarized in Table 5.2.

Table 5.2: Operational parameters used in the scenarios.

Operational parameter	Definition of the parameter	Parameter values
Fleet size	Number of shuttles available in the simulation	[1, 2, 3, 4]
Shuttle capacity	Number of passengers who can travel in a shuttle at the same time	[1, 2, 3, 4]
Maximum waiting time (min)	Maximum acceptable duration from request submission to pick-up time	[3, 5, 10]
Maximum detour ratio	Ratio of the actual travel time (increased due to pooling) to the direct travel time	[1.4, 1.7, 2]

The KPIs analyzed in this section were selected based on the literature review on key performance indicators presented in Section 2.4.6. A total of 7 KPIs are examined, each of which is individually discussed in detail in the following sub-sections.

The results are presented using matrix plots arranged in a 3x3 layout (Arslan & Hoffmann, 2025a). In each subplot, the x-axis represents the parameter “**Capacity**”, while the y-axis corresponds to the specific KPI being analyzed. Each line within a subplot represents the parameter “**Fleet size**”. The rows of the matrix plots represent varying levels of the parameter “**Maximum waiting time**” (MWT), specifically 3, 5, and 10 minutes from top to bottom. Similarly, the columns of the matrix plots represent different values of the parameter “**Maximum detour ratio**” (MDR): 1.4, 1.7, and 2 from left to right.

The purpose of this analysis is to identify performance trends when changing a parameter. The impact of fleet size can be assessed by comparing the four lines within a subplot. The impact of capacity is observable along a single line within a subplot. Furthermore, the impact of MWT can be evaluated by comparing the subplots within the same column of the matrix plot. Similarly, the impact of MDR can be inferred by comparing the subplots across the same row of the matrix plot.

5.2.1. Impact on Service Rate

The KPI “service rate” is defined as the ratio of the number of served requests to the total number of requests. This metric is particularly important in on-demand systems, as fulfilling every request cannot be guaranteed, and some requests may need to be rejected. This occurs mostly due to the operational constraints of the system, e.g., no available shuttle can serve a new passenger within the allowed maximum waiting time. One of the main goals of the on-demand operation is therefore to maximize the service rate.

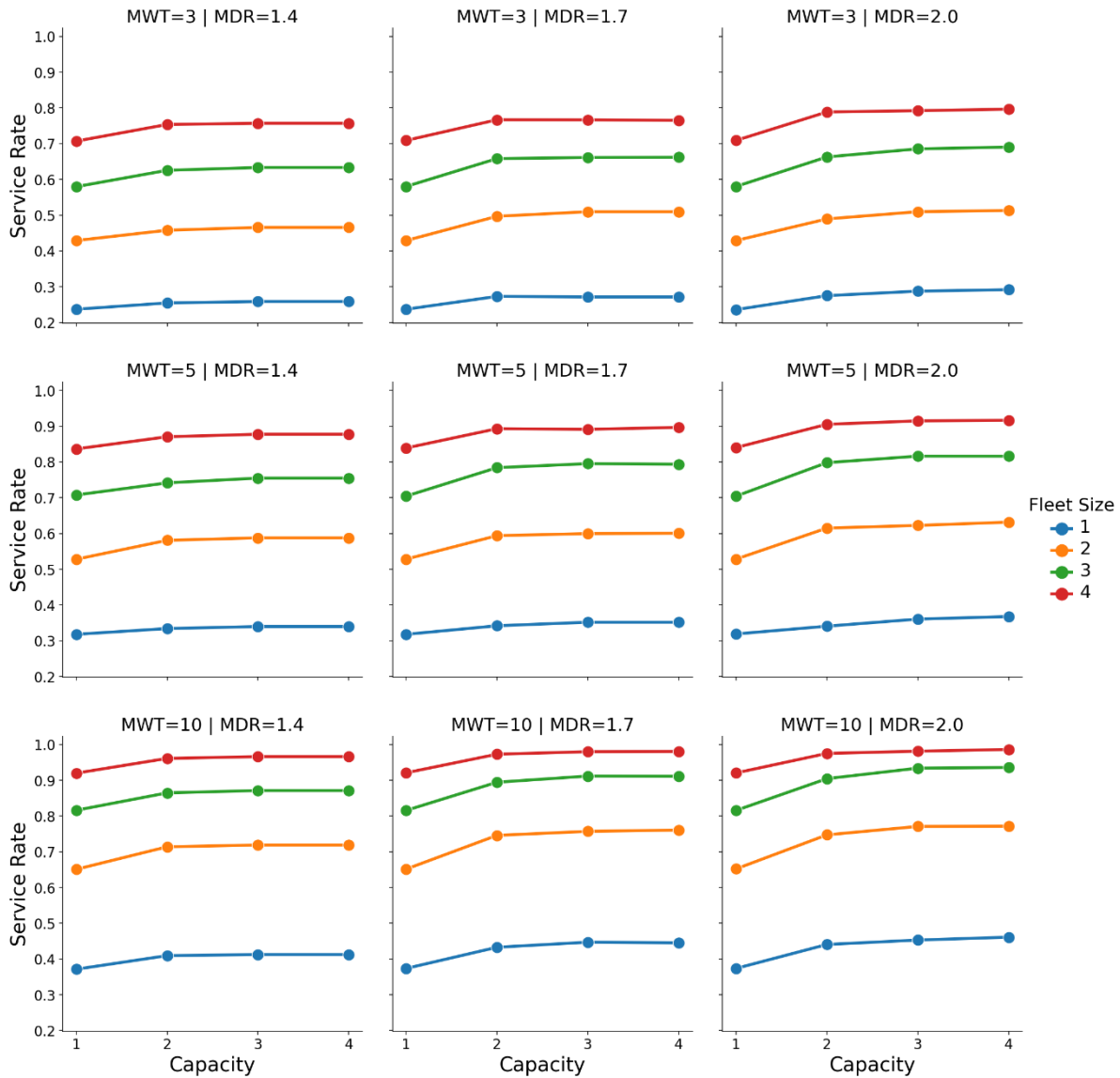


Figure 5.1: Impact of operational parameters on service rate.

As illustrated in Figure 5.1, the service rate increases with fleet size, with the most significant improvement observed when increasing the fleet size from 1 to 2 shuttles. Shuttle capacity also positively influences the service rate. However, the increase becomes marginal when capacity rises from 2 to 3, and nearly negligible from 3 to 4. This suggests that, in most cases, no more than 2 passengers were on board in a shuttle simultaneously.

A higher maximum waiting time (MWT) also has a positive impact on service rate, as greater passenger tolerance allows more time for an appropriate vehicle to become available. In contrast, variations in maximum detour ratio (MDR) appear to have little to no effect, service rate stays almost constant across

different MDR values. This implies that an MDR of 1.4 is already sufficient for the on-demand operation in terms of service rate.

The highest service rate (0.99) is achieved with a fleet size of 4, shuttle capacity of 4, maximum waiting time of 10 minutes, and maximum detour ratio of 2. Under this configuration, almost all requests can be served in the system.

5.2.2. Impact on Sharing Rate

The KPI “sharing rate” is calculated by dividing the number of requests that share a ride, by the total number of served requests. Sharing rate provides insights into the system’s shareability. A low sharing rate would infer more exclusive trips in the system. For the extreme case where capacity is 1, all trips are exclusive and there is no pooling, therefore the sharing rate is always equal to 0.

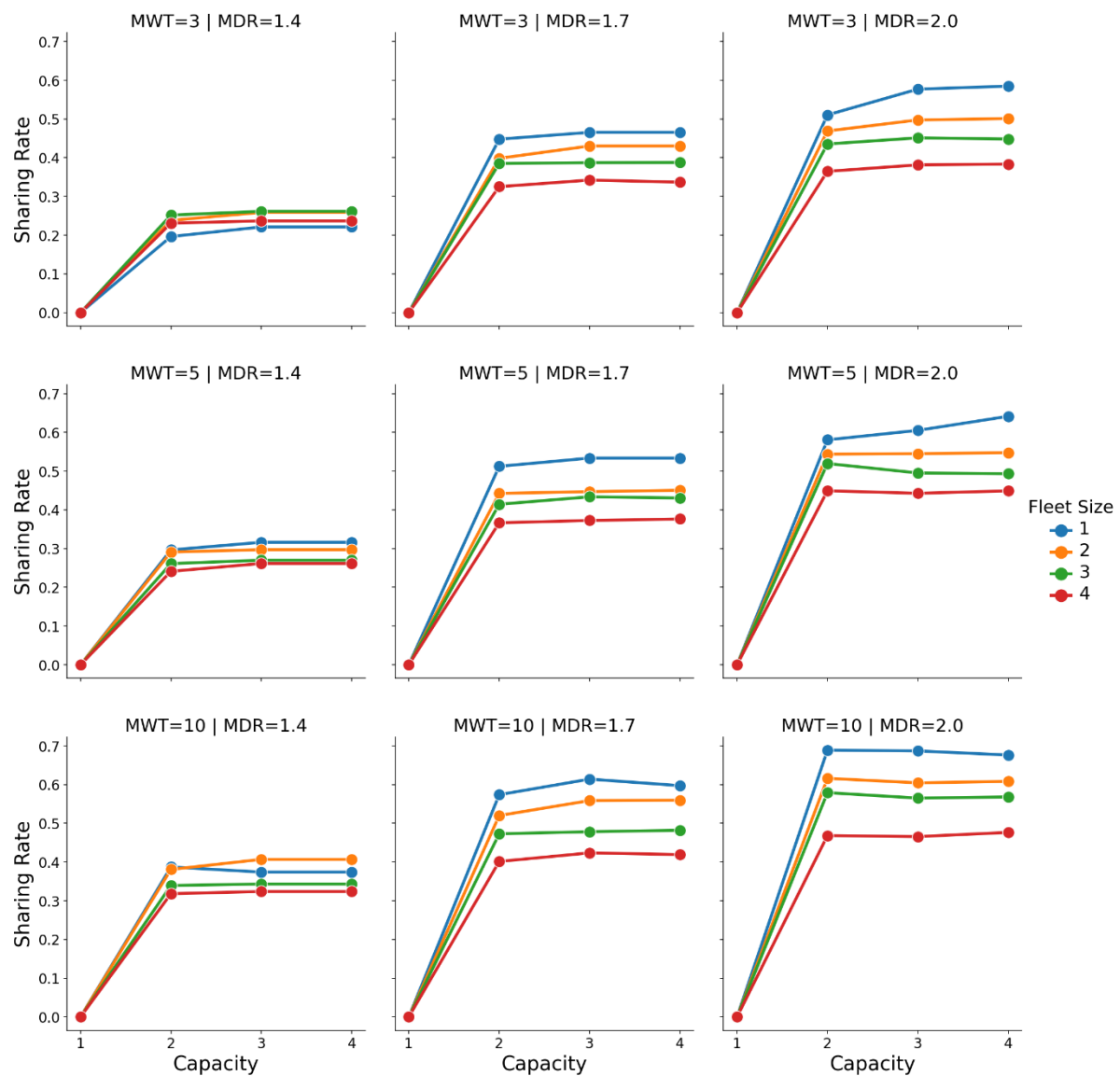


Figure 5.2: Impact of operational parameters on sharing rate.

Figure 5.2 shows that sharing rate decreases with fleet size, since more requests tend to be matched with “free” shuttles instead of being pooled: larger fleet sizes lead to less shareability. This effect becomes particularly evident at higher MDR values. Moreover, the capacity generally has a positive impact on

sharing rate. In some cases, though, this is the opposite case, this can be due to the lower service rate at low capacities, leading to a relative higher sharing rate.

Furthermore, both the MWT and MDR parameters contribute positively to the sharing rate, as shareability increases with higher MWT and MDR values. These findings are consistent with previous research, such as Bilali et al. (2019b) and Santi et al. (2014).

The highest observed sharing rate (0.69) is reached with a fleet size of 1, shuttle capacity of 3, maximum waiting time of 10 min and maximum detour ratio of 2.

5.2.3. Impact on Mean Waiting Time

The KPI “mean waiting time” is defined as total waiting time of all served requests (in seconds), divided by the number of served requests. It is an important indicator for passengers, as longer waiting times are generally undesirable and may discourage the use of the service.

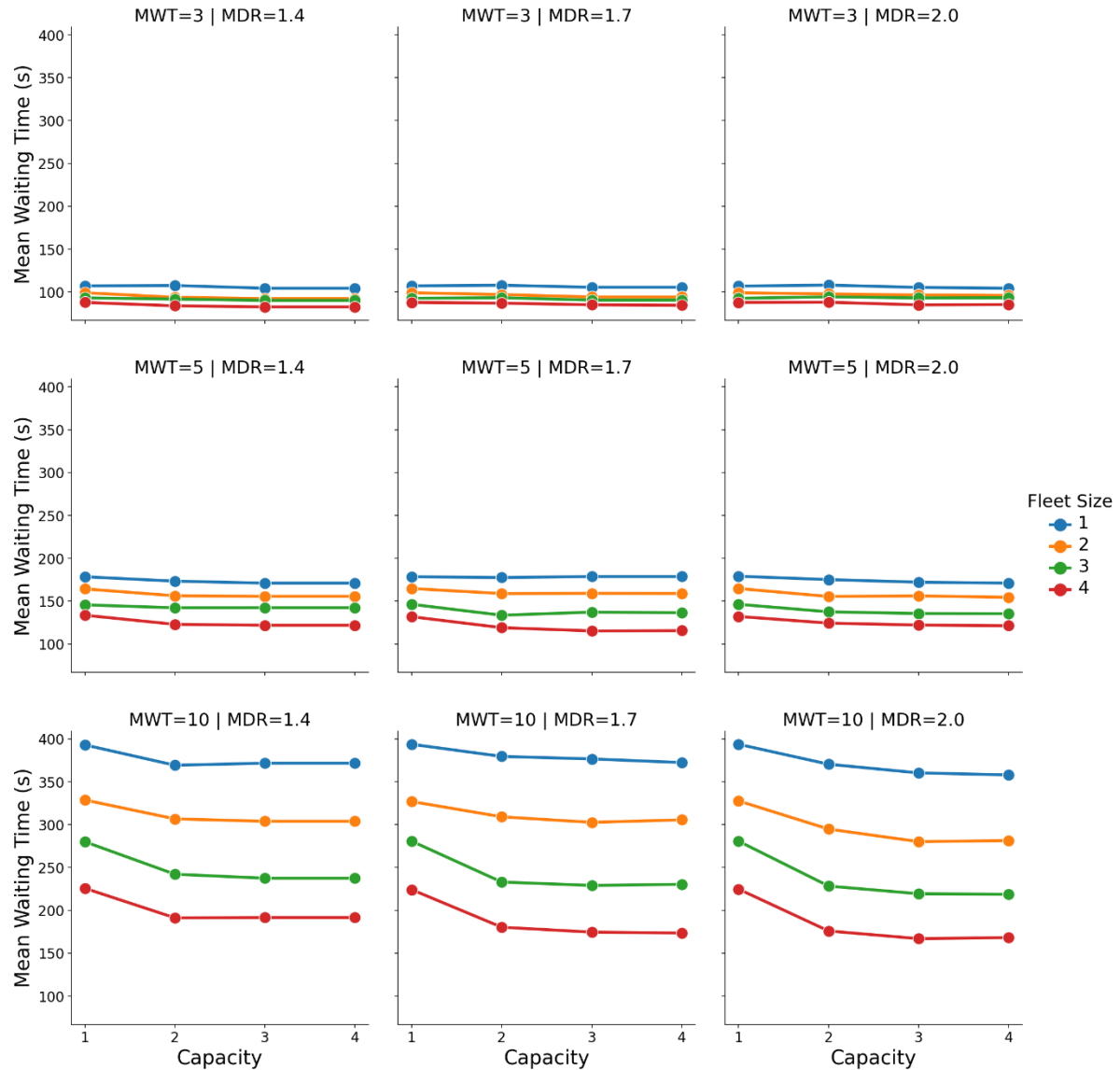


Figure 5.3: Impact of operational parameters on mean waiting time.

Figure 5.3 shows that mean waiting time decreases with increasing fleet size. For example, in the scenario with Capacity = 4, MWT = 10, and MDR = 2, the mean waiting time drops steadily from 358

seconds (fleet size of 1) to 168 seconds (fleet size of 4). Increasing the capacity also leads to lower waiting times, with the most significant improvement observed when increasing capacity from 1 to 2. The influence of capacity becomes more pronounced in scenarios where maximum waiting time (MWT) is set to 10 minutes.

As expected, a higher MWT value results in higher mean waiting times, as longer waiting times of passengers are allowed. In contrast, the MDR parameter has minimal effect on this KPI.

5.2.4. Impact on Mean Detour Time

The KPI “mean detour time” is defined as total detour time of all served requests, divided by the number of served requests. Detour times arise only when there is deviation from the direct trip time, typically as a result of pooling. Accordingly, detour time is 0 in the case of exclusive trips, i.e., when the shuttle capacity is equal to 1.

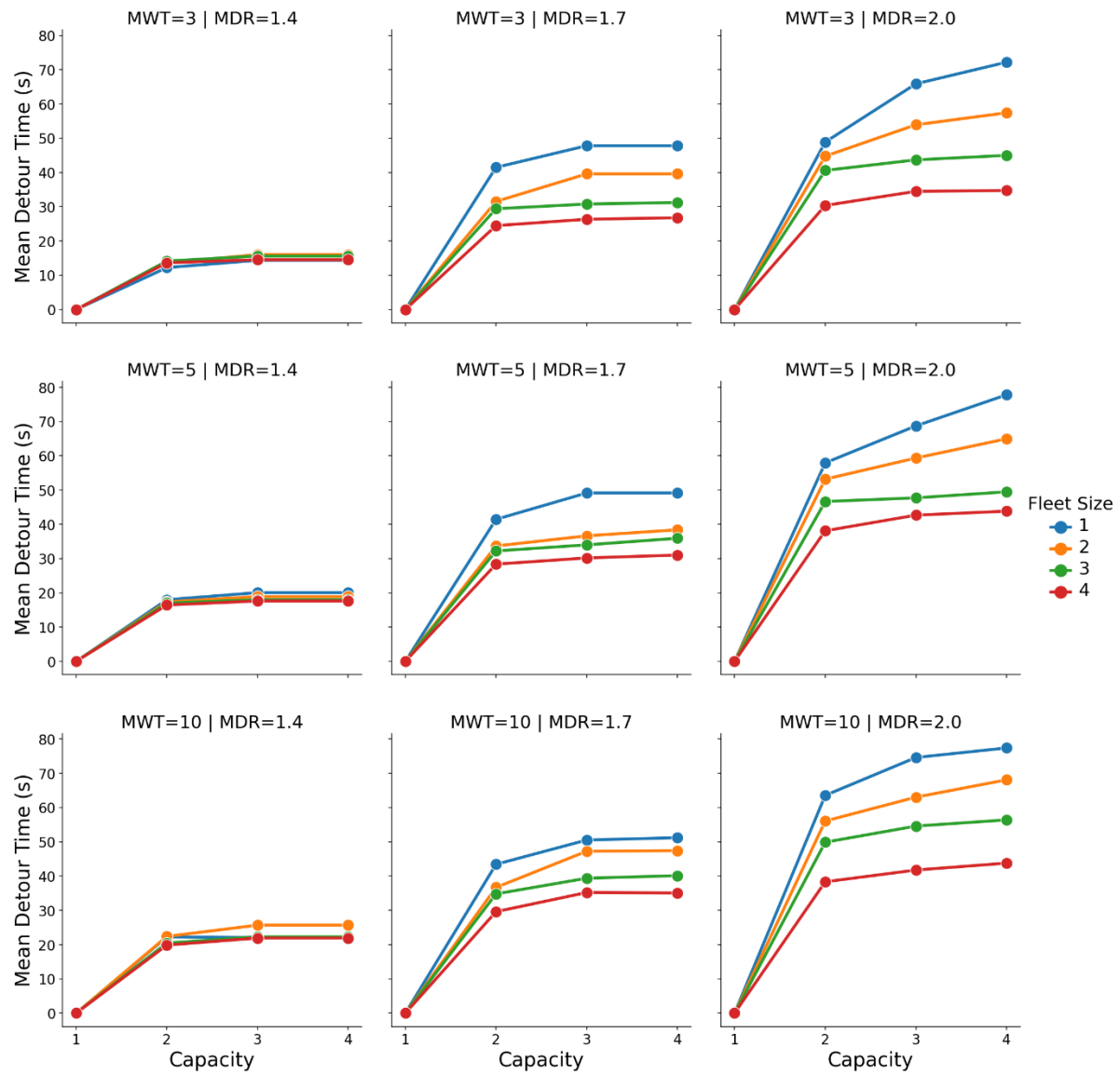


Figure 5.4: Impact of operational parameters on mean detour time.

As illustrated in Figure 5.4, mean detour time drops with increasing fleet size, as more shuttles imply less shareability, less pooling and hence less detours. Conversely, increasing shuttle capacity leads to

higher mean detour times, as greater capacity allows for more pooling and shareability. It can also be observed that mean detour time moderately increases with maximum waiting time (MWT). This can be due to the fact that higher MWT leads to higher shareability, hence higher detour values. Lastly, as expected, the MDR parameter has a direct positive effect on the mean detour time, higher MDR values permit longer detours, and the simulation results reflect this accordingly.

5.2.5. Impact on Mean Mileage

The KPI “mean mileage” is defined as the total distance traveled by all shuttles in the system (in kilometers) divided by the number of shuttles. Unlike the previous KPIs, mean mileage primarily reflects the operator’s perspective in the on-demand system.

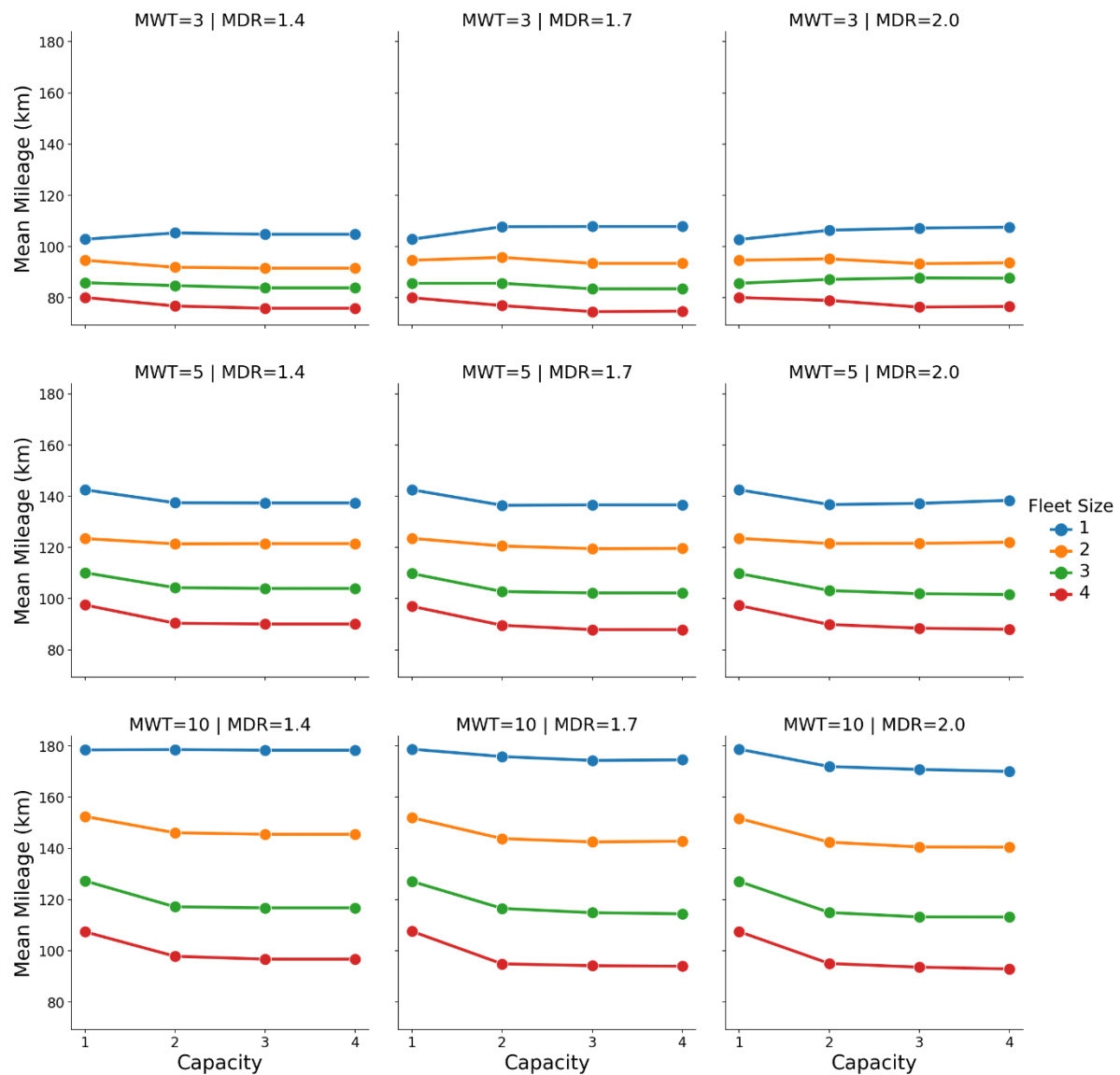


Figure 5.5: Impact of operational parameters on mean mileage.

Figure 5.5 depicts that mean mileage decreases with increasing fleet size. While a larger fleet can serve more requests and thus generate more total mileage, the average mileage per vehicle tends to decrease, suggesting greater operational efficiency. However, it is important to note that increasing the fleet size also raises fixed operational costs. Furthermore, shuttle capacity tends to reduce the mean mileage,

although the effect is not always strong. In some cases, even a slight increase in mileage is observed, likely due to the higher service rate, which in turn lead to higher mileages.

It can also be observed that MWT parameters make an increasing effect on mean mileage: Higher MWT facilitates higher service rate, and consequently, increased mileage. In contrast, variations in MDR have minimal impact: As discussed in Section 5.2.1, service rate remains stable across different MDR values, leading to a nearly unchanged mean mileage.

In the best-performing scenario with a fleet size of 4, shuttle capacity of 4, MWT of 10 minutes, and MDR of 2, a mean mileage of 93 km per shuttle is observed, which corresponds to a total mileage of 372 km in the entire system.

5.2.6. Impact on Mean Occupancy

The KPI “mean occupancy” is defined as the total passenger-kilometers divided by the total mileage. This metric reflects the average number of passengers in the shuttles and can also take values below 1, e.g., in case of empty trips with no passengers on board.

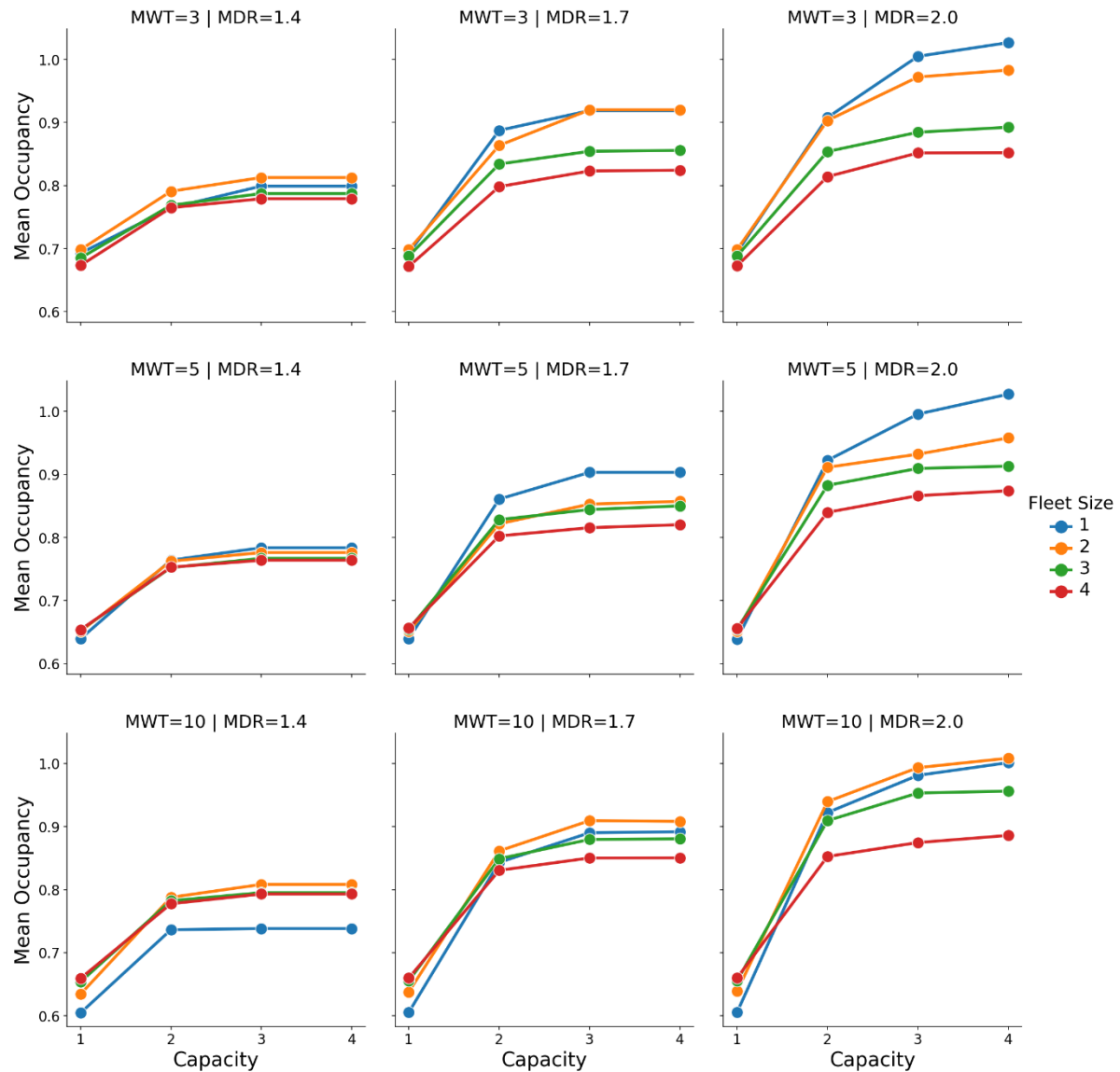


Figure 5.6: Impact of operational parameters on mean occupancy.

Figure 5.6 demonstrates that impact on mean occupancy shows similar trends to the impact on sharing rate (see Figure 5.2). Specifically, mean occupancy decreases with increasing fleet size, as a larger fleet results in less shareability. Conversely, increases in shuttle capacity and MDR value lead to higher mean occupancy, since there is then a higher chance of pooling. Regarding MWT value, no consistent trend can be identified. The results exhibit both increasing and decreasing effects across different scenarios.

5.2.7. Impact on Mileage Index

The KPI “mileage index” is defined as the ratio between the total direct trip distances of all served passengers and the total mileage driven by the fleet. This metric reflects how effectively the system utilizes vehicle kilometers to fulfill travel demand. For example, if a shuttle serves two requests with direct trip distances of 7 km and 8 km, but covers only 10 km due to pooling, this will indicate a mileage index of 1.5.

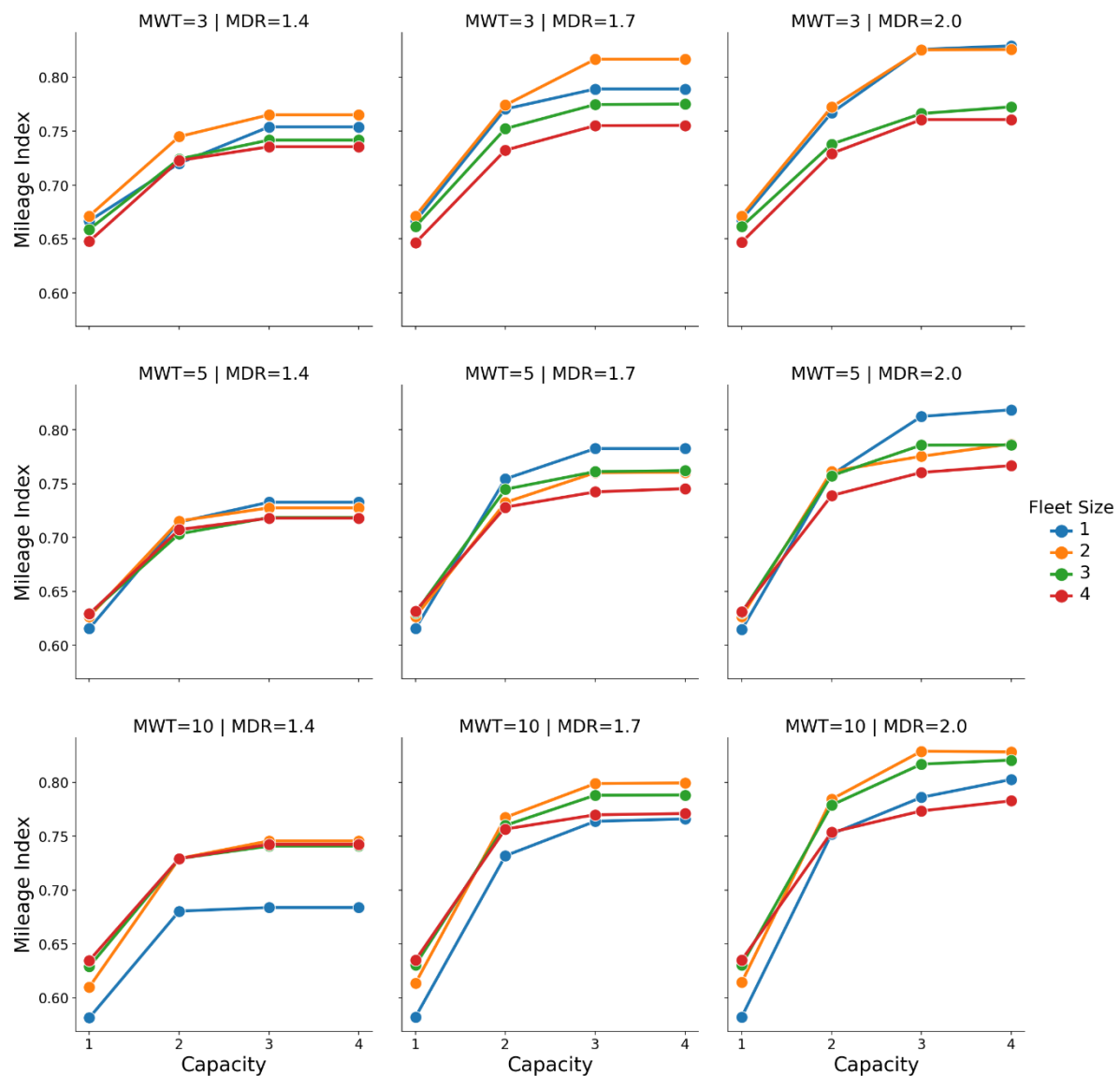


Figure 5.7: Impact of operational parameters on mileage index.

As shown in Figure 5.7, mileage index generally declines as fleet size increases. However, for high MWT values, a fleet size of 1 performs worse in terms of mileage index than slightly larger fleets. This may be attributed to insufficient vehicle availability, causing longer and less effective routes.

Shuttle capacity and maximum detour ratio (MDR) both have a positive impact on mileage index, enabling more pooling. By contrast, maximum waiting time (MWT) does not demonstrate a consistent or significant influence across scenarios.

In all scenarios, mileage index remains below 1, indicating that the fleet covers more kilometers than the cumulative direct trip distances of the served requests. This outcome is primarily related to the request density of the system: higher mileage index would require a denser request pattern, allowing more frequent pooling. In the current scenarios, periods of low demand limit such opportunities, which lead to less (or no) pooling and hence reduces the mileage index. In addition, increasing the weighting parameter for the operator's perspective in the cost function (denoted as W_o in Equation (3.4)) can improve mileage index as well, as this KPI mainly reflects operator's point of view (see Section 5.3.3).

5.3. Results of Batch Assignment

This section presents the results of the simulation experiments performed using the batch assignment. The underlying principles of batch assignment methodology were discussed in detail in Section 3.3. For the analysis in this section, the batch period was set to 2 minutes.

Moreover, the travel demand was increased compared to the previous analysis, in order to model a more complex operation. While the earlier section considered the demand from a single day, the one with the highest number of MORE Sharing trips (see Section 4.3), this section is based on the aggregated demand from three days. The resulting demand used in the simulation experiments in this section is shown in Figure 5.8.

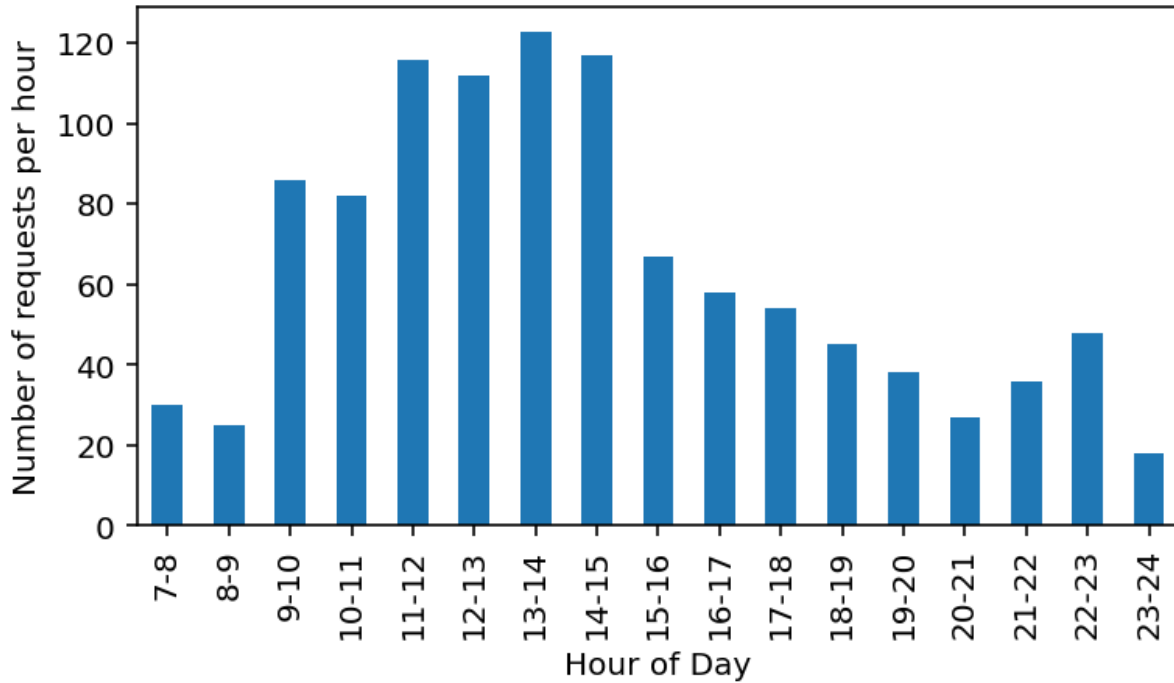


Figure 5.8: Demand profile of batch assignment scenarios over the day.

From the MORE Sharing database, the three days with the highest number of trip requests were selected. Requests from these days were aggregated across the time of day (regardless of the calendar date) to better capture a representative daily demand distribution on campus. Since the travel demand before 7 a.m. was almost zero, only trips after 7 a.m. until midnight were considered. This resulted in a 17-hour

simulation period (7:00–24:00) containing a total of 1,082 trip requests. As depicted in Figure 5.8, the temporal distribution of the demand over the day shows that the request volumes peak around midday, particularly between 11:00 and 15:00.

The next section analyzes the evolution of key performance indicators (KPIs) over the full simulation horizon under a base scenario. Subsequent sections explore the resulting system KPIs under various scenarios.

5.3.1. Base Scenario

In the base scenario, the following operational parameters were applied: fleet size of 6, shuttle capacity of 4, maximum waiting time (MWT) of 10 minutes, and maximum detour ratio (MDR) of 1.7. It is important to note that the travel demand in this scenario is approximately tripled, as a 3-day aggregated demand is used instead of a single-day demand. Accordingly, the fleet size is scaled up to reflect this increase.

Figure 5.9 shows the temporal evolution of the KPIs “service rate” and “sharing rate” throughout the simulation, with values aggregated in 15-minute intervals. The service rate is equal to 1 at the beginning of the simulation, indicating that all incoming requests are successfully served by the shuttles. With the increase in demand, the service rate drops below 0.6, and fluctuates during the high demand period until approximately 15:30. After this time, the service rate stabilizes again at 1, suggesting that all requests are once more being fulfilled during lower demand periods.

Regarding the sharing rate, no pooling occurs during the initial phase of the simulation, resulting in a value of 0. Beginning at 07:45, the first pooled trips appear, with the sharing rate increasing to 0.45, and subsequently to 0.72. Although a temporary decline is observed afterward, the onset of the high demand period causes the sharing rate to climb again, reaching values above 0.9, indicating that over 90% of the requests are pooled, while fewer than 10% are conducted as exclusive trips. After the end of the high demand period, the sharing rate tends to decrease again, with another rise after 21:00.

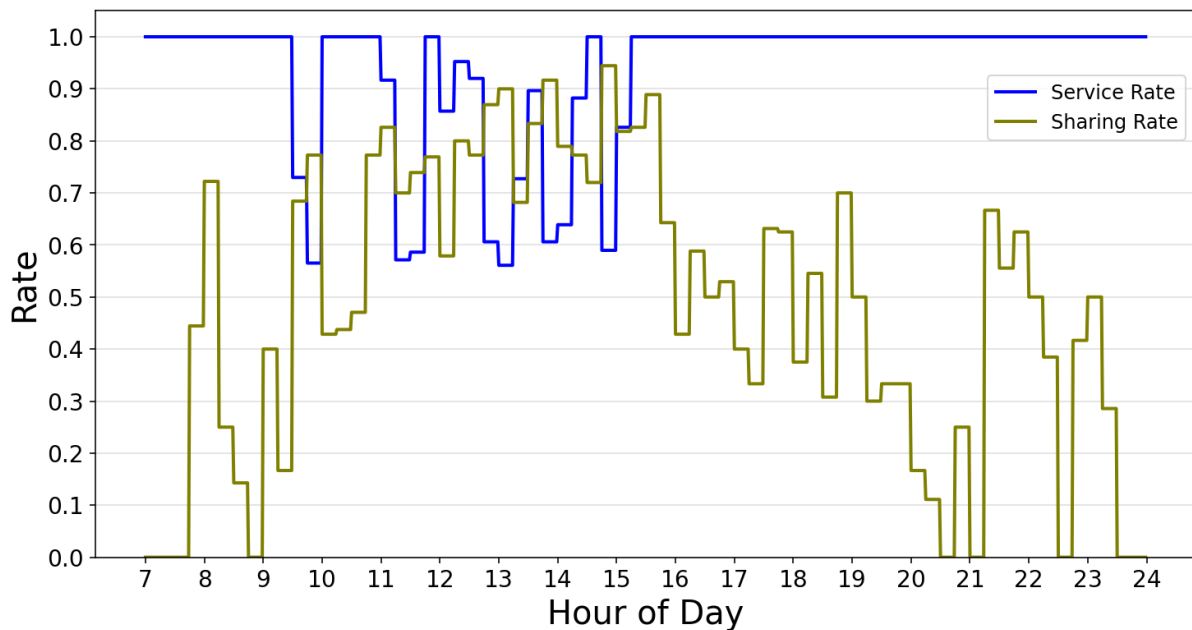


Figure 5.9: Time-series of service rate and sharing rate in a simulation day (in 15-minutes aggregation).

Figure 5.10 demonstrates the cumulative vehicle mileage of the on-demand shuttles over the course of the simulation day. Here, three distinct metrics are presented: total mileage defined as the total distance traveled by the shuttle fleet, occupied mileage defined as the distance traveled with at least 1 passenger on board, and empty mileage as the distance traveled with no passengers on board, for instance empty trips to pick up a passenger, or trips from/to depot points.

The cumulative curves reveal some insights about how the fleet is utilized throughout the day. Firstly, total mileage rises steadily from 0 at 07:00 to 770 km at midnight, while the increase is steeper between 09:00-15:00, where travel demand is higher. Secondly, occupied mileage follows a similar shape and increases more rapidly during the high demand period. It reaches around 560 km at the end of the simulation, which corresponds roughly to the 75% of the total mileage. Finally, the empty mileage climbs to 210 km in a more constant fashion than the other two metrics, accounting for 25% of the total mileage.

To reduce empty mileage, several strategies could be considered. One approach involves implementing intelligent rebalancing policies, particularly during low-demand periods. Additionally, finding techniques in the simulation to allow multiple shuttles to stay in the same stop can have an impact; as in the current simulation, an idle shuttle is sent to depot point if it blocks another shuttle, which in turn leads to empty trips. Furthermore, increasing the value of the weighting parameter for the operator's perspective in the cost function (denoted as W_o in Equation (3.5)) can also help reduce the empty mileage, as this metric primarily reflects operator's point of view (see Section 5.3.3).

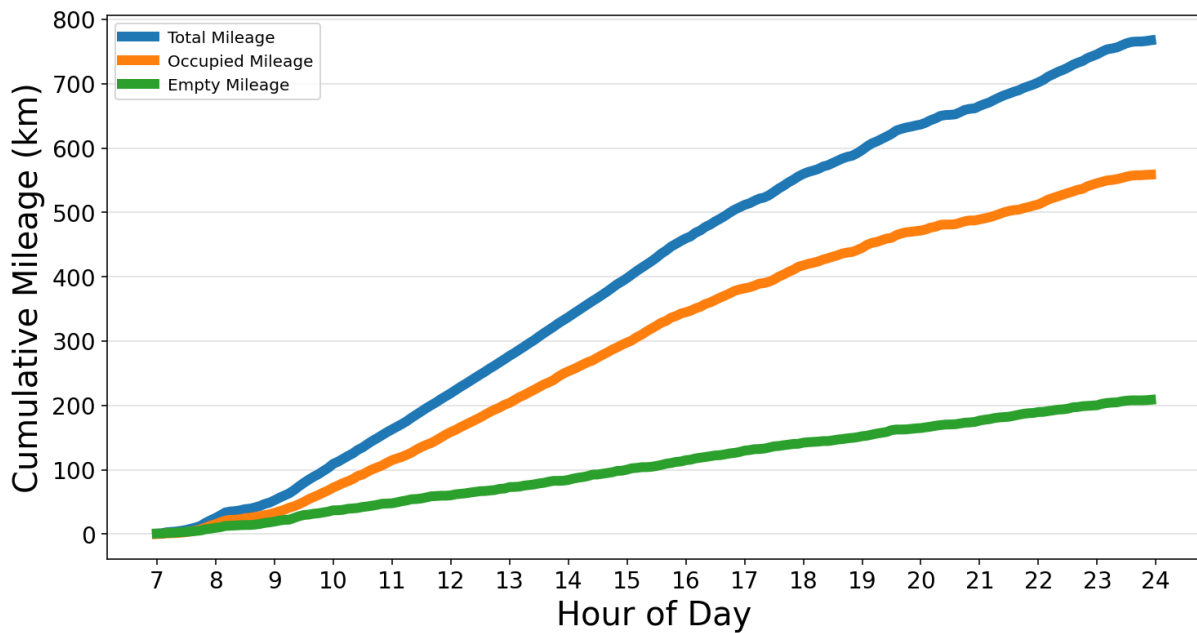


Figure 5.10: Distance traveled by the shuttles in the entire simulation (cumulative).

Figure 5.11 presents the variation in mean waiting time for served requests, aggregated in 15-minute intervals throughout the simulation day. It can be observed that the mean waiting time hovers around 200 seconds on average at the beginning of the simulation, with a single jump to 340 seconds around 08:00. As demand increases, the mean waiting time rises steadily, reaching 450 seconds in certain intervals. The peak is observed at 13:15, with a value of 520 seconds. Following this peak, the trend

declines, and values fluctuate around 180 seconds after the end of the high demand period until the end of the simulation. The mean waiting time across the entire simulation is observed to be 295 seconds.

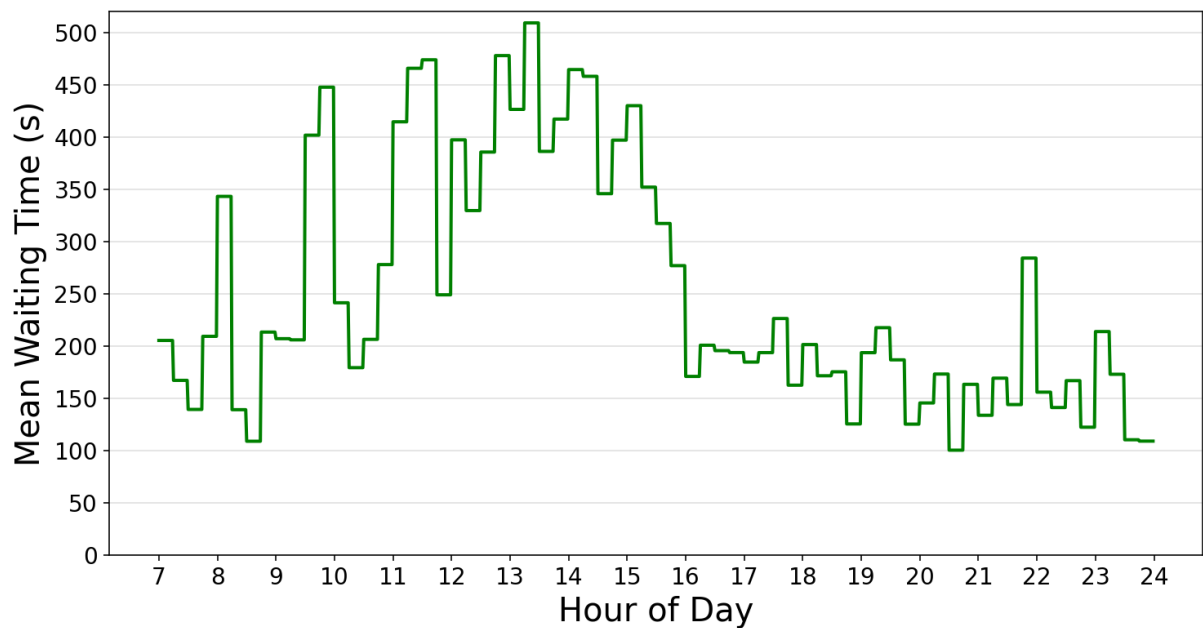


Figure 5.11: Time-series of mean waiting time in a simulation day (in 15-minutes aggregation).

Similarly, Figure 5.12 shows the evolution of the mean detour time of served requests, also aggregated in 15-minute intervals. The mean detour time follows a similar trend to that of mean waiting time; the detours increase with high demand period, reaching a maximum of approximately 120 seconds, and oscillates between 0 and 60 seconds during the low demand period. Intervals with a mean detour time of 0 seconds imply that all requests within these intervals were served via exclusive trips, without any pooling and detour. The mean detour time across the entire simulation is observed to be 61 seconds.

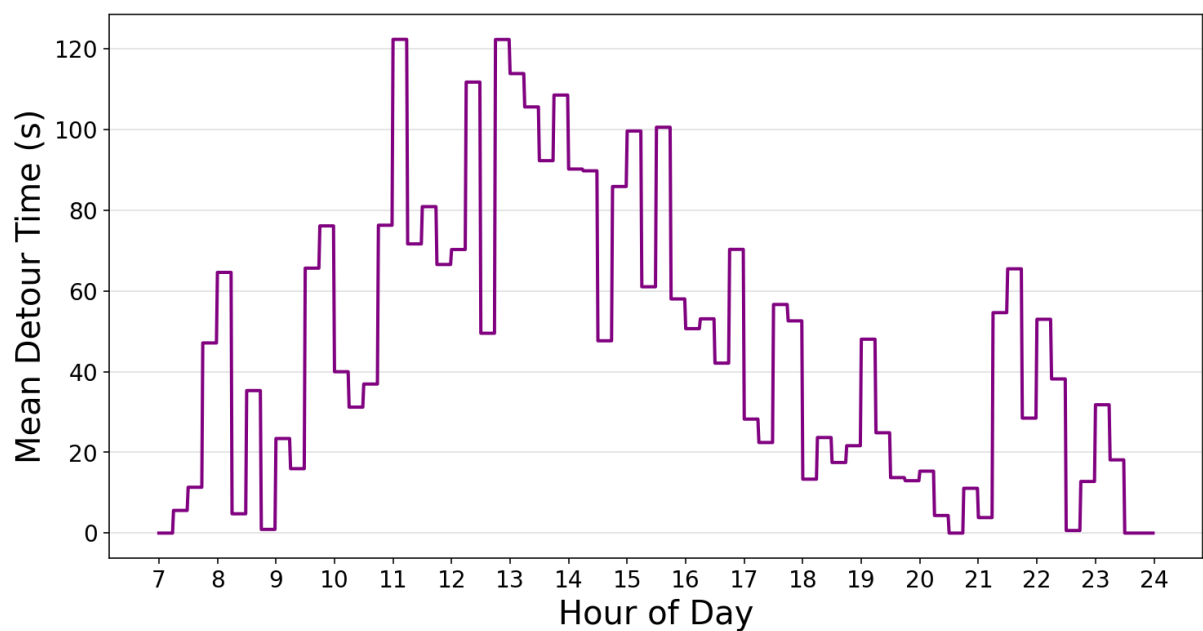


Figure 5.12: Time-series of mean detour time in a simulation day (in 15-minutes aggregation).

To gain a clearer understanding of the distribution of waiting and detour times for served requests during the simulation, the following box plot diagram (Figure 5.13) can be explored. The waiting times range

from 0 to 750 seconds. A waiting time of 0 seconds may occur when the passenger sends a request at the end of the batch period from a stop, where a shuttle is already waiting in idle mode. In such cases, the passenger is picked up immediately, leading to zero waiting time. Although a maximum waiting time (MWT) of 10 minutes is set in the assignment for this scenario, the observed upper bound of 750 seconds can be attributed to operational dynamics. Specifically, the MWT constraint is checked during the assignment process, hence before the pick-up trip itself. But the actual time to pick-up may be prolonged due to interactions of the shuttle with other traffic participants, congestion situations or queuing behind another shuttle already performing a stop event. In general, waiting times are distributed between 150 and 440 seconds, with a median value of 250 seconds.

Regarding detour times, the minimum observed value is 0 seconds, which also corresponds to the 25th percentile, indicating that at least one-quarter of all served requests experienced no detour. These cases typically arise during the low demand period of the simulation. Most detour times fall within the 0 to 100 seconds range, although some outliers up to 390 seconds are also present.

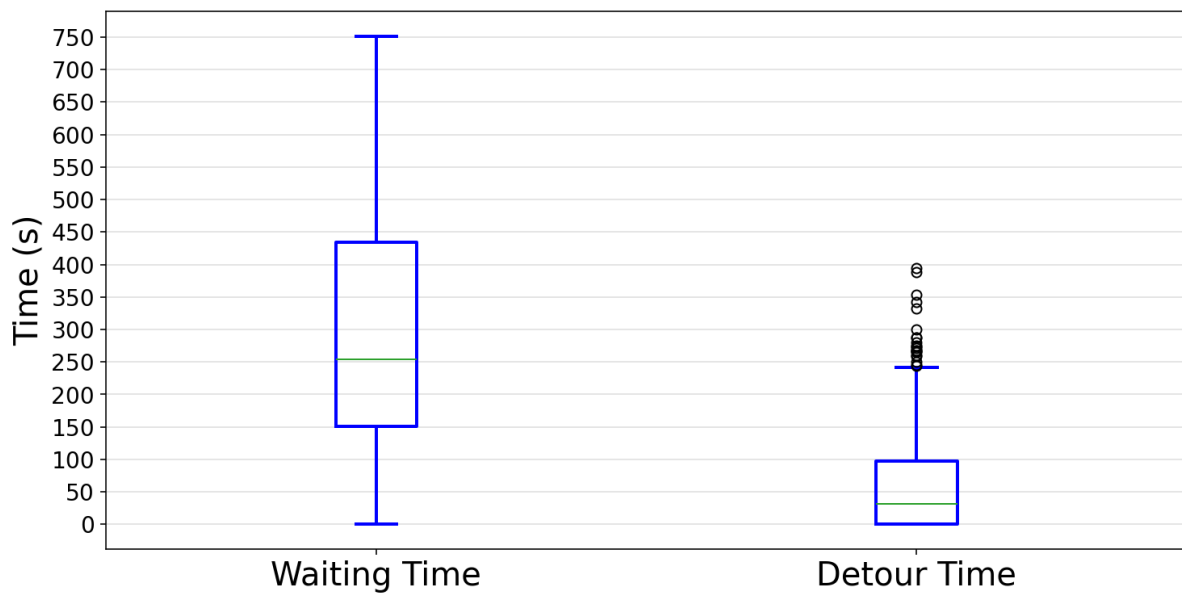


Figure 5.13: Box plot of waiting time and detour time of served requests ($n = 935$).

Figure 5.14 illustrates the evolution of the mean occupancy of on-demand shuttles, aggregated in 15-minute intervals. As mean occupancy is calculated as the total passenger-kilometers divided by the total mileage, it can also take values below 1, e.g., if empty trips without any passengers are dominant.

The results show a notable increase in occupancy during the high-demand period, where values exceed 1, indicating that multiple requests are pooled. The highest value of 1.7 is reached around 14:00. Following this, a gradual decline is observed, reflecting the low demand period. Interestingly, a secondary rise in occupancy is seen after 21:00, which corresponds with earlier findings on increased sharing rates during this time (see Figure 5.9).

Across the full simulation period, the average mean occupancy is calculated as 1.1, suggesting moderate levels of ride-pooling throughout the day.

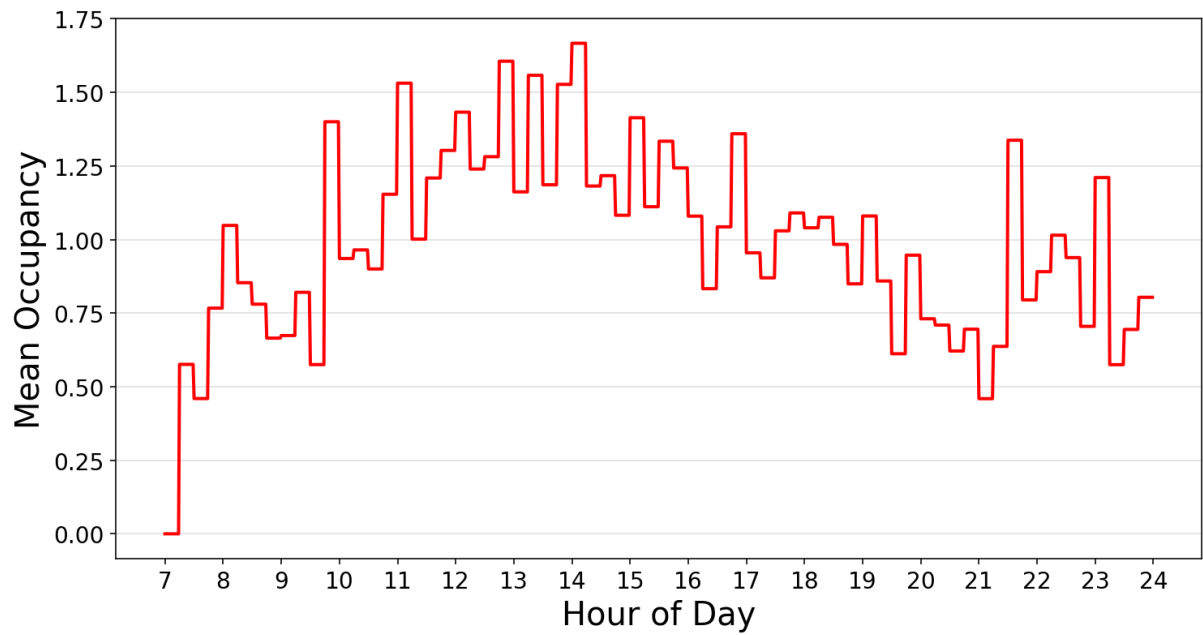


Figure 5.14: Time-series of mean occupancy in a simulation day (in 15-minutes aggregation).

Although Figure 5.14 provides insights about the mean occupancy across all on-demand shuttles, it does not directly indicate the number of passengers in individual shuttles at any given time. For this analysis, Figure 5.15 can be utilized.

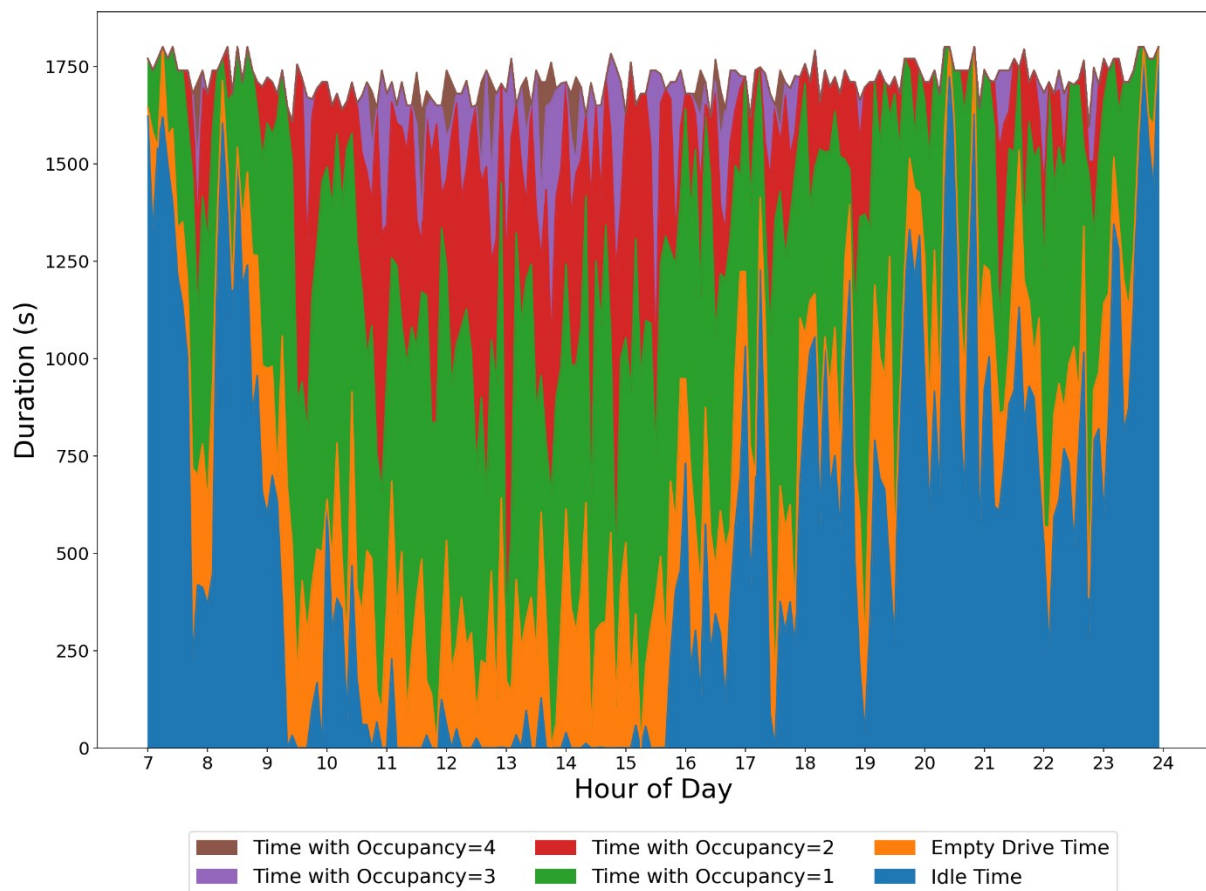


Figure 5.15: Utilization of the on-demand fleet (in 5-minutes aggregation).

The illustration in Figure 5.15 shows the fleet utilization of the on-demand shuttles throughout the day. A shuttle waits in idle mode when no request is currently assigned. It performs an empty drive, for example, when it travels to pick up a passenger without anyone already on board. Otherwise, the shuttle operates with one to four passengers, as four being the maximum capacity defined in the base scenario.

For this analysis, the operational status of each shuttle is recorded in 5-minute intervals and aggregated across the fleet. The results show that during low demand periods, particularly in the early morning and late evening hours, shuttles predominantly remain idle. In contrast, during high demand periods, especially between 11:00 and 15:00, idle times are significantly reduced, while the amount of empty driving increases.

Throughout the day, shuttles most frequently operate with one passenger on board, and two-passenger occupancy is also common during peak periods. Higher occupancies occur less frequently; three-passenger occupancy is observed in the peak times and occasionally in other periods, while four-passenger occupancy is observed only rarely, mostly confined to the high demand period.

5.3.2. Impact of Travel Demand

In this section, the impact of travel demand of the on-demand system on system KPIs is elaborated.

Firstly, the operational parameters are taken as in the base scenario: fleet size of 6, shuttle capacity of 4, maximum waiting time (MWT) of 10 minutes, and maximum detour ratio (MDR) of 1.7. Then, three different scenarios with varying demand are generated. The first scenario is based on the request pool of a single day, the day with the highest demand in MORE Sharing, comprising 385 requests. The second scenario is based on a 2-day request pool. Here, the two days with the highest number of MORE Sharing trip requests are selected, and requests from these days are aggregated across the time of day, using the same methodology as explained at the beginning of Section 5.3. This comprises a total of 745 requests. Finally, the last scenario is based on the request pool of 3 days, containing 1,082 requests.

Keeping all other parameters constant, Figure 5.16 shows the impact of travel demand on service rate and sharing rate. The service rate has a value of 1 in the 1-day demand scenario, suggesting that all requests can be successfully served by the fleet, and no request rejections occur. However, considering a 2-day demand affects this KPI negatively, and the service rate slightly reduces to 0.97. A 3-day demand has a much greater effect: 13% of the requests are rejected, corresponding to a service rate of 0.87. This indicates that the fleet cannot handle all requests under high demand scenarios.

Regarding the sharing rate, the percentage of the shared trips increases with increasing demand, since a larger request pool has a positive effect on shareability, allowing more requests to be pooled together. In the 1-day demand scenario, the sharing rate is 0.35, meaning almost two-thirds of the requests are served exclusively, in ride-hailing mode. This value rises to 0.53 for 2-day demand and to 0.63 for 3-day demand scenario. It is important to note that nearly two-thirds of the requests are now pooled, so the situation in the low demand scenario has effectively reversed.

Figure 5.17 exhibits the impact of travel demand on mean waiting time and mean detour time. It can be seen from the diagram that both KPIs are negatively affected as demand increases. The mean waiting time is 171 seconds in the low demand scenario, increasing to 244 seconds under moderate demand, and rising further to 294 seconds in the high demand scenario. This represents a 70% increase from low to high demand.

Mean detour time shows a similar trend, though the percentage increase is even more significant. In the low demand scenario, the mean detour time is 25 seconds, rising to 47 seconds in the moderate demand

case, and reaching 61 seconds under high demand. This corresponds to a 135% increase from low to high demand.

It can therefore be concluded that increasing travel demand not only reduces the service rate, but also increases the mean waiting and detour times for served requests. One possible way to mitigate these effects could be to increase the fleet size accordingly, for example.

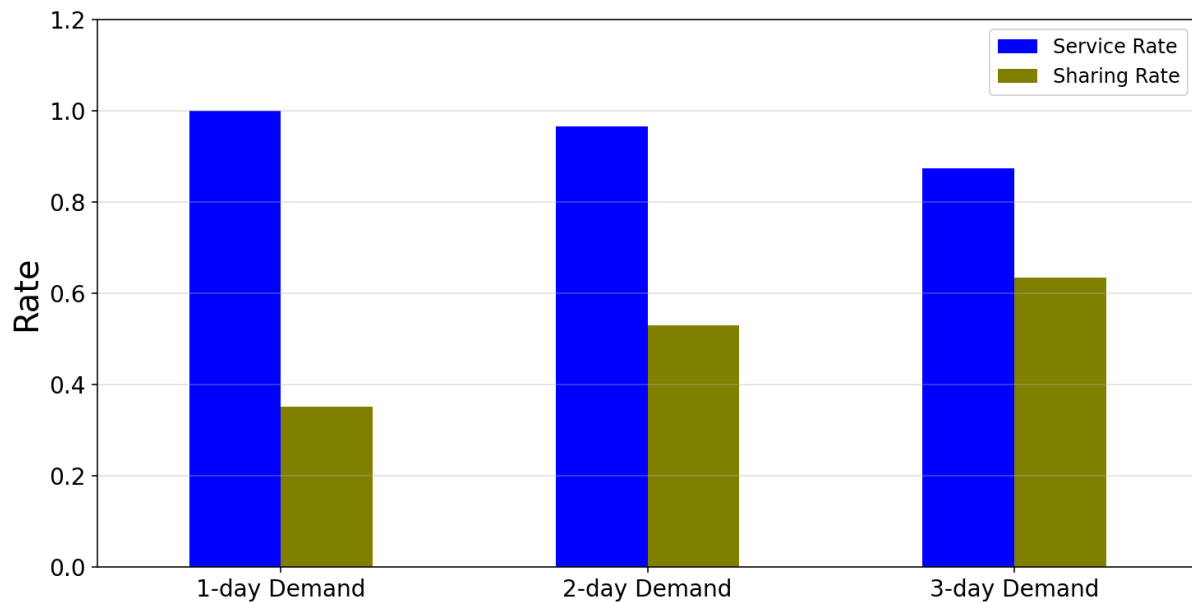


Figure 5.16: Impact of travel demand on service rate and sharing rate.

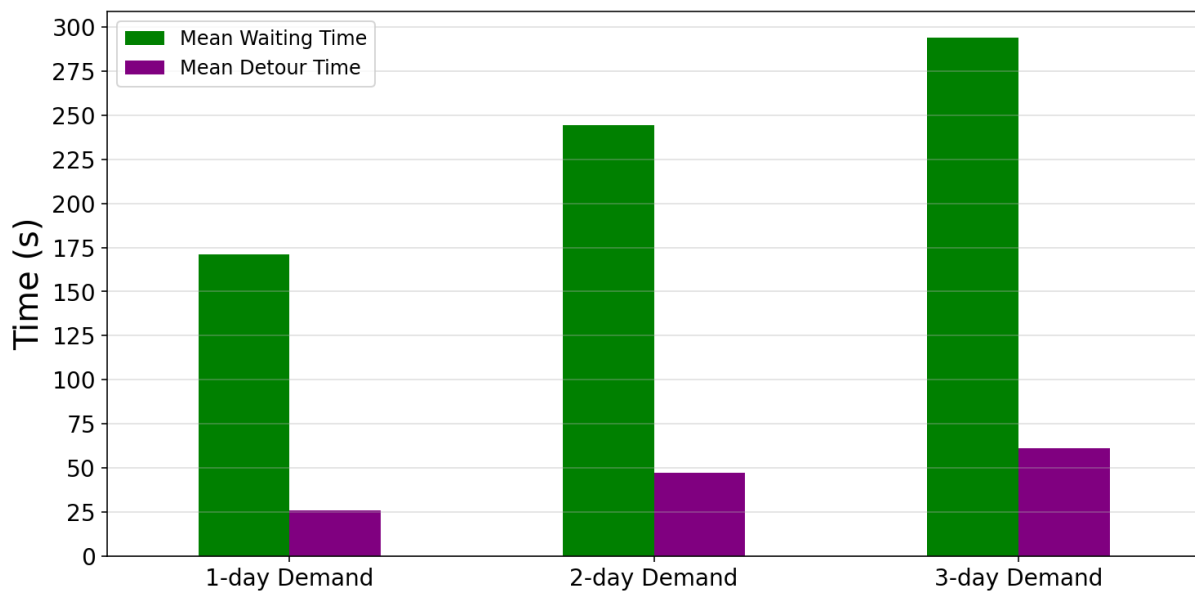


Figure 5.17: Impact of travel demand on mean waiting time and mean detour time.

In Figure 5.18, system KPIs that reflect more the operator's perspective are illustrated. These include mean occupancy, mileage index, and empty mileage ratio. Here, increasing travel demand has a positive effect on all three KPIs. For example, the mean occupancy of the shuttles increases from 0.82 to 0.97, and then to 1.08 across the rising demand scenarios. This is consistent with the findings related to the increase in sharing rate under different demand patterns. The mean occupancy values below 1 are caused by empty trips with no passengers on board (see Section 5.2.6).

Furthermore, there is a gradual increase in mileage index from the lowest to the highest demand scenario. This outcome is expected, as higher demand allows for greater shareability and more pooling opportunities.

Lastly, the figure shows that the empty mileage ratio, calculated as the empty mileage of the fleet divided by the fleet's total mileage, decreases slightly as demand increases. This KPI drops from 0.32 in the low demand scenario, to 0.29 in moderate demand, and further to 0.27 under high demand.

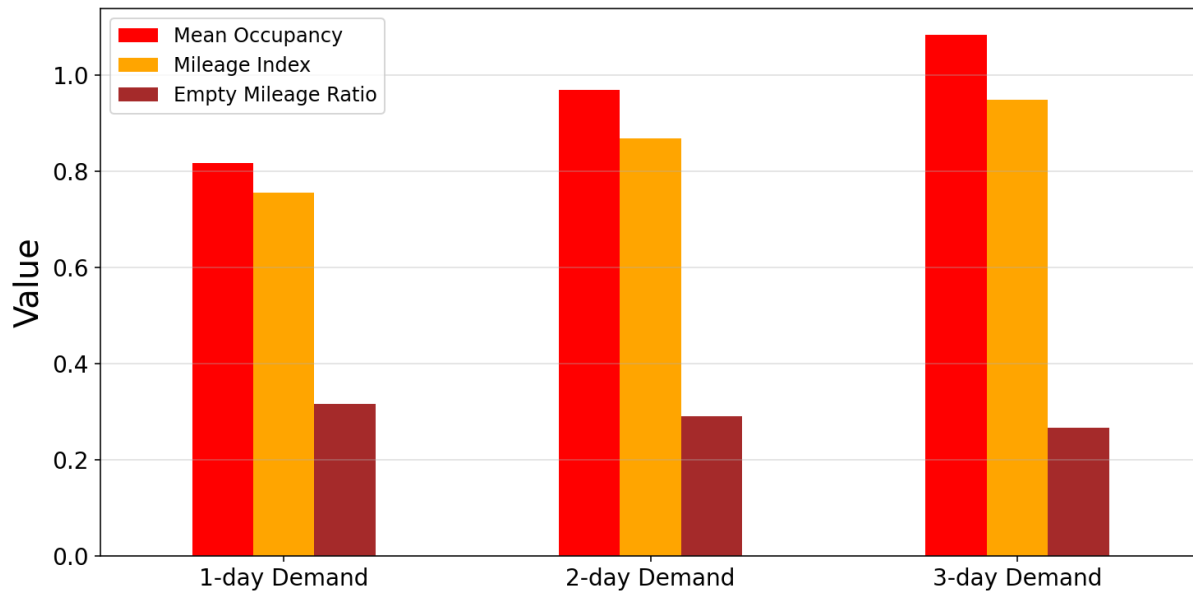


Figure 5.18: Impact of travel demand on mean occupancy, mileage index, and empty mileage ratio.

Similar to Figure 5.18, the Figure 5.19 also focuses on the operator-oriented KPIs. The first metric examined is total mileage. This value increases from 390 km in the 1-day demand scenario to 637 km in the 2-day scenario, and reaches 770 km in the 3-day scenario. An increase in total mileage is observed with increasing demand, however the increase is not proportional to the size of the request pool, it grows at a slower rate. This can be attributed to two main factors:

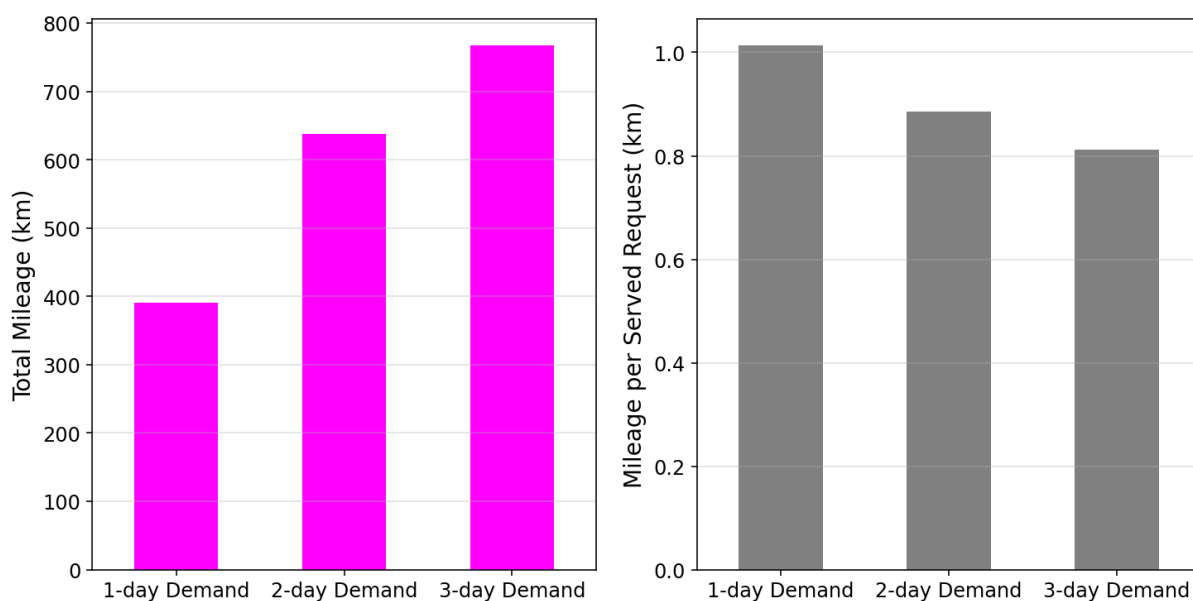


Figure 5.19: Impact of travel demand on total mileage and mileage per served request.

Firstly, as demand rises, the service rate declines, resulting in a smaller proportion of requests being served, which in turn slows down the increase in total mileage. Secondly, with increasing demand, more shareability and pooling is achieved, therefore the increase in total mileage becomes proportionally smaller.

The second KPI shown in Figure 5.19 is “mileage per served request”, which is calculated as the total mileage divided by the total number of served requests. This KPI is primarily useful to neutralize the effect of the service rate. With this analysis, it becomes clearer that a better shareability and pooling is provided with higher demand patterns: mileage per served request decreases steadily from 1-day to the 3-day demand scenario.

5.3.3. Impact of Weighting Parameters in the Cost Function

This section analyzes the impact of the weighting parameters (W_o and W_p) in the cost function (see Equation (3.5)) on certain KPIs. These parameters are used to place more emphasis on either the operator or passenger side. This allows for flexibility, enabling either side to be prioritized depending on the overall system strategy. In the default setting, both weightings are set equally to reflect a balanced approach.

To examine the impact of different weighting schemes on system KPIs, simulations with five weighting pairs are conducted: (100,0), (75,25), (50,50), (25,75) and (0,100), where the first value shows the weighting on operator side, and the second value the weighting on passenger side. For example, the pair (100,0) indicates a purely operator-focused objective, while (50,50) allocates equal priority to both perspectives.

The operational parameters are identical to those in the base scenario: fleet size of 6, shuttle capacity of 4, maximum waiting time (MWT) of 10 minutes, and maximum detour ratio (MDR) of 1.7.

Figure 5.20 presents the influence of varying weightings on the mean waiting time and mean detour time. From left to right in the diagram, the impact of the operator side diminishes while the impact of the passenger side increases. This shift is reflected in the KPIs: mean waiting time drops from 347 seconds to 308, 294 and 289 seconds, respectively. In the final step, when the weighting pair is changed from (25,75) to (0,100), no further improvement is observed, and the mean waiting time remains constant. The reduction in mean waiting time slows down from left to right of the diagram. Mean detour time follows a similar pattern, dropping from 79 seconds in the purely operator-focused scenario to 68, 61, 54, and finally 49 seconds in the purely passenger-focused scenario.

These results suggest that increasing the emphasis on the passenger perspective in the cost function leads to lower waiting and detour times.

The next analyses examine the system from the operator’s perspective. This time, the effect of the weightings on operator-related KPIs is investigated.

In Figure 5.21, the variation in total mileage and mileage per served request is presented. Total mileage shows a steady increase from left to right in the diagram: from 725 km in the purely operator-focused scenario to 846 km in the purely passenger-focused scenario, corresponding to an increase of about 17%. It is important to note here that the service rate remains nearly constant across all scenarios, thereby excluding any influence of the service rate on this variation. This observation is further supported by the trend of the KPI “mileage per served request”, which closely mirrors the pattern of total mileage.

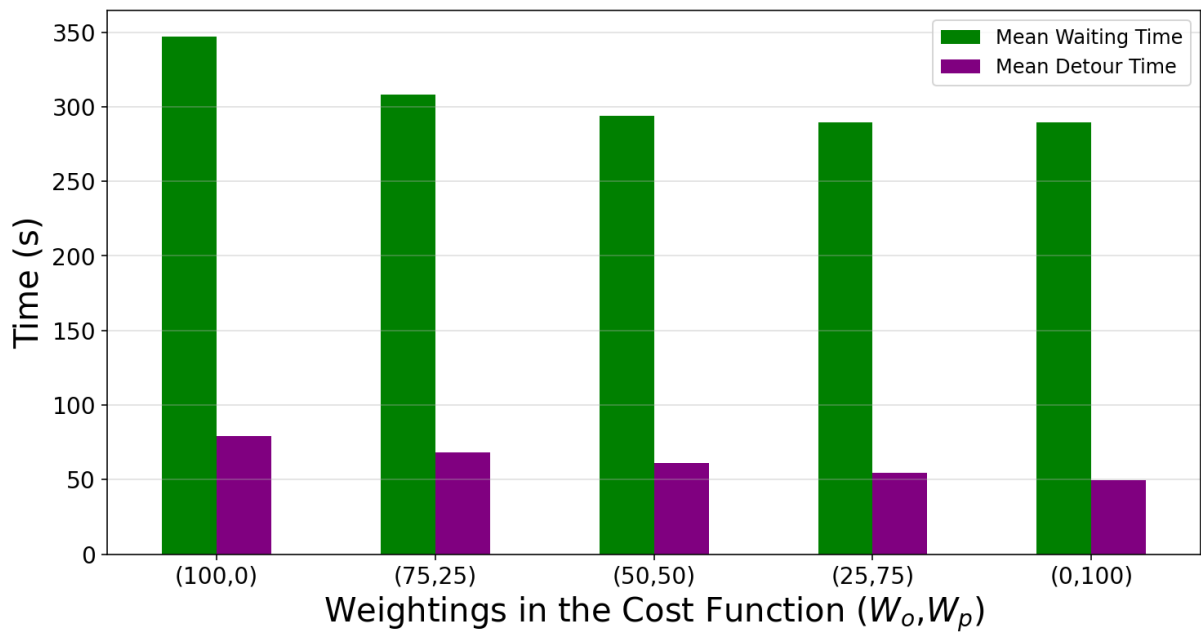


Figure 5.20: Impact of weightings in the cost function on mean waiting time and mean detour time.

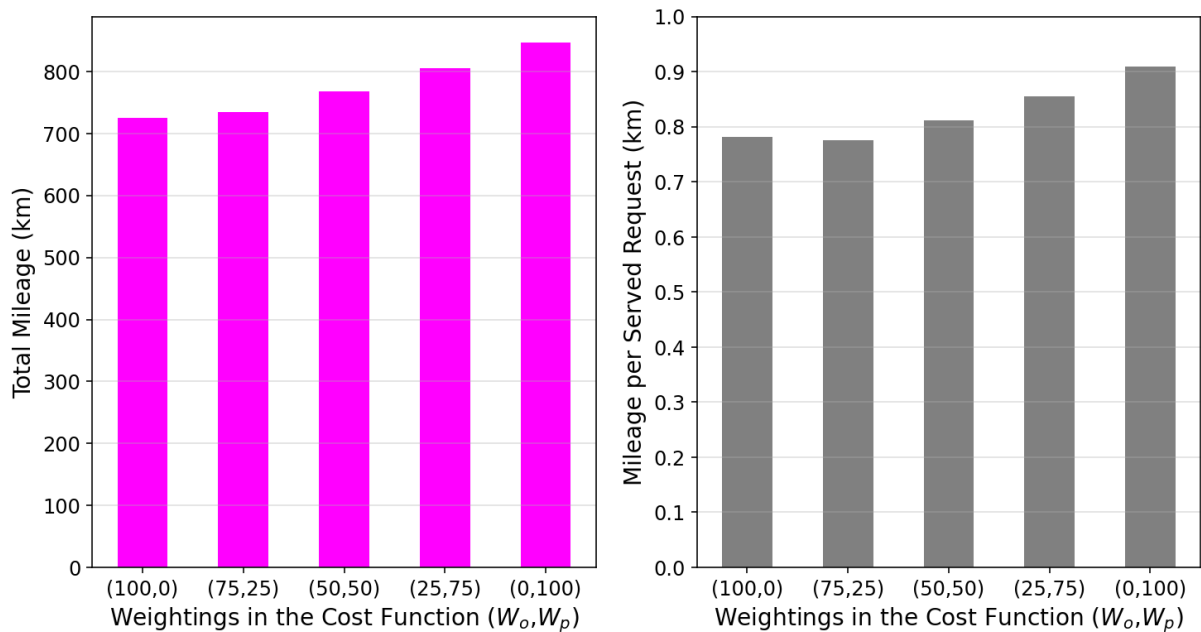


Figure 5.21: Impact of weightings in the cost function on total mileage and mileage per served request.

Figure 5.22 shows that a decreasing operator weighting (from left to right in the diagram) results in a decline in the mileage index, a KPI that reflects how effectively requests are served by the fleet in terms of total distance traveled. As the focus shifts toward the passenger side, achieving distance savings becomes more challenging. In the same manner, the empty mileage ratio experiences a slight increase with operator side less emphasized.

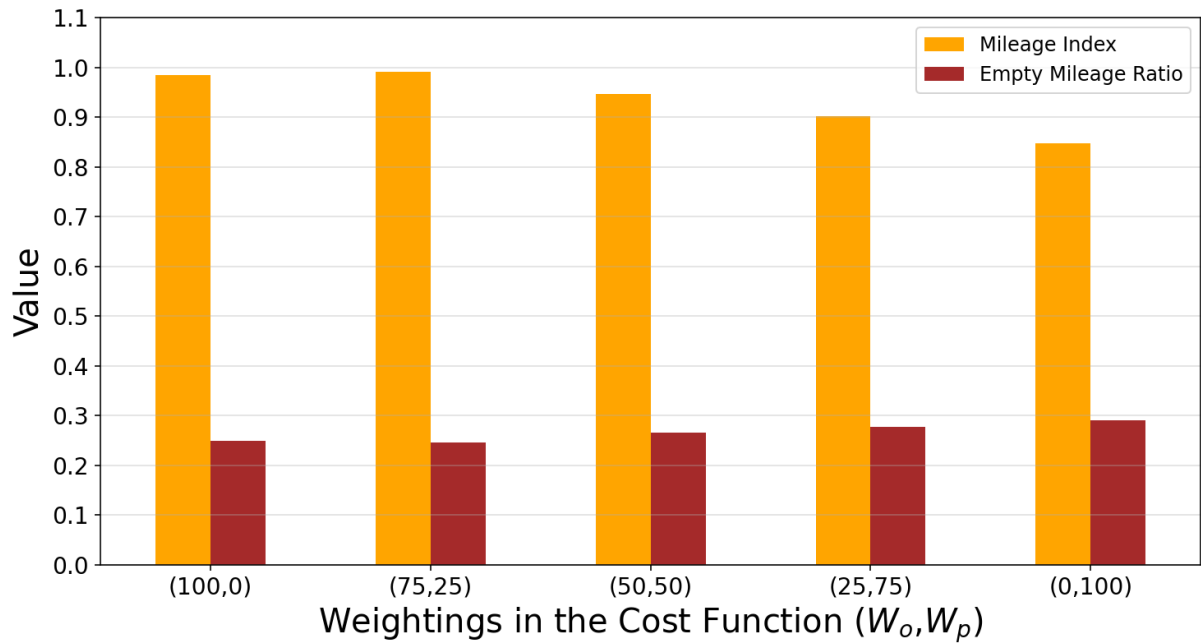


Figure 5.22: Impact of weightings in the cost function on mileage index and empty mileage ratio.

5.3.4. Comparison of Ride-hailing and Ride-pooling

In this section, two well-known operational strategies in on-demand systems, ride-hailing and ride-pooling, are compared. The advantages of ride-pooling over ride-hailing are quantified based on key system KPIs.

Definitions and core characteristics of both operational strategies are provided in Section 2.1.4 (for ride-hailing) and Section 2.1.5 (ride-pooling). In brief, ride-hailing systems offer exclusive services to passengers, meaning that trips are not shared with any other passenger. In contrast, ride-pooling allows trip sharing, where multiple passengers may be pooled together in a single on-demand vehicle.

The operational parameters used in this analysis of the section are identical to those in the base scenario, with one exception. A fleet size of 6, maximum waiting time (MWT) of 10 minutes, and maximum detour ratio (MDR) of 1.7 are maintained. The only difference lies in the shuttle capacity: for the ride-hailing strategy, capacity is set to 1, due to the restriction on trip sharing. In contrast, the ride-pooling strategy retains a shuttle capacity of 4, consistent with the base scenario.

Figure 5.23 illustrates the comparison between ride-hailing and ride-pooling based on seven system KPIs. The service rate reaches a value of 0.75 in ride-hailing, whereas in ride-pooling this increases to 0.87, indicating a 17% improvement. Mean waiting time also benefits from pooling: while ride-hailing results in a mean waiting time of 352 seconds, this value drops to 294 seconds in ride-pooling. This means that ride-pooling not only serves more requests but also reduces waiting times, thus improving passenger-related KPIs and increasing overall user satisfaction.

The total mileage also decreases with ride-pooling, from 897 to 767 km, a 14% reduction. This is especially notable given that the service rate is higher in ride-pooling scenario, which would typically suggest greater mileage. This observation is supported by the next KPI, mileage per served request, which experiences a stronger decrease of 27% (from 1.11 to 0.81 km).

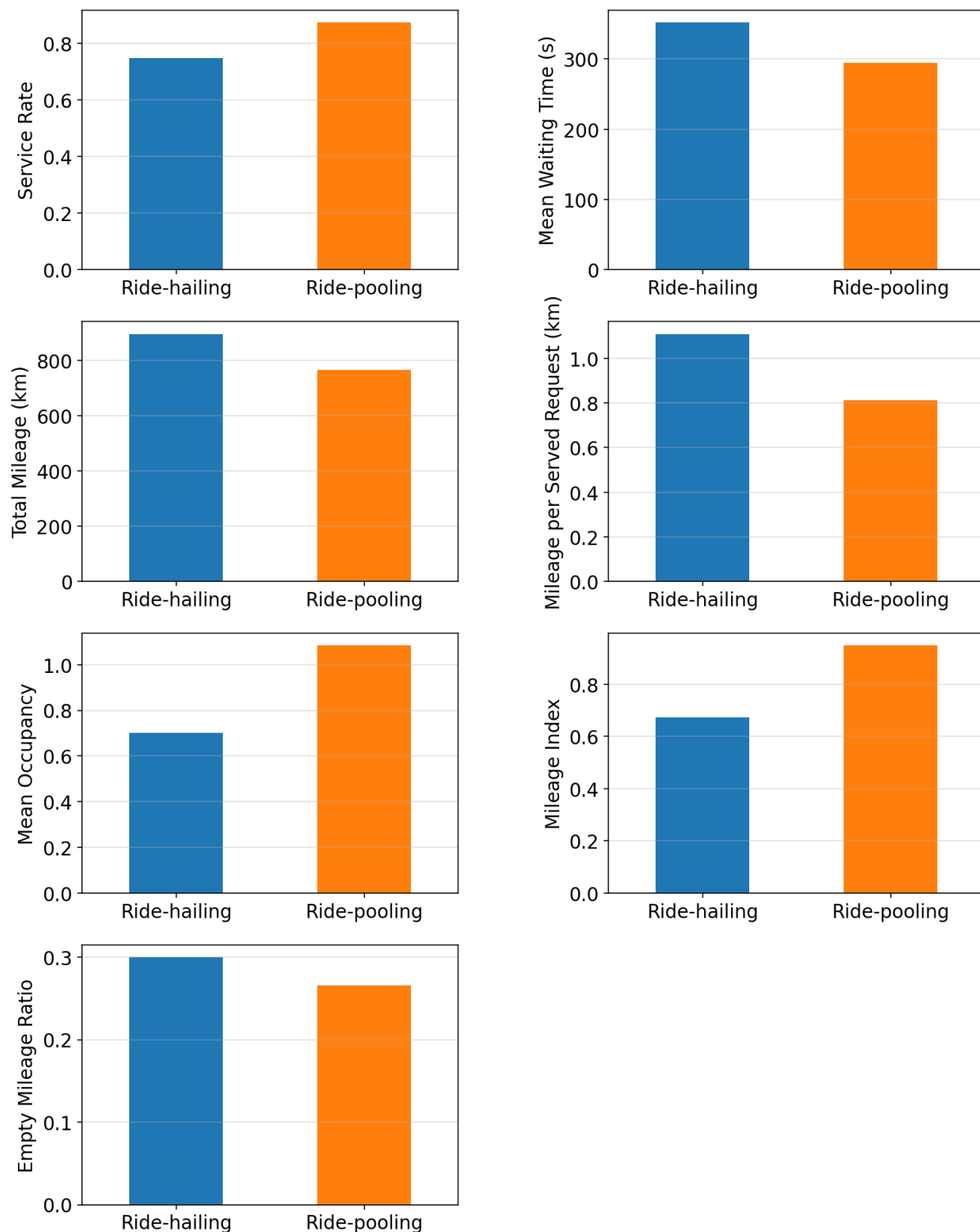


Figure 5.23: Comparison of ride-hailing and ride-pooling based on system KPIs.

Other operator-related KPIs also show improvements. For example, mean occupancy rises from 0.70 to 1.08, mileage index increases from 0.67 to 0.95, and empty mileage ratio decreases slightly from 0.30 to 0.27, when the operational strategy is shifted from ride-hailing to ride-pooling. These findings suggest that the pooling strategy contributes not only to passenger-related KPIs but also to the operator-related KPIs.

A comparison of sharing rate and mean detour time is not applicable here, as ride-hailing does not involve trip sharing or detours, and therefore a comparison would not be meaningful.

For a clearer overview, the relative change in system KPIs from ride-hailing to ride-pooling is summarized in Figure 5.24.

Overall, it can be concluded that allowing trip sharing through ride-pooling substantially improves system performance compared to ride-hailing, where trips are realized exclusively. The benefits are likely to become even more pronounced in scenarios with higher demand. While larger fleets may be required to meet such demand under ride-hailing, this would come with significantly higher fixed operational costs for the operator. On the passenger side, potential trade-offs in ride-pooling are the reduced comfort, as users would need to share the ride with others instead of traveling alone, and longer travel times due to detours.

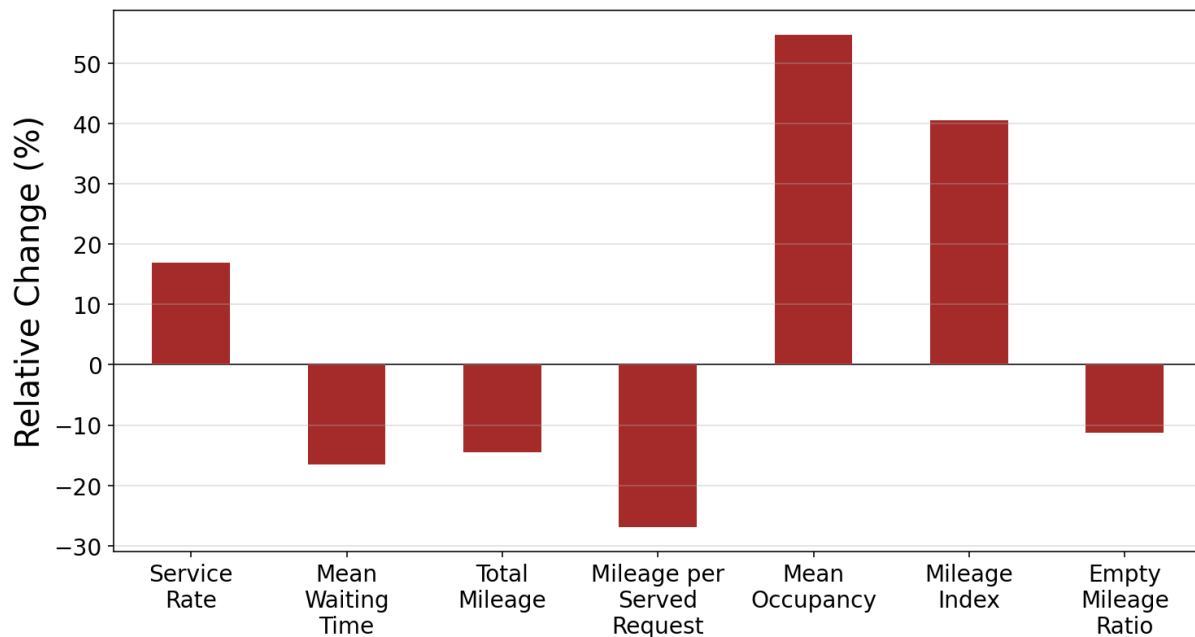


Figure 5.24: Relative change in system KPIs from ride-hailing to ride-pooling.

5.3.5. Impact of Batch Period

In batch assignment, requests are collected over a fixed batch period and assigned at the end of that period. The batch period is determined once at the beginning of the service and remains constant throughout the entire operation.

This section investigates whether varying the batch period has any measurable effect on the performance of the on-demand system. For this analysis, the default batch period of 2 minutes is adjusted to two additional values, resulting in three scenarios with batch periods of 1, 2, and 3 minutes. All the operational parameters remain the same across all scenarios, and are identical to those in the base scenario: fleet size of 6, shuttle capacity of 4, maximum waiting time (MWT) of 10 minutes, and maximum detour ratio (MDR) of 1.7.

The results of the system KPIs across these three scenarios are summarized in Table 5.3. The findings suggest that changing the batch period does not have a significant impact on system KPIs. All metrics remain relatively stable across the scenarios, except for “mean waiting time”, which shows a clear increase as the batch period gets longer.

This is an expected outcome: passengers who submit a request to the system do not receive an immediate response, but must wait until the end of the current batch period. Thus, a longer batch period naturally

results in longer waiting times. Specifically, in the 1-minute batch period scenario, the mean waiting time is 277 seconds; in the 2-minute scenario, it increases to 294 seconds; and in the 3-minute scenario, it rises further to 316 seconds. Interestingly, the increase in waiting times is around 20 seconds at each step, which is less than 30 seconds, the increase of expected value of the waiting times due to batch period.

Table 5.3: Comparison of system KPIs across three scenarios with varying batch period.

KPI	Batch Period		
	1 minute	2 minutes	3 minutes
Service Rate	0.87	0.87	0.88
Sharing Rate	0.62	0.63	0.63
Mean Waiting Time (s)	277.29	294.11	315.97
Mean Detour Time (s)	59.77	60.87	58.69
Total Mileage (km)	769.07	767.35	765.31
Mileage per Served Request (km)	0.81	0.81	0.81
Empty Mileage Ratio	0.26	0.27	0.26
Mileage Index	0.95	0.95	0.96
Mean Occupancy	1.09	1.08	1.09

5.3.6. Comparison of Matching Strategies

In this section, the matching strategies are compared based on the system KPIs. Three matching strategies are analyzed here. The first two are the spontaneous and batch assignment strategy, as covered in the previous sections. The third is an extended version of the batch assignment.

In the standard batch assignment, a request is assigned to a shuttle only once, and this assignment does not change during the simulation. However, an alternative approach allows for reassignment of requests that have already been assigned but not yet picked up. In this strategy, such requests may be reassigned during the following batch period, with the hope that a favorable match becomes available. This strategy is referred to as “batch with reassignment” and represents the third matching strategy.

All scenarios share the same operational parameters as the base case: fleet size of 6, shuttle capacity of 4, maximum waiting time (MWT) of 10 minutes, and maximum detour ratio (MDR) of 1.7. In addition, the same request pool, consisting of 1,082 requests, is used across all strategies.

Figure 5.25 shows that the service rate increases only slightly when the strategy shifts from spontaneous to batch assignment. In the literature, this increase is typically more substantial. For example, Engelhardt et al. (2020) employed similar algorithms for both spontaneous and batch strategies, but observed a greater increase. This discrepancy can be attributed to the size of the travel demand: their study examined simulations with 10,000 to 120,000 requests, an order of magnitude larger than the demand considered here. Moreover, their analyses confirmed that the difference in service rates between the two strategies diminishes in low-demand scenarios, which aligns with the results observed in Figure 5.25.

Additionally, batch with reassignment strategy produces a modest improvement over standard batch assignment, increasing the service rate from 0.87 to 0.89.

For the sharing rate, a slight increase is observed from spontaneous to batch assignment (0.61 to 0.63). However, the reassignment strategy does not yield any further enhancement in this metric.

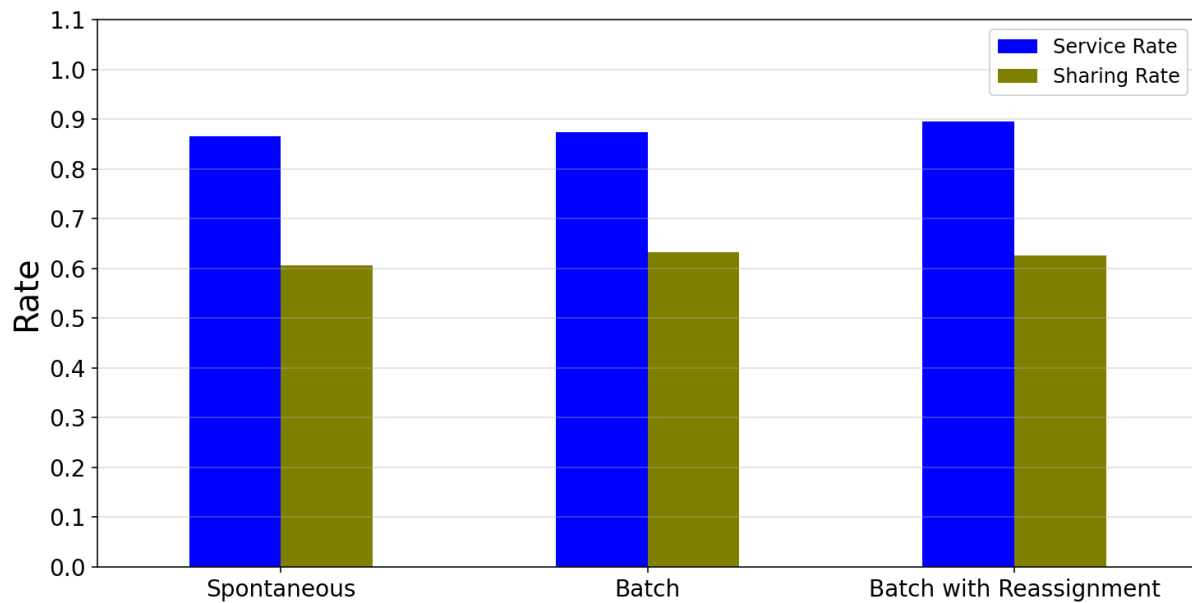


Figure 5.25: Comparison of matching strategies in terms of service rate and sharing rate.

Figure 5.26 illustrates the change of mean waiting time and mean detour time across the three strategies. A notable increase in waiting time is observed when moving from the spontaneous to the batch strategy. This outcome is expected, as requests in the batch strategy, different from spontaneous, are not processed immediately upon arrival but are held until the end of the current batch period. This in turn increases the mean waiting time by 15%, from 257 to 294 seconds. However, incorporating reassignment within the batch strategy helps to mitigate this effect: the mean waiting time decreases back to the level observed under the spontaneous strategy. This improvement is likely due to reassignment being based on more up-to-date system information, which enables better matching for passengers who have been assigned but not yet picked up.

In contrast, the mean detour time remains relatively stable between the spontaneous and batch strategies. Introducing the reassignment strategy yields a modest improvement, reducing the mean detour time by 6%, from 61 to 57 seconds.

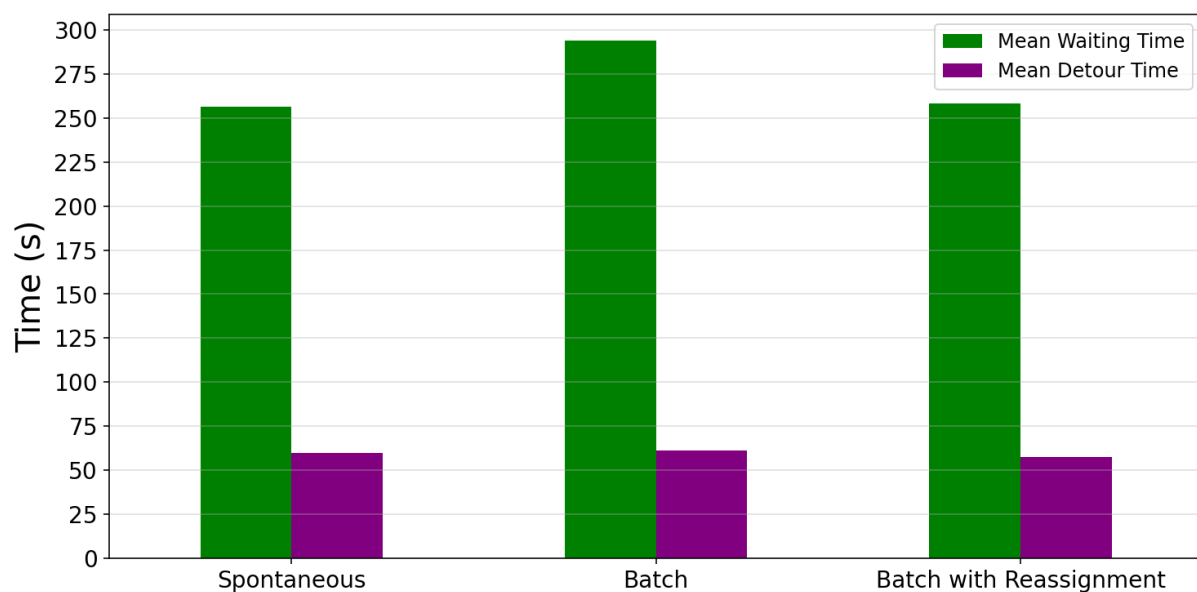


Figure 5.26: Comparison of matching strategies in terms of mean waiting time and mean detour time.

In Figure 5.27, the operator-related KPIs mean occupancy, mileage index, and empty mileage ratio are analyzed. The mean occupancy remains nearly unchanged between spontaneous and batch strategies, with a slight increase observed when the reassignment strategy is applied. Mileage index also follows a similar trend, showing a marginal improvement with reassignment. The empty mileage ratio, however, remains constant across all three assignment strategies.

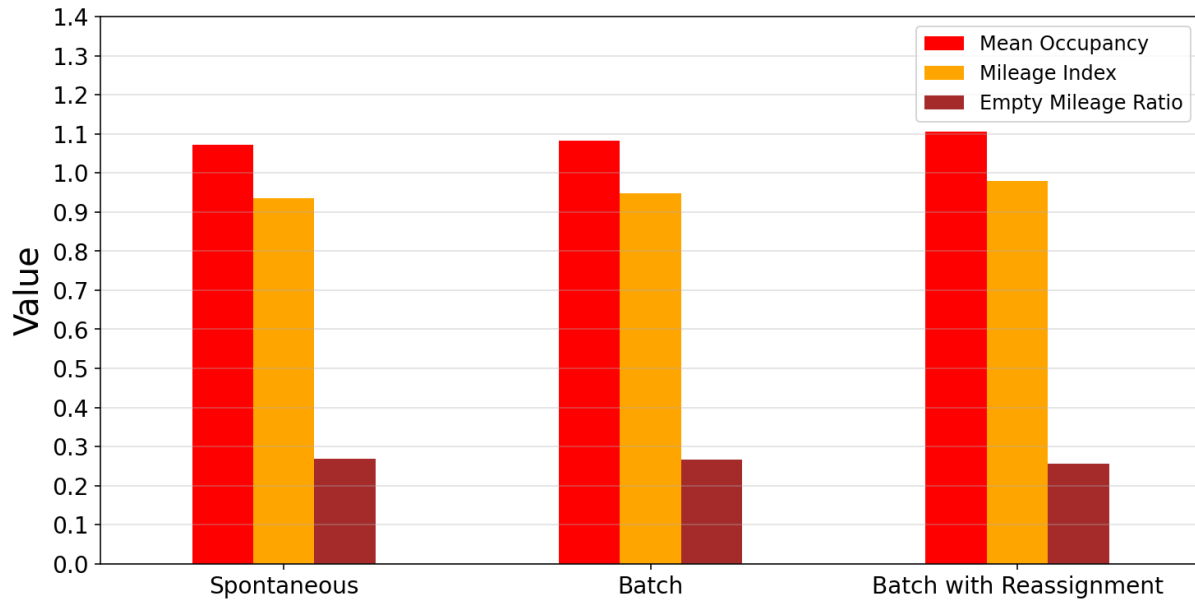


Figure 5.27: Comparison of matching strategies in terms of mean occupancy, mileage index, and empty mileage ratio.

In Figure 5.28, the variation in total mileage and mileage per served request across the different strategies is presented. The total mileage exhibits a slight decrease when moving from spontaneous to batch assignment, and a further minor reduction with the inclusion of reassignment. The declines in mileage per served request are slightly more pronounced.

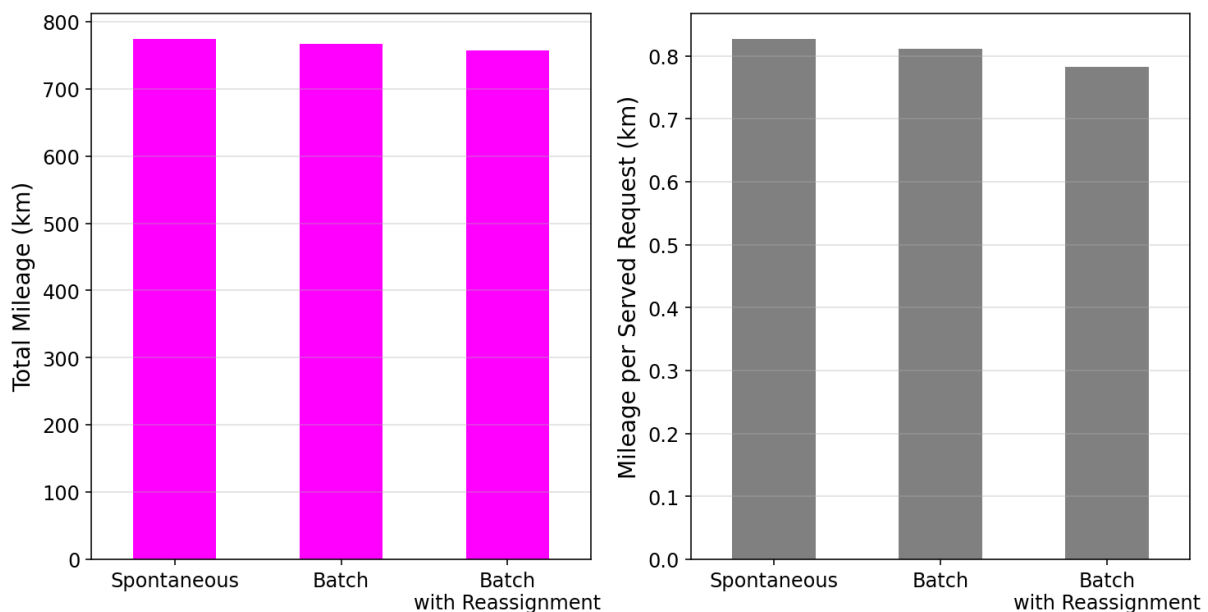


Figure 5.28: Comparison of matching strategies in terms of total mileage and mileage per served request.

Overall, it can be concluded that batch assignment strategy performs slightly better than the spontaneous strategy. However, the improvements are minimal, likely due to the conditions of the simulation experiments, particularly the limited size of the request pool. Among all system KPIs, mean waiting time is the only one that worsens with batch assignment, though this can be recovered by using the reassignment strategy. In addition, other KPIs also show slight improvements under the reassignment strategy.

5.4. Summary

This chapter presents an extensive evaluation of an on-demand ride-pooling system through simulation experiments conducted in a multimodal microsimulation environment. Building upon the matching algorithms and simulation setup introduced in earlier chapters, the experiments assessed the effects of various operational parameters and strategic design choices on key performance indicators (KPIs).

In Section 5.1, the initial experimental setup was outlined, including key assumptions, traffic input parameters, simulation resolution, and total simulation period. In addition, the required number of simulation runs was determined to ensure statistical significance.

Section 5.2 explored the spontaneous assignment strategy, where requests are matched to shuttles immediately upon arrival. A series of simulation runs evaluated the impact of four operational parameters: fleet size, shuttle capacity, maximum waiting time (MWT), and maximum detour ratio (MDR), on seven KPIs: service rate, sharing rate, mean waiting time, mean detour time, mean mileage, mean occupancy, and mileage index. For this, a special illustration technique was employed to observe the effects of each single operational parameter on these KPIs.

In Section 5.3, the batch assignment strategy, where the assignment of requests occurs in time batches, was examined under increased travel demand, with a batch period of 2 minutes. Here, firstly the temporal evolution of specific KPIs throughout the entire simulation day was investigated. Results demonstrated that the distribution of demand during the day had a pronounced effect on the KPIs. An accompanying analysis of fleet utilization highlighted variations in shuttle occupancies closely tied to fluctuations in demand.

Subsequent analyses in Section 5.3 explored various scenarios and their impacts on different KPIs. Initially, scenarios with varying travel demand were created and the changes in KPIs were discussed. This was followed by an analysis of weighting parameters in the cost function, enabling prioritization of either operator's or passenger's perspective. Here, KPI changes across scenarios provided insights into development of a flexible system strategy.

Next, a comparison between ride-hailing and ride-pooling confirmed the significant advantage of pooling, based on system KPIs. It was observed from the results that allowing trip sharing through ride-pooling substantially improved system performance compared to ride-hailing, where trips are realized exclusively.

Further analysis investigated the impact of variations in batch period on the on-demand shuttle system. It was shown that almost all KPIs remained unaffected, only mean waiting time experienced an increase with longer batch periods.

Lastly, three matching strategies; spontaneous, batch, and batch with reassignment, were compared. While batch assignment showed modest improvements over spontaneous assignment, reassignment

further enhanced performance by allowing previously matched but unpicked requests to be reassigned to better options.

Despite the complexity of the simulation, runtimes remained manageable. Each 17-hour simulation run required approximately 45 minutes in Vissim's quick mode, regardless of whether spontaneous or batch assignment was used.

Overall, these experimental results confirm that the developed matching methodologies can be effectively applied within a microscopic simulation. These experiments offer valuable insights into the operational performance of on-demand shuttle systems, addressing **Research Question 4** of this thesis: **“What are the impacts of system parameters and design conditions on the on-demand system KPIs, and how do these KPIs evolve throughout the microsimulation?”**. The COM interface plays a crucial role in this context, its flexibility not only enables the implementation of matching and dispatching algorithms but also facilitates the analysis of numerous system KPIs.

6. Conclusions and Outlook

This chapter concludes the thesis by providing a summary of the key findings, discussing the limitations of the model, and outlining opportunities for future research.

6.1. Summary and Conclusions

This thesis has addressed the modeling, simulation, and evaluation of on-demand ride-pooling services within a dynamic transportation environment. Through a combination of extensive literature review, algorithm development, and simulation-based experimentation, the research has contributed both methodologically and practically to the understanding and improvement of ride-pooling systems. By focusing on real-time request-vehicle matching strategies and evaluating their performance in a detailed microscopic simulation environment, the work bridges a key gap between theoretical modeling and practical implementation of on-demand mobility services. Moreover, it provides guidance for researchers aiming to integrate on-demand services into a microsimulation environment.

Chapter 1 of this thesis introduced the research topic and defined the key research questions that guided the study.

In Chapter 2, a comprehensive literature review was conducted to gain insights for the implementation. Firstly, different types of on-demand systems were described (Section 2.1). This was followed by a review of well-established algorithms related to shortest route calculation (Section 2.2) and vehicle routing problems (VRP) (Section 2.3). The chapter then reviewed simulation platforms previously used for modeling on-demand services. It was found that microsimulation tools were rarely employed in this context, identifying a clear research gap. In addition, the matching strategies, objective function perspectives, and key performance indicators used in previous studies were elaborated (Section 2.4).

Chapter 3 focused on the matching methodologies utilized in this thesis. Following a brief introduction to the components of an on-demand ride-hailing/ride-pooling system, and differentiation of spontaneous and batch assignment strategies, a methodology for spontaneous assignment of requests was introduced, where the arriving requests are immediately assigned to a vehicle (Section 3.2). Subsequently, a batch assignment method was presented, in which requests are collected over a defined time interval and assigned simultaneously (Section 3.3). Both methods were formulated in a way that both operator's and passenger's perspectives were incorporated. Weighting parameters in the cost functions enabled to put more emphasis on either side according to the overall system strategy. Moreover, operational constraints on vehicle capacity, waiting time, and detour ratio were defined to eliminate operationally undesirable options. Both methods were described in detail and supported with illustrative examples to enhance clarity and understanding for the readers.

Chapter 4 provided insights into the application of the on-demand shuttle system in the microscopic traffic simulation. The thesis employed the detailed microscopic traffic simulation platform PTV Vissim. This made a distinction from the literature, which sets this work apart from much of the existing literature, that predominantly relies on macroscopic or agent-based simulation frameworks.

A detailed case study was carried out within the campus area of the University of the Bundeswehr Munich (Section 4.1). In addition to the general simulation setup based on the specific characteristics of the campus, a methodology was developed to feed the simulation with vehicle inputs and turning ratios

for transport modes except on-demand shuttles. This was achieved by utilizing data from traffic detection cameras installed at different locations on the campus. A Python-based interface was implemented to gather the relevant historical data from the cameras and import it into the Vissim simulation. This is one of the essential contributions of this thesis, as it enables an automatic connection between a data collection technology and the simulation model, which simplifies the essential data entry in the simulation (Section 4.2.1.1).

Another major contribution lies in the design and implementation of the simulation setup for the on-demand shuttle traffic. Conventional microsimulation frameworks typically rely on static routing, in which vehicles are assigned predefined paths at intersections, which can be regarded as decision points. This approach can be useful in case there are not several travel possibilities from A to B. An on-demand system, however, is different from this standard case, since vehicles start their journey triggered by a real-time event (e.g., an upcoming request, or bringing a passenger to drop-off point) and the route taken is calculated promptly as the shortest path/time based on their current positions. Moreover, vehicle movements can occur from every stop point to every stop point, which leads to several number of movement possibilities. Hence, a static routing becomes infeasible. To address this challenge, the simulation was enhanced with several features, as described in Sections 4.2.2 and 4.2.3: usage of dynamic assignment and COM interface, construction of nodes and edges for shortest path calculation, configuration of shuttle stops, introduction of new attributes for Vissim objects, and control of the simulation through an external Python code. In addition, a realistic request pool for the on-demand operation was generated by analyzing the data of the micromobility sharing system of the campus, MORE Sharing (Section 4.3). This ensured that the simulated travel demand reflected actual usage patterns and temporal distribution of mobility needs.

In Chapter 5, the case study was implemented and an extensive evaluation of the on-demand ride-pooling system was carried out through full-day simulation experiments. A broad set of key performance indicators (KPIs) was used to evaluate the system under different scenarios. These KPIs included both passenger-focused metrics such as service rate, sharing rate, mean waiting time and mean detour time, and operator-focused metrics such as total mileage, mileage per served request, vehicle occupancy, mileage index, and empty mileage ratio.

Initially, a series of simulation runs using spontaneous assignment strategy assessed the effects of operational parameters on these KPIs (Section 5.2). Here, it was found that the highest service rate was achieved with a fleet size of 4, shuttle capacity of 4, maximum waiting time of 10 minutes, and maximum detour ratio of 2.

These analyses were followed by simulations employing the batch assignment under increased travel demand, in order to investigate the temporal evolution of KPIs throughout the simulation day (Section 5.3.1). The results demonstrated that the distribution of demand significantly affected those. For example, during high demand periods, the system exhibited increased sharing rate, mean waiting time, mean detour time, mean occupancy, and fleet utilization, while the service rate declined.

The size of the request pool proved to be a critical factor in the system performance. As shown in the analyses in Section 5.3.2, increasing travel demand while keeping other parameters constant led to a decline in the service rate, empty mileage ratio and mileage per served request. In contrast, sharing rate, mean waiting time, mean detour time, mileage index, mean occupancy, and total mileage increased with higher demand levels.

Further analysis in Section 5.3.3 revealed that varying the weighting parameters in the cost function could shift the emphasis between operator and passenger side. The results indicated how passenger-related KPIs such as mean waiting time and mean detour time could be reduced when the balance shifted

from operator to passenger side. In the same manner, results demonstrated that in this case total mileage, mileage per served request and empty mileage ratio increased, and mileage index dropped.

A comparison of ride-hailing and ride-pooling under identical system configurations further highlighted the advantages of pooling (Section 5.3.4). Service rate increased by 17%, whereas mean waiting time and total mileage were reduced by 17% and 14%, respectively. These results underscored once again the value of pooling in on-demand shuttle systems, despite the trade-offs of increased detours and reduced passenger comfort due to shared trips.

The analysis comparing matching strategies revealed that batch strategy slightly outperformed spontaneous strategy in terms of sharing rate and total mileage. However, this benefit came at the cost of increased mean waiting times. The “batch with reassignment” strategy, where already assigned but unpicked passengers could be reassigned in the next batch period, could mitigate this effect on mean waiting time, and introduced further improvements into the batch assignment. Specifically, it led to increases in service rate, mean occupancy and mileage index, while reducing mean detour time, total mileage, and mileage per served request.

Overall, this thesis sought to address four research questions that were identified at the introduction chapter (Section 1.2):

Research Question 1: How can an on-demand ride-pooling system be implemented in a micro-simulation model, specifically in Vissim?

- This research question was addressed through the development of a customized implementation framework detailed in Section 4.2.2, 4.2.3, and 4.3.

Research Question 2: How can requests be matched with vehicles in an optimized way, taking both operator’s and passenger’s perspectives into account?

- This research question was addressed through the presented matching methodologies elaborated in Chapter 3.

Research Question 3: How to build an interface between a camera-based traffic data collection system and the microsimulation model, to automate the vehicle data feed?

- This research question was addressed through the developed methodology presented in Section 4.2.1.1.

Research Question 4: What are the impacts of system parameters and design conditions on the on-demand system KPIs, and how do these KPIs evolve throughout the microsimulation?

- This research question was addressed through the extensive analyses conducted in simulation experiments in Section 5.2 and Section 5.3.

6.2. Limitations of the Model

Although the microsimulation model developed in this thesis yielded established insights, some of its limitations must also be acknowledged.

The model was developed for a small-scale on-demand ride-pooling system, with a limited fleet and request pool size, especially compared to other ride-pooling studies in the literature. This raises the question of scalability: whether the model could be extended to large-scale environments, such as an entire city. In such cases, two challenges arise: Firstly, the basic simulation setup would need to be performed over a larger area, making it more complex and time-consuming. Secondly, as the fleet and request pool size expand, some heuristic methods would need to be integrated into the matching methodologies in order to have reasonable computational times. Therefore, the scalability of the presented model should be evaluated with caution.

The on-demand system assumed immediate requests without advanced bookings, and did not model passenger behavior such as cancellations, preferences, or the ability to reject offers. Incorporating these behavioral dynamics could provide a more realistic assessment of system performance and acceptance. Additionally, the system did not account for rebalancing of vehicles, which in practice plays a role in maintaining service availability and efficiency in real-world operations.

The implementation was carried out using the microsimulation tool Vissim, taking into account its specific data and object structure. Key objects used for the on-demand component include vehicles, nodes, edges, parking lots, vehicle paths (based on node sequences), and priority rules. Additionally, vehicle inputs, vehicle compositions, routing decisions, and routes are other objects used for automatic data feed of remaining transport modes. The methodology presented in this thesis is closely tailored to Vissim data model. This creates a limitation in transferring the developed methodologies and algorithms to other microsimulation tools. Although the foundational principles of most established microsimulation tools are generally similar, variations in their internal data models may necessitate adjustments to the implementation.

Some issues arose in Vissim during the multimodal simulation, particularly in the pedestrian dynamics within shared space areas. In Vissim, pedestrians are modeled as a special type of “vehicle” governed by no specific car-following model. In certain cases, conflict areas at intersections failed to function as intended: the intersections were blocked by two traffic participants (at least one of whom was a pedestrian) both yielding to each other, leading to an unrealistic deadlock situation.

One particular aspect that required careful consideration was the treatment of idle shuttles. In the current model, shuttles that completed their assigned routes remained at the stop until a new task was received. This occasionally led to complications, as the model did not allow more than one shuttle to occupy the same stop simultaneously. For instance, if a shuttle was idle at a stop and a second shuttle attempted to access the same stop for its own assignment, the latter would be forced to queue behind it for an extended, often unrealistic, duration. To address this, the algorithm was extended so that in such cases, the idle shuttle was sent to its depot point to free up the stop.

6.3. Future Research

This thesis lays a foundation for researchers that aim to implement on-demand systems in microscopic traffic simulations. The proposed methodology is particularly well-suited to small-scale contexts, such as university campuses, hospital complexes, or constrained urban neighborhoods. The knowledge from this thesis can be transferred to these use-cases easily. Furthermore, an important direction for future research involves scaling this approach to larger urban environments, such as entire cities. In such cases, simulations would employ significantly more vehicles and requests, requiring the integration of heuristic methods into the matching algorithms to ensure manageable computational times. Additionally, such large-scale simulations could investigate the broader impact of shifting the travel demand from private

cars to on-demand service on the traffic flow metrics such as congestions, queue lengths, delay times, and emissions.

Vehicle rebalancing strategies are particularly relevant in large-scale systems, where efficient fleet distribution becomes critical. Here, integrating machine learning techniques, such as reinforcement learning, could be explored to optimize vehicle repositioning. In scenarios employing electric vehicles, monitoring the battery status and charging events can be included in the on-demand operation.

In the current algorithm, the shortest paths are calculated based on the distance between nodes. A natural extension in large-scale systems would involve accounting for dynamic traffic conditions, especially in congested areas where travel times of edges vary continuously. In such settings, the current traffic situation can be extracted, and the edge weights could be modified in real-time, allowing Dijkstra's algorithm to rely on the shortest time instead of shortest distance.

The matching algorithms could also be enhanced to handle group bookings, where a request consists of more than one passenger. This would require adapting the capacity checking mechanism, passenger side cost calculations, and passenger-related KPI analyses accordingly. Moreover, the inclusion of walking events, where passengers need to walk to a nearby shuttle stop for pick-up, would increase the realism of the model. In this case, a minimum waiting time must be incorporated to the algorithms to ensure passengers arrive at the stop before the shuttle.

Another promising direction is the integration of passenger behavior into the simulation. This includes modeling of cancellations, no-shows, or rejection of offers, thereby capturing a more realistic operation. Furthermore, a more flexible service model can be introduced, allowing passengers to choose between exclusive (ride-hailing) and shared (ride-pooling) services. In other words, a hybrid operation of ride-hailing and ride-pooling can be modeled.

Another line of research could explore integrating advanced booking functionality alongside immediate requests, in case passengers do not want to travel at once, but want to book for a future time. In this case, advanced bookings should be integrated to the dynamic routing of vehicles in an appropriate way that the service for these bookings is always ensured.

Finally, integration of on-demand services with existing public transport systems presents an opportunity for further research in the context of intermodal operations. Including key public transport hubs in the simulation model would introduce latest-arrival constraints for passengers that use the on-demand service as a first-mile solution, such as those traveling to a suburban train station to make a scheduled transfer.

In conclusion, while this thesis presents a robust framework for simulating on-demand ride-pooling in a microscopic environment, it also opens up a wide range of possibilities for further refinement, scalability, and real-world applicability.

7. References

- Ackermann, T. (2023). Linienbedarfsverkehr: Zukunftsgerecht, integriert und nachfragegesteuert. Warum eine ÖPNV-Angebotsoffensive im Linien- und Linienbedarfsverkehr notwendig ist. VDV - Verband Deutscher Verkehrsunternehmen e.V.
- Alita Anruflinientaxi. (2024). DVB. <https://www.dvb.de/de-de/fahrplan/nachtverkehr/anruflinientaxi> (accessed on 12.12.2024)
- Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3), 462–467. <https://doi.org/10.1073/pnas.1611675114>
- Armellini, M. G. (2021). A Tool for Simulating Demand Responsive Transport Systems in SUMO. 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Heraklion, Greece. <https://doi.org/10.1109/MT-ITS49943.2021.9529311>
- Arslan, O., & Hoffmann, S. (2023). Implementation of a Spontaneous Matching Algorithm for On-Demand Shuttle Systems in Microsimulation. *Proceedings of the 9th IEEE International Smart Cities Conference 2023. IEEE International Smart Cities Conference 2023*, Bucharest, Romania.
- Arslan, O., & Hoffmann, S. (2024). Implementierung eines spontanen Matching-Algorithmus für On-Demand Shuttle-Systeme in der Mikrosimulation. *Heureka - Optimierung in Verkehr und Transport 2024*, Stuttgart, Germany.
- Arslan, O., & Hoffmann, S. (2025a). Impacts of On-Demand Shuttle System Parameters on its Metrics: A Microsimulation Study. *Transportation Research Procedia*, 86, 491–499. <https://doi.org/10.1016/j.trpro.2025.04.062>
- Arslan, O., & Hoffmann, S. (2025b). Integration of On-Demand Ride-Pooling into a Microscopic Traffic Flow Simulation Environment. 11th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS), Porto, Portugal.
- Assmann, C. (2020). Systemanalyse der Wirkungen von On-Demand-Mobilitätsdienstleistungen im Kontext nachhaltiger urbaner Mobilität [Dissertation]. Technical University of Munich.
- AST Harburg. (2024). Landkreis Harburg. <https://www.kvg-bus.de/fahrplaene/landkreis-harburg/bedarfsorientierter-verkehr/ast> (accessed on 12.12.2024)
- Atasoy, B., Ikeda, T., Song, X., & Ben-Akiva, M. E. (2015). The concept and impact analysis of a flexible mobility on demand system. *Transportation Research Part C: Emerging Technologies*, 56, 373–392. <https://doi.org/10.1016/j.trc.2015.04.009>
- Berbeglia, G., Pesant, G., & Rousseau, L.-M. (2011). Checking the Feasibility of Dial-a-Ride Instances Using Constraint Programming. *Transportation Science*, 45(3), 399–412. <https://doi.org/10.1287/trsc.1100.0336>
- Bierlaire, M. (2015). *Optimization: Principles and Algorithms (2.)*. EPFL Press. <https://doi.org/10.55430/6116V1MB>

- Bilali, A., Dandl, F., Fastenrath, U., & Bogenberger, K. (2019a). An Analytical Model for On-Demand Ride Sharing to Evaluate the Impact of Reservation, Detour and Maximum Waiting Time. *IEEE Intelligent Transportation Systems Conference (ITSC)*, 1715–1720. <https://doi.org/10.1109/ITSC.2019.8917280>
- Bilali, A., Dandl, F., Fastenrath, U., & Bogenberger, K. (2019b). Impact of service quality factors on ride sharing in urban areas. *6th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, 1–8. <https://doi.org/10.1109/MTITS.2019.8883364>
- Bischoff, J., Maciejewski, M., & Nagel, K. (2017). City-wide shared taxis: A simulation study in Berlin. *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 275–280. <https://doi.org/10.1109/ITSC.2017.8317926>
- Blablacar. (2024). Blablacar Mitfahrgelegenheit. <https://www.blablacar.de/> (accessed on 15.12.2024)
- Bolt. (2024). Bolt Transport Technology. <https://bolt.eu/> (accessed on 15.12.2024)
- Bonabeau, E. (2002). Agent-Based Modeling: Methods and Techniques for Simulating Human Systems. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10), 7280–7287.
- Borndörfer, R., Grötschel, M., Klostermeier, F., & Küttner, C. (1999). Telebus Berlin: Vehicle Scheduling in a Dial-a-Ride System. In N. H. M. Wilson (Ed.), *Computer-Aided Transit Scheduling* (pp. 391–422). Springer Berlin Heidelberg.
- Bussi. (2024). Ruhrbahn Essen Ride-Pooling. <https://bussi.ruhrbahn.de/> (accessed on 15.12.2024)
- Cambio. (2024). Cambio Carsharing. <https://www.cambio-carsharing.de> (accessed on 15.12.2024)
- Cordeau, J.-F. (2006). A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Operations Research*, 54(3), 573–586.
- Cordeau, J.-F., & Laporte, G. (2003). The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2). <https://doi.org/10.1007/s10288-002-0009-8>
- Cordeau, J.-F., & Laporte, G. (2007). The dial-a-ride problem: Models and algorithms. *Annals of Operations Research*, 153(1), 29–46. <https://doi.org/10.1007/s10479-007-0170-8>
- Cortés, C. E., Pagès, L., & Jayakrishnan, R. (2005). Microsimulation of Flexible Transit System Designs in Realistic Urban Networks. *Transportation Research Record: Journal of the Transportation Research Board*, 1923, 153–163.
- Daamen, W., Buisson, C., & Hoogendoorn, S. P. (2015). *Traffic Simulation and Data—Validation Methods and Applications*. CRC Press.
- Dandl, F., Bracher, B., & Bogenberger, K. (2017). Microsimulation of an autonomous taxi-system in Munich. *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, 833–838. <https://doi.org/10.1109/MTITS.2017.8005628>
- Dandl, F., Engelhardt, R., & Bogenberger, K. (2021). On the Dynamism of User Rejections in Mobility-on-Demand Systems. *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 3399–3404. <https://doi.org/10.1109/ITSC48978.2021.9564918>

- Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 6(1), 80–91. <https://doi.org/10.1287/mnsc.6.1.80>
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269–271.
- Engelhardt, R., & Bogenberger, K. (2021). Benefits of Flexible Boarding Locations in On-Demand Ride-Pooling Systems. 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Heraklion, Greece. <https://doi.org/10.1109/MT-ITS49943.2021.9529284>
- Engelhardt, R., Dandl, F., Bilali, A., & Bogenberger, K. (2019). Quantifying the Benefits of Autonomous On-Demand Ride-Pooling: A Simulation Study for Munich, Germany. *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2992–2997. <https://doi.org/10.1109/ITSC.2019.8916955>
- Engelhardt, R., Dandl, F., & Bogenberger, K. (2020). Speed-up Heuristic for an On-Demand Ride-Pooling Algorithm. <https://arxiv.org/pdf/2007.14877>
- Engelhardt, R., Dandl, F., Syed, A.-A., Zhang, Y., Fehn, F., Wolf, F., & Bogenberger, K. (2022a). FleetPy: A Modular Open-Source Simulation Tool for Mobility On-Demand Services (No. arXiv:2207.14246). *arXiv*. <https://doi.org/10.48550/arXiv.2207.14246>
- Engelhardt, R., Mahmassani, H., & Bogenberger, K. (2024). Vorhersagebasierte Fahrzeugrepositionierung für On-Demand- Ride-Pooling-Dienste. *Heureka - Optimierung in Verkehr und Transport 2024*, Stuttgart, Germany. <https://doi.org/10.1287/ae432231-d1a0-4414-9311-e8d54e554833>
- Engelhardt, R., Malcolm, P., Dandl, F., & Bogenberger, K. (2022b). Competition and Cooperation of Autonomous Ridepooling Services: Game-Based Simulation of a Broker Concept. *Frontiers in Future Transportation*, 3. <https://doi.org/10.3389/ffutr.2022.915219>
- Fagnant, D. J., & Kockelman, K. M. (2018). Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas. *Transportation*, 45(1), 143–158. <https://doi.org/10.1007/s11116-016-9729-z>
- FGSV. (2006). Hinweise zur mikroskopischen Verkehrsflusssimulation—Grundlagen und Anwendung. Forschungsgesellschaft für Strassenverkehr (FGSV).
- Fiedler, D., Certicky, M., Alonso-Mora, J., & Cap, M. (2018). The Impact of Ridesharing in Mobility-on-Demand Systems: Simulation Case Study in Prague. 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 1173–1178. <https://doi.org/10.1109/ITSC.2018.8569451>
- Flexa. (2024). LVB Leipzig Ride-Pooling. <https://leipzig-move.de/flexa/> (accessed on 15.12.2024)
- Flinkster. (2024). Flinkster Carsharing. <https://www.flinkster.de/de/start> (accessed on 15.12.2024)
- Flood, M. (1956). The Traveling-Salesman Problem. *Operations Research*, 4(1), 61–75.
- Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, 5(6), 345–345. <https://doi.org/10.1145/367766.368168>
- FreeNow. (2024). FreeNow Carsharing. <https://www.free-now.com/de/> (accessed on 15.12.2024)

- Friedrich, M., Hartl, M., & Magg, C. (2018). A modeling approach for matching ridesharing trips within macroscopic travel demand models. *Transportation*, 45(6), 1639–1653. <https://doi.org/10.1007/s11116-018-9957-5>
- Ghandeharioun, Z., & Kouvelas, A. (2023). Real-time ridesharing operations for on-demand capacitated systems considering dynamic travel time information. *Transportation Research Part C: Emerging Technologies*, 151, 104115. <https://doi.org/10.1016/j.trc.2023.104115>
- Gkiotsalitis, K. (2022). *Public Transport Optimization*. Springer International Publishing. <https://doi.org/10.1007/978-3-031-12444-0>
- Gurobi Optimization LLC. (2025). *Gurobi Optimizer Reference Manual* [Online]. <http://www.gurobi.com>
- Harmann, D., Scharfe, R., Trick, J.-S., & Friedrich, B. (2024). Empirical Investigation of Dwell Times in Ridepooling Systems. *Transportation Research Procedia*, 78, 119–126. <https://doi.org/10.1016/j.trpro.2024.02.016>
- Hartleb, J., Friedrich, M., & Richter, E. (2021a). Umlaufbildung für On-Demand Fahrzeugflotten in makroskopischen Nachfragemodellen. *Heureka - Optimierung in Verkehr und Transport 2021*, Stuttgart, Germany.
- Hartleb, J., Friedrich, M., & Richter, E. (2021b). Vehicle scheduling for on-demand vehicle fleets in macroscopic travel demand models. *Transportation*, 49(4), 1133–1155. <https://doi.org/10.1007/s11116-021-10205-4>
- Harz, J., & Sommer, C. (2021). Erfolgsfaktoren und Hemmnisse des integrierten Ridesharing in ländlichen Räumen. *Heureka - Optimierung in Verkehr und Transport 2021*, Stuttgart, Germany.
- Heiner Liner. (2024). RMV Darmstadt Ride-Pooling. <https://www.heinerliner.de/> (accessed on 15.12.2024)
- Ho, S. C., Szeto, W. Y., Kuo, Y.-H., Leung, J. M. Y., Petering, M., & Tou, T. W. H. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111, 395–421. <https://doi.org/10.1016/j.trb.2018.02.001>
- Hopper. (2024). RMV Offenbach Ride-Pooling. <https://www.kvgof-hopper.de/> (accessed on 15.12.2024)
- Horni, A., Nagel, K., & Axhausen, K. W. (2016). *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press. <https://doi.org/10.5334/baw>
- Hyland, M., & Mahmassani, H. S. (2018). Dynamic autonomous vehicle fleet operations: Optimization-based strategies to assign AVs to immediate traveler demand requests. *Transportation Research Part C: Emerging Technologies*, 92, 278–297. <https://doi.org/10.1016/j.trc.2018.05.003>
- Iacobucci, R., & Schmöcker, J.-D. (2021). Dynamic pricing for ride-hailing services considering relocation and mode choice. 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Heraklion, Greece. <https://doi.org/10.1109/MT-ITS49943.2021.9529301>
- Johnson, D., & Chaniotakis, E. (2021). Innovative last mile delivery concepts: Evaluating last mile delivery using a traffic simulator. 2021 7th International Conference on Models and Technologies for

- Intelligent Transportation Systems (MT-ITS), Heraklion, Greece. <https://doi.org/10.1109/MT-ITS49943.2021.9529279>
- Jung, J., & Jayakrishnan, R. (2014). Simulation Framework for Modeling Large-Scale Flexible Transit Systems. *Transportation Research Record: Journal of the Transportation Research Board*, 2466, 31–41. <https://doi.org/10.3141/2466-04>
- Kagerbauer, M., Kosterz, N., Wilkes, G., Dandl, F., Engelhardt, R., Glöckl, U., Fraedrich, E., & Zwick, F. (2021). Ridepooling in der Modellierung des Gesamtverkehrs—Methodenbericht zur MOIA Begleitforschung. *Karlsruher Institut für Technologie (KIT)*. <https://doi.org/10.5445/IR/1000141282>
- Kloth, H., & Mehler, S. (2018). Nachfragegesteuerte Verkehre oder On-Demand-Ridepooling? *Der Nahverkehr*, 2018(06), 36–39.
- Kucharski, R., & Cats, O. (2021). MaaSSim—Agent-based two-sided mobility platform simulator. 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Heraklion, Greece.
- Kucharski, R., & Cats, O. (2022). Simulating two-sided mobility platforms with MaaSSim. *PLoS ONE*, 17(6). <https://doi.org/10.1371/journal.pone.0269682>
- Kümmel, M. (2016). Taxis, passengers and stable marriage: Stable simultaneous assignment of taxis to passenger booking requests [Dissertation, Technical University of Munich]. <https://doi.org/10.32657/10220/45961>
- Liebchen, C., Lehnert, M., Mehler, C., & Schiefelbusch, M. (2020). Ridepooling-Effizienz messbar machen: Betriebliche Effizienzgrößen für Ridepooling-Systeme. *Der Nahverkehr*, 2020(09), 18–21. <https://doi.org/10.15771/1318>
- Liu, Y., & Samaranayake, S. (2022). Proactive Rebalancing and Speed-Up Techniques for On-Demand High Capacity Ridesourcing Services. *IEEE Transactions on Intelligent Transportation Systems*, 23(2), 819–826. <https://doi.org/10.1109/TITS.2020.3016128>
- Lyft. (2024). Lyft Transport Platform. <https://www.lyft.com/> (accessed on 15.12.2024)
- Miles. (2024). Miles Mobility Carsharing. <https://miles-mobility.com/de> (accessed on 15.12.2024)
- Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM*, 7(4), 326–329. <https://doi.org/10.1145/321043.321046>
- MOIA. (2024). MOIA Hamburg Ride-Pooling. <https://www.moia.io/> (accessed on 15.12.2024)
- MORE Mobility Research. (2024). MORE - Munich Mobility Research Campus für die Universität der Bundeswehr.
- Musolino, G., Rindone, C., & Vitetta, A. (2021). A modelling framework to simulate paths and routes choices of freight vehicles in sub-urban areas. 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Heraklion, Greece. <https://doi.org/10.1109/MT-ITS49943.2021.9529293>
- MVV Flex. (2024). MVV Munich Ride-Pooling. <https://www.mvv-muenchen.de/mobilitaetsangebote/flex/index.html/> (accessed on 15.12.2024)

- Narayan, J., Cats, O., Van Oort, N., & Hoogendoorn, S. (2017). Performance assessment of fixed and flexible public transport in a multi agent simulation framework. *Transportation Research Procedia*, 27, 109–116. <https://doi.org/10.1016/j.trpro.2017.12.029>
- Narayan, J., Cats, O., Van Oort, N., & Hoogendoorn, S. (2020a). Integrated route choice and assignment model for fixed and flexible public transport systems. *Transportation Research Part C: Emerging Technologies*, 115, 102631. <https://doi.org/10.1016/j.trc.2020.102631>
- Narayan, J., Cats, O., Van Oort, N., & Hoogendoorn, S. P. (2020b). Fleet size determination for a mixed private and pooled on-demand system with elastic demand. *Transportmetrica A: Transport Science*, 17(4), 897–920. <https://doi.org/10.1080/23249935.2020.1819910>
- Narayan, J., Cats, O., Van Oort, N., & Hoogendoorn, S. P. (2022). On the scalability of private and pooled on-demand services for urban mobility in Amsterdam. *Transportation Planning and Technology*, 45(1), 2–18. <https://doi.org/10.1080/03081060.2021.2017214>
- Narayanan, S., Chaniotakis, E., & Antoniou, C. (2020). Shared autonomous vehicle services: A comprehensive review. *Transportation Research Part C: Emerging Technologies*, 111, 255–293. <https://doi.org/10.1016/j.trc.2019.12.008>
- Negro, P., Ridderskamp, D., Paul, M., Fehn, F., Belzner, H., & Bogenberger, K. (2021). Cost Structures of Ride-Hailing Providers in the Context of Vehicle Electrification and Automation. 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Heraklion, Greece. <https://doi.org/10.1109/MT-ITS49943.2021.9529308>
- Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1), 1–11. <https://doi.org/10.1016/j.ejor.2012.08.015>
- Pobudzei, M., & Hoffmann, S. (2024). User Segmentation based on Usage Frequency: A Case Study of a Multimodal Shared Micromobility in a Non-Urban Campus Environment. 2024 IEEE Intelligent Vehicles Symposium (IV), 992–999. <https://doi.org/10.1109/IV55156.2024.10588699>
- Pobudzei, M., Kemmerzehl, R., & Ulusoy, C. (2023). Beyond the City: Testing Shared Micromobility with MORE Sharing. *Proceedings of EWGT 2023. 25th Euro Working Group on Transportation Meeting (EWGT 2023)*, Santander, Spain.
- Psaraftis, H. N. (1980). A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. *Transportation Science*, 14(2), 130–154. <https://doi.org/10.1287/trsc.14.2.130>
- Psaraftis, H. N. (1983). An Exact Algorithm for the Single Vehicle Many-to-Many Dial-A-Ride Problem with Time Windows. *Transportation Science*, 17(3), 351–357. <https://doi.org/10.1287/trsc.17.3.351>
- Psaraftis, H. N. (1995). Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61(1), 143–164. <https://doi.org/10.1007/BF02098286>
- Psaraftis, H. N., Wen, M., & Kontovas, C. A. (2016). Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1), 3–31. <https://doi.org/10.1002/net.21628>
- PTV AG. (2024). VISSIM Version 2024. Karlsruhe, Germany: <https://www.ptvgroup.com/de/loesungen/produkte/ptv-vissim>

- PuLP. (2025). PuLP : A Linear Programming Toolkit for Python. <https://github.com/coin-or/pulp>
- Revierflitzer. (2024). STOAG Oberhausen Ride-Pooling. <https://www.stoag.de/service/revierflitzer.html/> (accessed on 15.12.2024)
- Richter, E., Friedrich, M., Migl, A., & Hartleb, J. (2019). Integrating ridesharing services with automated vehicles into macroscopic travel demand models. 6th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), 1–8. <https://doi.org/10.1109/MTITS.2019.8883315>
- Richter, M., & Hess, J. (2021). Rentabilität und Wettbewerbsfähigkeit einer Robotaxi Flotte – Eine makroskopische Verkehrssimulation in der Stadt Zürich. *Journal für Mobilität und Verkehr*, 2021(08), 2–13.
- Ritzer-Angerer, P. (2021). Sharing Economy trifft ÖPNV — das neue Personenbeförderungsgesetz. *Wirtschaftsdienst*, 101(10), 789–794. <https://doi.org/10.1007/s10273-021-3025-z>
- Ruch, C., Lu, C., Sieber, L., & Frazzoli, E. (2020). Quantifying the Efficiency of Ride Sharing. *IEEE Transactions on Intelligent Transportation Systems*, 22, 1–6. <https://doi.org/10.1109/TITS.2020.2990202>
- Rufbus Lüneburg. (2024). Landkreis Lüneburg. <https://www.landkreis-lueneburg.de/fuer-unsere-buergerinnen-und-buerger/mobil-im-landkreis/bus-und-bahn/rufbus.html> (accessed on 12.12.2024)
- RufTaxi. (2024). RufTaxi in MVV Munich. <https://www.mvv-muenchen.de/mobilitaetsangebote/bedarfsverkehr/ruftaxi-flexlinie/index.html> (accessed on 12.12.2024)
- Santi, P., Resta, G., Szell, M., Sobolevsky, S., Strogatz, S. H., & Ratti, C. (2014). Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37), 13290–13294. <https://doi.org/10.1073/pnas.1403657111>
- Schmöller, S., Niels, T., Hardt, C., Lippoldt, K., Dandl, F., & Bogenberger, K. (2019). Carsharing in Deutschland. *Journal für Mobilität und Verkehr*, 2019(02), 1–9.
- Sellaouti, A., Tiessler, M., Pobudzei, M., & Hoffmann, S. (2024). Evolution and characteristics of shared e-scooters usage in Munich, Germany – Results of an over 8 million trips data analysis. *Transportation Research Procedia*, 78, 40–46. <https://doi.org/10.1016/j.trpro.2024.02.006>
- Sharing-Vergleich. (2024). Geschäftsgebiete aller Sharing-Anbieter in Berlin. <http://www.sharing-vergleich.de/> (accessed on 15.12.2024)
- Simonetto, A., Monteil, J., & Gambella, C. (2019). Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, 101, 208–232. <https://doi.org/10.1016/j.trc.2019.01.019>
- Soteropoulos, A., Berger, M., & Ciari, F. (2019). Impacts of automated vehicles on travel behaviour and land use: An international review of modelling studies. *Transport Reviews*, 39(1), 29–49. <https://doi.org/10.1080/01441647.2018.1523253>
- SSB Flex. (2024). SSB Stuttgart Ride-Pooling. <https://www.ssb-ag.de/tickets/ticketkauf-und-beratung/ssb-flex/> (accessed on 15.12.2024)

- Stueger, P. N., Fehn, F., & Bogenberger, K. (2022). Minimizing the Effects of Urban Mobility-on-Demand Pick-Up and Drop-Off Stops: A Microscopic Simulation Approach. *Transportation Research Record: Journal of the Transportation Research Board*, 2677(1), 814–828. <https://doi.org/10.1177/03611981221101894>
- Tang, Q., & Armellini, M. G. (2021). An ant colony algorithm with penalties for the dial-a-ride problem with time windows and capacity restriction. 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Heraklion, Greece. <https://doi.org/10.1109/MT-ITS49943.2021.9529319>
- Thomsen, N. (2023). Implementing a Ride-sharing Algorithm in the German National Transport Model (DEMO). *Transportation Research Record: Journal of the Transportation Research Board*, 2677(5), 1–10. <https://doi.org/10.1177/03611981221127015>
- Tiessler, M., Pobudzei, M., Sellaouti, A., & Hoffmann, S. (2023). Electric Scooters and Where to Find Them—A Spatial Analysis of the Utilization of Shared E-Scooter in Munich, Germany. *Transportation Research Board (TRB) Conference*, Washington D.C., USA.
- Uber. (2024). Uber Transport Technologies Inc. <https://www.uber.com/> (accessed on 15.12.2024)
- UniBw Careers. (2024). University of the Bundeswehr Munich, Careers Homepage. <https://www.unibw.de/karriere> (accessed on 15.12.2024)
- von Mörner, M. (2018). *Sammelverkehr mit autonomen Fahrzeugen im ländlichen Raum [Dissertation]*. Technical University of Darmstadt.
- Wilkes, G., Engelhardt, R., Briem, L., Dandl, F., Vortisch, P., Bogenberger, K., & Kagerbauer, M. (2021a). Self-Regulating Demand and Supply Equilibrium in Joint Simulation of Travel Demand and a Ride-Pooling Service. *Transportation Research Record: Journal of the Transportation Research Board*, 2675(8), 226–239. <https://doi.org/10.1177/0361198121997140>
- Wilkes, G., Hilgert, T., Briem, L., Kagerbauer, M., & Vortisch, P. (2021b). *Mikroskopische Abbildung von Mobility on Demand im Verkehrsnachfragemodell mobiTopp. Heureka - Optimierung in Verkehr und Transport 2021*, Stuttgart, Germany.
- Williams, H. P. (2013). *Model Building in Mathematical Programming*. John Wiley & Sons.
- Wilson, N., & Colvin, N. (1977). Computer control of the Rochester Dial-A-Ride System (Technical Report Nos. R77-31). Dept. of Civil Engineering, Massachusetts Institute of Technology, USA.
- Yen, J. Y. (1971). Finding the K-Shortest Loopless Paths in a Network. *Management Science*, 17(11), 712–716. <https://doi.org/10.1287/mnsc.17.11.712>
- Zhou, Z., & Roncoli, C. (2021). Computationally efficient dynamic assignment for on-demand ridesharing in congested networks. 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Heraklion, Greece. <https://doi.org/10.1109/MT-ITS49943.2021.9529283>
- Zubair, M., Bock, R., & Hoffmann, S. (2025). Optimizing Demand Responsive Transit: A Study on Fleet Size and Service Performance in Munich's Northern Suburbs. *Transportation Research Board (TRB) Conference*, Washington D.C., USA.

Zwick, F., Wilkes, G., Engelhardt, R., Axer, S., Dandl, F., Rewald, H., Kosterz, N., Fraedrich, E., Kagerbauer, M., & Axhausen, K. W. (2022). Mode choice and ride-pooling simulation: A comparison of mobiTopp, Fleetpy, and MATSim. *Procedia Computer Science*, 201, 608–613. <https://doi.org/10.1016/j.procs.2022.03.079>

List of Figures

Figure 1.1: Illustration of a scenario on a square grid network.	2
Figure 3.1: Components of an on-demand ride-hailing/ride-pooling system.	40
Figure 3.2: Spontaneous assignment versus batch assignment.	41
Figure 3.3: Illustration of an exemplary situation on a square grid network (spontaneous assignment).	47
Figure 3.4: Illustration of an exemplary situation on a square grid network (batch assignment).	52
Figure 3.5: List of the vehicle/request-set combinations with the associated costs per combination. ..	56
Figure 4.1: Location of the campus in Munich Metropolitan Area (OpenStreetMap and GoogleMaps).	58
Figure 4.2: Campus map of the University of the Bundeswehr Munich (OpenStreetMap).	59
Figure 4.3: Setting the virtual gates on the camera images in FLOW client.	61
Figure 4.4: Data flow diagram to import camera data into the simulation.	61
Figure 4.5: Interaction between the simulation, back-end server, and Python script.	63
Figure 4.6: Overview of the vehicle inputs and routes used in the simulation (OpenStreetMap).	64
Figure 4.7: Usage of priority rules for bottlenecks in the campus.	65
Figure 4.8: Steps to calculate the cost and node sequence of Vissim paths using Dijkstra's shortest route algorithm.	68
Figure 4.9: Comparison of two different node structures in Vissim.	69
Figure 4.10: Implementation of priority rules at shuttle stops.	70
Figure 4.11: Overview of the external Python code.	74
Figure 4.12: Cumulative speed distributions used at two spots in the simulation.	75
Figure 4.13: 16 regions in the campus and corresponding shuttle stops (OpenStreetMap).	81
Figure 4.14: Distribution of MORE Sharing trips.	81
Figure 4.15: Overview of the processes to generate a request pool.	82
Figure 4.16: Steps for building a multimodal microscopic traffic simulation with on-demand ride-pooling.	83
Figure 4.17: The workflow of the simulation utilized in this research.	84
Figure 5.1: Impact of operational parameters on service rate.	88
Figure 5.2: Impact of operational parameters on sharing rate.	89
Figure 5.3: Impact of operational parameters on mean waiting time.	90
Figure 5.4: Impact of operational parameters on mean detour time.	91
Figure 5.5: Impact of operational parameters on mean mileage.	92
Figure 5.6: Impact of operational parameters on mean occupancy.	93

Figure 5.7: Impact of operational parameters on mileage index.	94
Figure 5.8: Demand profile of batch assignment scenarios over the day.	95
Figure 5.9: Time-series of service rate and sharing rate in a simulation day (in 15-minutes aggregation).	96
Figure 5.10: Distance traveled by the shuttles in the entire simulation (cumulative).	97
Figure 5.11: Time-series of mean waiting time in a simulation day (in 15-minutes aggregation).	98
Figure 5.12: Time-series of mean detour time in a simulation day (in 15-minutes aggregation).	98
Figure 5.13: Box plot of waiting time and detour time of served requests (n = 935).	99
Figure 5.14: Time-series of mean occupancy in a simulation day (in 15-minutes aggregation).	100
Figure 5.15: Utilization of the on-demand fleet (in 5-minutes aggregation).	100
Figure 5.16: Impact of travel demand on service rate and sharing rate.	102
Figure 5.17: Impact of travel demand on mean waiting time and mean detour time.	102
Figure 5.18: Impact of travel demand on mean occupancy, mileage index, and empty mileage ratio.	103
Figure 5.19: Impact of travel demand on total mileage and mileage per served request.	103
Figure 5.20: Impact of weightings in the cost function on mean waiting time and mean detour time.	105
Figure 5.21: Impact of weightings in the cost function on total mileage and mileage per served request.	105
Figure 5.22: Impact of weightings in the cost function on mileage index and empty mileage ratio.	106
Figure 5.23: Comparison of ride-hailing and ride-pooling based on system KPIs.	107
Figure 5.24: Relative change in system KPIs from ride-hailing to ride-pooling.	108
Figure 5.25: Comparison of matching strategies in terms of service rate and sharing rate.	110
Figure 5.26: Comparison of matching strategies in terms of mean waiting time and mean detour time.	110
Figure 5.27: Comparison of matching strategies in terms of mean occupancy, mileage index, and empty mileage ratio.	111
Figure 5.28: Comparison of matching strategies in terms of total mileage and mileage per served request.	111

List of Tables

Table 2.1 Overview of ride-hailing and ride-pooling studies involving matching strategies.....	38
Table 2.2 Overview of ride-hailing and ride-pooling studies involving matching strategies (cont.)... ..	39
Table 3.1: Possible options for the exemplary situation (spontaneous assignment).	48
Table 4.1: Daily distribution of vehicle volumes at the main gate of the campus (incoming traffic).. ..	64
Table 4.2: New attributes for the vehicles in Vissim.	71
Table 4.3: New attributes for the passengers in Vissim.	72
Table 4.4: An example of faulty data (third row) in the geolocations data table.	79
Table 5.1: Determination of the required number of simulation runs.	86
Table 5.2: Operational parameters used in the scenarios.	87
Table 5.3: Comparison of system KPIs across three scenarios with varying batch period.	109

List of Abbreviations

ALT	Anruf-Linien-Taxi (public taxi on-call)
API	Application Programming Interface
AST	Anruf-Sammel-Taxi (shared taxi on-call)
COM	Component Object Model
CVRP	Capacitated Vehicle Routing Problem
DARP	Dial-a-Ride Problem
DRT	Demand Responsive Transit
DVD	Digital Video Disc
FMOD	Flexible Mobility On-demand
HCPPT	High-Coverage Point-to-Point Transit
ILP	Integer Linear Programming
KPI	Key Performance Indicator
MDR	Maximum Detour Ratio
MILP	Mixed Integer Linear Programming
MORE	Munich Mobility Research Campus
MTZ	Miller-Tucker-Zemlin
MWT	Maximum Waiting Time
OD	Origin-Destination
PBefG	Personenbeförderungsgesetz (passenger transport act)
PDP	Pick-up and Delivery Problem
PUDO	Pick-up & Drop-off
TNC	Transportation Network Company
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows