Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Optimizing Ridesharing with Transfers in Urban Areas

Adéla Kubíková

Supervisor: Ing. David Fiedler
Field of study: Open Informatics
Subfield: Artificial Intelligence and Computer Science
May 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kubíková Adéla**	Personal ID number:	**491924**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence and Computer Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Optimizing Ridesharing with Transfers in Urban Areas

Bachelor’s thesis title in Czech:

Optimalizace sdílených jízd s přestupem v městských oblastech

Guidelines:

Mobility-on-demand (MoD) system with ridesharing is a widely discussed mobility concept. However, when deploying the MoD system in an urban area, it requires a large number of vehicles to guarantee an acceptable availability. To tackle this issue, an MoD system with passenger transfers is proposed. In such system, travelers can switch the car, using two or more vehicles for completing their journey. The goal of this work is to analyze the state-of-the art algorithms for ridesharing with transfers and deliver a method that will provide either a performance, or a ridesharing efficiency improvement.
1) Study the literature about ridesharing in MoD systems. Focus on ridesharing with transfers.
2) Based on your research, implement a state-of-the-art method for ridesharing with transfers as a baseline and integrate it into the SiMoD simulation framework (github.com/aicenter/simod).
3) With the help of the demand filtration and generation tools provided by your supervisor, create a travel demand dataset with characteristics similar to the dataset on which the selected baseline method was evaluated.
4) Evaluate the performance and efficiency of the baseline method using the created dataset.
5) Using the insights gathered during your work, design a new/improved algorithm for ridesharing with transfers that will overcome the baseline either in ridesharing efficiency or in computational performance.
6) Implement the designed algorithm in the SiMoD framework.
7) Evaluate the proposed algorithm in SiMoD using the created travel demand dataset and analyze the results.

Bibliography / sources:

[1] S. Mitrović-Minić and G. Laporte, “The Pickup And Delivery Problem With Time Windows And Transshipment,” INFOR: Information Systems and Operational Research, vol. 44, no. 3, pp. 217–227, Aug. 2006, doi: 10.1080/03155986.2006.11732749.
[2] W. Herbawi and M. Weber, “Evolutionary Multiobjective Route Planning in Dynamic Multi-hop Ridesharing,” in Evolutionary Computation in Combinatorial Optimization, Berlin, Heidelberg, 2011, pp. 84–95. doi: 10.1007/978-3-642-20364-0_8.
[3] Yunfei Hou, X. Li, and C. Qiao, “TicTac: From transfer-incapable carpooling to transfer-allowed carpooling,” in 2012 IEEE Global Communications Conference (GLOBECOM), Dec. 2012, pp. 268–273. doi: 10.1109/GLOCOM.2012.6503124.
[4] B. Coltin and M. Veloso, “Ridesharing with passenger transfers,” in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sep. 2014, pp. 3278–3283. doi: 10.1109/IROS.2014.6943018.
[5] S. Ben Cheikh, C. Tahon, and S. Hammadi, “An evolutionary approach to solve the dynamic multihop ridematching problem,” SIMULATION, vol. 93, no. 1, pp. 3–19, Jan. 2017, doi: 10.1177/0037549716680025.
[6] J. Schönberger, “Scheduling constraints in dial-a-ride problems with transfers: a metaheuristic approach incorporating a cross-route scheduling procedure with postponement opportunities,” Public Transp, vol. 9, no. 1, pp. 243–272, Jul. 2017, doi: 10.1007/s12469-016-0139-6.
[7] S. Lotfi, K. Abdelghany, and H. Hashemi, “Modeling Framework and Decomposition Scheme for On-Demand Mobility Services with Ridesharing and Transfer,” Computer-Aided Civil and Infrastructure Engineering, vol. 34, no. 1, pp. 21–37, 2019, doi: 10.1111/mice.12366.

Name and workplace of bachelor’s thesis supervisor:

Ing. David Fiedler   Artificial Intelligence Center FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **06.01.2022**   Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. David Fiedler	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

First and foremost, I would like to sincerely thank Ing. David Fiedler, for his guidance and feedback throughout the project, his valuable and substantive advice and comments, and his helpfulness during consultations.

I would also like to thank my family and friends for their support throughout the study and the Czech Technical University in Prague for the education provided.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

Prague, 20 May 2022

Abstract

The problem of increasing transport demand in cities, rising costs of car operation, pollution, and insufficient capacity of the road network can be solved by ridesharing - a service that allows sharing rides in a vehicle with other people. The implementation of transfers, i.e., the segmenting of a passenger's trip between multiple vehicles, promises to further increase transport efficiency. This topic is particularly applicable in the commercial sector for taxi services such as Liftago, Bolt, or Uber.

We review previous research and already proposed methods to solve ridesharing with transfers. We propose and implement a method for dispatching vehicles to requests using Insertion Heuristics, allowing at most one transfer at predefined transfer stations. We evaluate this method on real data from New York City and compare it to conventional ridesharing without transfers. Our method achieved a 17.9% improvement in the number of dropped demands and reduced average delay by 13.7 %.

Keywords: ridesharing, ridesharing with transfers, heuristics, SiMoD

Supervisor: Ing. David Fiedler
E-323,
Department of Computer Science,
Czech Technical University in Prague,
Karlovo náměstí 13,
121 35 Praha 2

Abstrakt

Problém rostoucí poptávky po dopravě ve městech, rostoucích nákladů na provoz automobilů, znečištění životního prostředí a nedostatečné kapacity silniční sítě lze řešit pomocí ridesharingu - služby, která umožňuje sdílet jízdu ve vozidle s dalšími lidmi. Dalšího zvýšení efektivity přepravy lze dosáhnout zavedením přestupů, tj. rozdělením cesty cestujícího mezi více vozidel. Toto téma se v komerčním sektoru týká zejména taxislužeb, jako jsou Liftago, Bolt nebo Uber.

Sumarizujeme přehled o předchozím výzkumu a již navržených metodách pro řešení sdílení jízd s přestupy. Navrhujeme a implementujeme metodu pro dispečink vozidel k cestujícím s využitím Insertion heuristiky umožňující maximálně jeden přestup na předem definovaných přestupních stanicích. Tuto metodu vyhodnocujeme na reálných datech z New Yorku a porovnáváme ji s konvenčním sdílením jízd bez přestupů. Naše metoda dosáhla 17,9 % zlepšení počtu odmítnutých požadavků a snížila průměrné zpoždění o 13,7 %.

Klíčová slova: sdílené jízdy, sdílené jízdy s přestupy, heuristiky, SiMoD

Překlad názvu: Optimalizace sdílených jízd s přestupem v městských oblastech

Contents

Figures

Tables

Chapter 1
Introduction

Due to the ever-increasing demand for urban transport, the number of vehicles is growing, and transport is facing significant problems, such as traffic jams, inaccessibility of parking spaces, and high pollution. Commuting by car is particularly uneconomical, given that most car seats are usually empty. In addition, rising petrol prices and accelerating environmental problems are making cars increasingly expensive to own and run. More and more people, especially in big cities, are using alternative modes of transport, commuting by bicycles, electric scooters, public transport, or taxis. But this has its problems too - in big cities, for example, it is increasingly difficult to call a taxi due to their busy schedule [1]. Moreover, taxi fares are also quite expensive, and it is costly to take a taxi every day. Because of the large number of cars, traffic jams often form. Cars also pollute the environment - up to 95 % of smog in cities is caused by traffic [2]. A possible solution that fights the above problems is sharing a route between multiple passengers in one vehicle.

The problem of sharing a route between passengers is solved by ridesharing. Ridesharing is a concept that essentially allows multiple people to use one vehicle at the same time. Therefore, multiple passengers can ride in a vehicle and share part or even all of the journey. The vehicle can change its current route plan if another passenger is served. While these changes may cause detours and delays for other passengers, the added inconvenience is usually balanced by a lower fare. For this reason, ridesharing can ultimately be beneficial to all passengers. Ridesharing also reduces the number of vehicles needed to service requests from passengers. Fewer vehicles also mean less congestion, higher speeds, and less air pollution.

In this work, we aim to improve the efficiency of transport even further by using a relatively new idea of ridesharing with transfers. In this model, passengers are allowed to transfer between vehicles, allowing taxis to serve a greater number of requests cooperatively. This scheduling problem is particularly challenging as the number of possible assignments of passengers to vehicles increases when transfers are allowed. However, available work

[1]Brits using Uber and other taxi apps face long waits and fare hikes amid driver shortage `https://www.cnbc.com/2021/09/27/uber-and-bolt-struggle-to-meet-demand-in-the-uk.html`

[2]Města se nebrání smogu z aut. Přitom jim to legislativa umožňuje `https://www.lidovky.cz/domov/mesta-se-nebrani-smogu-z-aut.A170409_150829_ln_domov_ELE`

suggests that ridesharing with transfers increases the efficiency of urban transport, increases the transport capacity, reduces the size of the necessary fleet, and thus reduces the impact on the environment and air pollution.

1.1 Project Target

This project aims to study various methods for solving Mobility-on-Demand systems with ridesharing with the possibility of transferring from one vehicle to another. Many different approaches, such as evolutionary algorithms, linear programming, or heuristic algorithms, have been proposed to solve this problem.

Our second goal is to choose a method that can provide a high-quality solution for sufficiently large instances in a reasonable amount of time. We then implement the selected method in the SiMoD simulation tool and evaluate its performance and efficiency.

In the following part, we propose a new algorithm that will bring improvements in either performance or efficiency. We again integrate the proposed algorithm into the SiMoD simulation tool and evaluate its results.

Finally, we compare the two implemented transfer methods with a conventional solution of ridesharing without transfers. We discuss the results and the comparison.

Chapter 2

Literature Review

Ridesharing is a riveting topic that has received attention from many researchers. However, considering transfers between vehicles is a relatively modern idea. Thanks to ridesharing, taxi services are able to carry more passengers at lower fares. Extending this idea by introducing transfers promises to further increase transport efficiency, allow lower fares, and potentially reduce mileage.

2.1 Ridesharing

Ridesharing can be operated in one of two modes. In static mode, all transport requests are known in advance, allowing all vehicle routes to be planned in advance. In dynamic mode, the requests are revealed throughout the day without prior knowledge, so algorithms need to match trip requests with the available vehicles on-the-fly and routes are built in real-time. We are considering two ridesharing services - Peer-to-peer ridesharing, where drivers share their personal trips, and centralized, which we can think of as a taxi service whose drivers communicate with each other.

Peer-to-peer ridesharing is a service that provides a platform for drivers to share their personal trips with riders who have similar itineraries.

Paper [20] summarizes in detail the existing P2P ridesharing literature. It focuses on modeling and solution methodologies for matching, routing, and scheduling problems. In conclusion, the authors discuss directions for future research, including autonomous vehicles. They consider ridesharing as a suitable platform for hosting autonomous electrified vehicles.

In [22], a similar approach as in [2] is used. Authors decompose the complicated problem of ridesharing matching into simpler parts which they solve separately. One of the unique contributions of [22] is to integrate the dynamic tree algorithm for solving ridesharing VRP. The dynamic tree allows using previously calculated driver schedules instead of calculating them from scratch.

Tafreshian and Masoud focus on the graph partitioning method in their paper [19]. To solve dynamic ridesharing, they present here the ε-uniform partitioning algorithm and show experimentally that the vehicle traveled distance savings produced by the trip-based ε-uniform algorithm follow the optimal results very closely.

Paper [21] addresses the problem of the small number of existing platforms for facilitating peer-to-peer ridesharing in a dynamic scenario that are integrated with multi-modal trip planners. It therefore presents the Xhare-a-Ride (XAR) system, a scalable platform for dynamic peer-to-peer ridesharing suitable for integration with a MMTP.

Centralized ridesharing solutions match passengers and drivers through a unified communication medium. It is particularly applicable in large cities for transport companies such as Liftago, Bolt, or Uber.

Various ridesharing systems are thoroughly surveyed in [1]. This paper reports that there is a growing interest from the research community to address the optimization issues in dynamic ridesharing, but the number of specific contributions is still small. In particular, it cites three areas for future research: fast optimization approaches for real-life instance sizes, incentive schemes to build critical mass, and optimization approaches that allow choice.

In paper [13], a real-time taxi sharing system based on a mobile cloud app has been proposed and developed. The taxi search process is simplified by dividing the road network into square networks. Experimental results have shown that it is possible to increase the capability of taxi services in a city to satisfy the commuting needs of more people. A decrease in the distance traveled was also achieved.

Other works propose heuristic algorithms. Paper [7] describes a tabu search heuristic for static DARP with multiple vehicles. Another solution is described in [16]. This paper presents several heuristics that combine greedy function, insertion heuristics, and local search. In [9], besides the solution using Insertion Heuristics, an evolutionary approach using genetic algorithms is also described.

Genetic algorithms are also provided in [8]. In this work, the feasibility of implementing MOEAs for solving the route planning problem in Dynamic Multihop Ridesharing Systems is investigated, and the deterministic Generalized Label Correcting algorithm and the scalable algorithm Nondominated Sorting Genetic Algorithm NSGA-II are presented here.

2.2 Ridesharing with Transfers

As with the standard ridesharing problem, ridesharing with transfers can be solved by many different methods.

Peer-to-peer ridesharing with transfers is examined in [14]. This work proposes a pre-processing procedure to reduce the size of the problem and devise a decomposition algorithm to solve the original ride-matching problem to optimality by solving multiple minor problems. Moreover, the sub-problems in each iteration are independent of each other, allowing the computations to be performed in parallel.

Article [4] deals with the problem of ridesharing for business trips in a closed community of companies. It first provides a general ILP formulation for the problem of ridesharing with meeting places. The model includes the possibility for a rider to transfer between drivers. Next, a greedy heuristic is proposed to solve larger instances.

The dynamic system, which generates a solution with an arbitrary number of transfers, is solved in [3]. In this paper, a novel approach, called MACGeO, is developed to address the multihop ride-matching problem. The originality of this approach lies in the fact that the coding of the chromosomes is dynamic and in the absence of a repair process for the crossover and mutation operators, which makes this approach different from other works using evolutionary algorithms.

Paper [15] focused on exact solution methods and presents the MIP formulation. For application to real scenarios, they consider a combination with branch-and-cut and branch-and-bound approaches.

A distributed model-free algorithm is served by [18]. This multihop ridesharing algorithm uses deep reinforcement learning to learn optimal vehicle dispatch and matching decisions by interacting with the external environment.

In [5], the real-time transfer mechanism is implemented in the environment with connected and self-driving vehicles. The developed scheme further increases the efficiency of ridesharing services and reduces the trip time of carpooled passengers.

Article [12] presents a modeling framework for MoD systems in metropolitan areas. The solution methodology uses a modified version of the GC algorithm and implements iterative decomposition techniques. They also consider possible parallelization by solving sub-problems independently.

Work [17] introduces genetic search-based memetic metaheuristic approach. This search algorithm is enhanced by a schedule-building procedure that postpones waiting times at selected locations if necessary.

Article [10] deals with the new problem of ridesharing with allowed transfers with the aim of maximizing the successful ridesharing ratio. A Driver Experience Considered Strategy and a Passenger Experience Considered Strategy for scheduling carpooling are proposed. This paper also concludes that allowing more than one transfer during a trip does not bring any benefits.

Heuristic solutions are dealt with in paper [6]. It proposes three solution methods, a greedy approach, an auction approach, and a graph search approach. It is shown that for some problems, transferring passengers reduces the distance traveled by almost 30%.

System for electric taxis is proposed in [11]. The new ridesharing scheme introduced herein considers both the limited battery of electric vehicles and the user requirements. The proposed TASeT problem is solved by a greedy heuristic algorithm. A taxi service that allows passengers to transfer at most once is considered.

As a criterion for selecting a method to implement, we considered centralized ridesharing with predefined transfer stations, and we also focused on the size of the instance it solves. To implement the baseline method, we chose the Greedy Heuristic described in [11]. This method solves centralized ridesharing, which can be applied in the taxi sector companies. We modify the presented model and simplify its conditions to define the problem for general vehicles. The algorithm presented here solves instances of thousands of vehicles, which is more than most of the work published on this topic.

Chapter 3

Problem Description

Ridesharing with transfers is defined as follows. We consider ridesharing planning by a central dispatch center. The dispatch center has the information about the current status of all the taxis (including current locations, route itineraries, passengers on board, etc.) and requests from passengers that need to be served. As stated previously, we consider the model in which no transfer or only one transfer from one taxi to another is possible. The transfer is possible on predefined transfer stations. All passengers must reach their destination within their tolerable delay time. The dispatch center tries to serve as many demands as possible in a given time with a limited number of taxis. In this section, we present the modified formulation.

3.1 Problem Formulation

The optimal solution of the problem can be found with Mixed-Integer Programming (MIP). We build a directed graph with additional data on its nodes and edges.

Let $G(N, E)$ be a directed graph with node set N and set of edges E. An edge from node i to node j is denoted by $(i, j) \in E$. We use $T \subseteq N$ to denote the set of transfer nodes in G. Let K be the set of taxis, the status of each taxi $k \in K$ at the time of dispatch is defined as a tuple (u_k, o_k) in which u_k denotes the seating capacity of vehicle k, and $o_k \in O$ denotes the start location (i.e., the initial location at the vehicle's of dispatch). Variable O is the set of starting locations of all taxis, $O \subseteq N$.

Let R be the set of passenger requests indexed by $r = 1, 2, ..., |R|$. Each request can be defined as a tuple (p_r, d_r), in which the first two parameters p_r and $d_r \in N$ are the pickup and dropoff locations of request $r \in R$, respectively. Let $D = \{d_r | \forall r \in R\}$ be the set of dropoff nodes and $P = \{p_r | \forall r \in R\}$ be the set of pickup nodes. The corresponding node is duplicated if two requests have a common pickup or dropoff location. In this model, each request is associated with exactly one pickup and dropoff pair. In graph $G, N = O \cup T \cup P \cup D$, each node in N is connected to all the other nodes, except that each node in O is only connected to all the nodes in P. In other words, a taxi may visit multiple pickup, transfer, or dropoff locations but not any start location of other taxis.

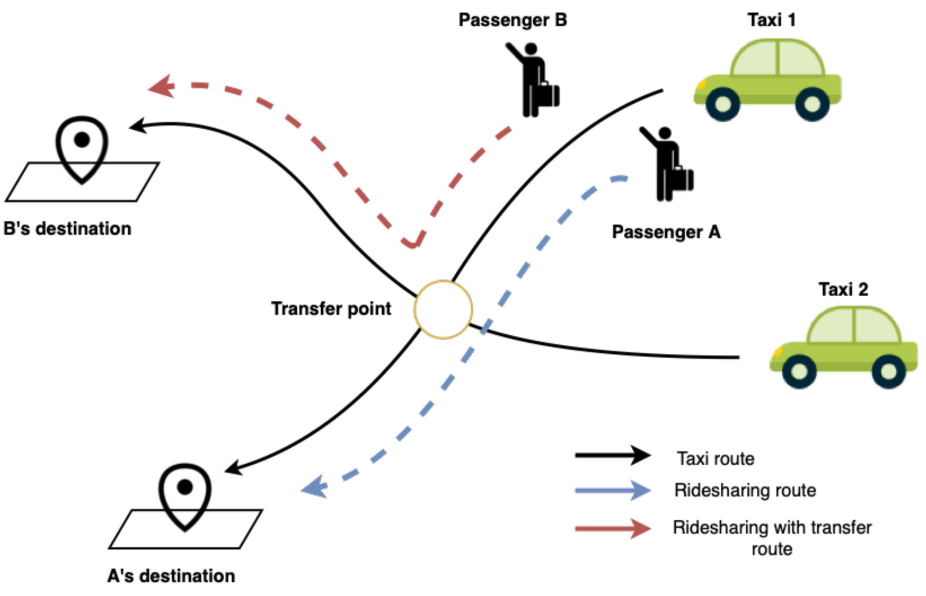


Figure 3.1: Example illustrating ridesharing with transfer problem

In this formulation, the {0, 1} variables represent the binary decision of taxi k (or request r) that uses edge (i, j). Specifically, we use two decision variables, $x^k_{(i,j)} = \{0, 1\}$ $k \in K$, $(i, j) \in E$; and $y^{kr}_{(i,j)} = \{0, 1\}$, $k \in K, r \in R, (i, j) \in E$. Let $x^k_{(i,j)} = 1$ if the taxi k uses edge (i, j) and $x^k_{(i,j)} = 0$ otherwise. Let $y^{kr}_{(i,j)} = 1$ if request r is served by taxi k on the edge (i, j), $y^{kr}_{(i,j)} = 0$ otherwise. The MIP model is stated as follows:

Maximize

$$\sum_{k\in K}\sum_{r\in R}\sum_{(i,j)\in E, j\in d_r} y^{kr}_{(i,j)} q_r \tag{3.1}$$

Subject to

$$\sum_{j:(i,j)\in E} x^k_{(i,j)} \leq 1 \quad \forall i = o_k \tag{3.2}$$

$$\sum_{j:(i,j)\in E} x^k_{(i,j)} - \sum_{j:(j,i)\in E} x^k_{(j,i)} \leq 0 \quad \forall k \in K, \quad \forall i \in T \cup D \tag{3.3}$$

$$\sum_{j:(i,j)\in E} x^k_{(i,j)} - \sum_{j:(j,i)\in E} x^k_{(j,i)} = 0 \quad \forall k \in K, \quad \forall i \in P \tag{3.4}$$

$$\sum_{k\in K}\sum_{j:(i,j)\in E} y^{kr}_{(i,j)} \leq 1 \quad \forall r \in R, \quad \forall i = p_r \tag{3.5}$$

$$\sum_{k\in K}\sum_{j:(p_r,j)\in E} y^{kr}_{(p_r,j)} - \sum_{k\in K}\sum_{j:(j,d_r)\in E} y^{kr}_{(j,d_r)} = 0 \quad \forall r \in R \tag{3.6}$$

$$\sum_{k\in K}\sum_{j:(i,j)\in E} y^{kr}_{(i,j)} - \sum_{k\in K}\sum_{j:(j,i)\in E} y^{kr}_{(j,i)} \leq 0 \quad \forall r \in R, \quad \forall i \in T \cup D \tag{3.7}$$

$$y^{kr}_{(i,j)} \leq x^{k}_{(i,j)} \quad \forall (i,j) \in E \quad \forall k \in K, \quad \forall r \in R \tag{3.8}$$

$$\sum_{r\in R} y^{kr}_{(i,j)} \leq u_k x^{k}_{(i,j)} \quad \forall (i,j) \in E, \quad \forall k \in K \tag{3.9}$$

The goal is to maximize the number of passengers served by the taxi fleet. Constraint (3.2) enforces that each taxi is scheduled at most once from its origin. Not all the taxis have to be dispatched if there are not too many requests. Constraints (3.3) and (3.4) maintain the flow conservation at transfer and dropoff locations. Constraints (3.5), (3.6) and (3.7) specify passenger routes - (3.5) enforces that each request is served at most once, (3.6) guarantees that passengers will reach their destination if they are picked up, (3.7) maintains the flow conservation at transfer nodes. Constraints (3.8) and (3.9) link the taxi and request flow. Constraint (3.8) states that if a request flow is on edge e, some taxi flows are on the same edge e. Constraint (3.9) enforces that each taxi would not carry more passengers than its seat capacity.

To capture the time constraints of the requests, a few additional constraints are defined. For an edge $(i,j) \in E$, let $t_{(i,j)}$ be the estimated travel time for a taxi from node i to j. Let $l_{(i,j)}$ be the distance between i and j. We use a^k_i and d^k_i to denote the arrival and departure times of taxi k at node i. Then, if a taxi k chose to travel on the edge (i,j), i.e., $x^k_{(i,j)} = 1$, it must satisfy $a^k_j \geq d^k_i + t_{(i,j)}$ and $d^k_j \geq a^k_j$ to handle the time sequence. Constraints (3.10) and (3.11) enforce these constraints by using the M constant (M is a large positive number).

$$d^k_i + t_{(i,j)} - a^k_j \leq M(1 - x^k_{(i,j)}) \quad \forall (i,j) \in E, \quad \forall k \in K \tag{3.10}$$

$$a^k_j \leq d^k_j \quad \forall j \in N, \quad \forall k \in K \tag{3.11}$$

Each request r has its pickup node p_r and dropoff node d_r. Two time windows $[s_{p_r}, e_{p_r}]$ and $[s_{d_r}, e_{d_r}]$ are associated with a request r. The pickup window defines maximum waiting time for a taxi arrival $t_w = e_{p_r} - s_{p_r}$. The dropoff window enforces that the total travel time of a passenger will not exceed maximum tolerable delay. Let t_{trip} be the travel time of the direct journey of a request served by a single vehicle without a transfer. Let t_d be the maximum additional trip time a passenger would accept. Start time and end time of the dropoff window can be expressed as $s_{d_r} = s_{p_r} + t_{trip}$ and $e_{d_r} = s_{d_r} + t_d$. The time constraints (3.12) and (3.13) are defined as follows:

$$s_{p_r} \leq a^k_{p_r},\ d^k_{p_r} \leq e_{p_r} \quad \forall k \in K, \quad \forall r \in R \tag{3.12}$$

$$s_{d_r} \leq a^k_{d_r},\ d^k_{d_r} \leq e_{d_r} \quad \forall k \in K, \quad \forall r \in R \tag{3.13}$$

To handle transfer, we use a logical counter c. Let $c^{kl}_{ir} = 1$ if the request r is transferred from taxi k to taxi l, $l \neq k$ at some transfer node $i \in T$, and $c^{kl}_{ir} = 0$ otherwise ($c^{kl}_{ir} \in \{0,1\}$ $\forall r \in R$, $\forall i \in T$, $\forall k,l \in K$, $k \neq l$). Constraints (3.14) and (3.15) together enforce that at a given point i a request r could transfer from taxi k to l only if taxi k arrives before the departure of taxi l. Constraint (3.16) allows passengers to transfer at most once during their trip.

$$a^k_i - d^l_i \leq M(1 - c^{kl}_{jr} \quad \forall r \in R \quad \forall i \in T \quad \forall k,l \in K,\ k \neq l \tag{3.14}$$

$$\sum_{j:ij \in E} y^{kr}_{ji} + \sum_{j:ji \in E} y^{lr}_{ij} \leq c^{kl}_{ir} + 1 \quad \forall r \in R \quad \forall i \in T \quad \forall k,l \in K,\ k \neq l \tag{3.15}$$

$$\sum_{i \in T} \sum_{l \in K,\, l \neq k} c^{kl}_{ir} \leq 1 \quad \forall r \in R \quad \forall k \in K \tag{3.16}$$

This provided MIP model is able to find the optimal solution. Due to its NP-hard complexity, it is only feasible for small instances, which makes it unsuitable to be directly applied to real-world scenarios. Therefore, heuristic strategies are often used in practice. Such strategies solve practical reasoning problems and can handle a large number of requests. In the following sections, we describe two heuristic methods that address ridesharing with transfers. In Chapter 4, we describe an algorithm using the Greedy Heuristic. In Chapter 5, we present a solution based on the Insertion Heuristic.

Chapter 4

Problem Solution - Baseline method

In this chapter, we describe the solution using the greedy heuristics, which is inspired from [11].

4.1 Greedy Heuristic Algorithm

The proposed heuristic finds all valid travel plans for a given passenger. It searches for both plans with a transfer and direct trips without a transfer. For each plan, it calculates its delay and transfer time and then uses these two values to select the final plan. In the following sections, we first describe the preparatory steps before the actual plan search, then describe the process of searching for valid plans, and then present the criteria used to select the final plan.

4.1.1 Preprocessing

Before the algorithm starts searching for possible schedules, it first calculates the arrival times of the vehicles at the transfer stations. The algorithm also looks for how many cars can potentially serve the requests and assigns priority to the requests accordingly.

In line 2 of Algorithm 1, the lookup table LT is maintained to keep track of potential taxis that can be arranged to transfer at the given transfer station. Entry $LT[t][k]$ stores the closest arrival time of taxi k from its current location to transfer station t. If there are already any passengers driving in taxi k, we must also check that setting t as the new transfer point does not exceed the tolerable delay for any passenger on board. If the drive to the transfer point t violates the maximum delay constraints, we set $LT[t][k] = \infty$. The values from LT are later used to find potential transfer stations and vehicles in lines 9 and 10.

In line 4, we rank requests in ascending order according to the number of vehicles that can pick up the given request in time. The request that has the least such vehicles is handled first. Requests with fewer available vehicles are less likely to be serviced, therefore we give them a higher priority.

4.1.2 Finding possible itineraries

Finding a trip itinerary is a two-step process. First, a direct route without a transfer is found in line 8. Lines 11-13 then search for an itinerary with a transfer. It is only possible to transfer at the predefined stations. When creating an itinerary with a transfer, we calculate the time for the transfer (i.e., the time from the arrival of the vehicle that arrives at station t first until the departure of the vehicle to which the passenger has transferred). If necessary, we will add an action for waiting at the station to the itineraries. We will check whether the new changes in the trip plan (i.e., the drive to the station t and potential waiting) do not conflict with the maximum passenger delays.

For one request, the algorithm may find multiple ridesharing itineraries. To build the final plan for the passenger, we will select one of the possible itineraries according to the rules described in the next section.

Algorithm 1 Greedy Heuristic

```
1:  function GreedyHeuristic(K, R)
2:      initialize $LT$
3:      $plans = \emptyset$
4:      $R' \leftarrow$ sort $R$ by the number of possible pickup taxis
5:      for each $r \in R$ do
6:          $templist = \emptyset$
7:          for each $k$ can pickup $r$ do
8:              $templist$ += FindPlan($r.src, r.dst, k$)
9:              for each $t \in$ potential transfer locations for $k$ do
10:                 for each $k'$ that $k$ can transfer to at $t$ do
11:                     $itnryp1 \leftarrow$ FindPlan($r.src, t, k$)
12:                     $itnryp2 \leftarrow$ FindPlan($t, r.dst, k'$)
13:                     $templist$ += CreateTransferPlan($itnryp1, itnryp2$)
14:                 end for
15:             end for
16:         end for
17:         sort $templist$ by delay and transfer time
18:         $selected \leftarrow$ itinerary with longest transfer time
19:                 in the top $\beta\%$ shortest delay
20:         update $k$, $k'$ and $LT$
21:         $plans$ += $selected$
22:     end for
23:     return $plans$
24: end function
```

4.1.3 Selecting the final plan

Two criteria apply to the selection of the plan.

- To select an itinerary with fewer detours, we only keep the top β % of plans with the shortest delay for the request. The variable β is a configurable parameter with default value $\beta = 20$.
- Among the remaining candidate itineraries, we choose the plan with the longest transfer time. This we want to ensure that passengers can comfortably connect to the next taxi. According to the original definition in [11], where the use of electrotaxis is considered, this time is also intended for vehicle charging.

The selected itinerary still has to meet all the constraints for passengers, i.e., the total delay still needs to be within the maximum tolerable delay for all participating passengers.

4.2 Feasible Rideshare Plan Finding

For a given request r (specified by its source and destination as in Algorithm 1 and a possible pickup taxi k, we calculate the feasible ridesharing plan as shown in Algorithm 2.

Algorithm 2 Find Feasible Rideshare Plan Without Transfer

```
1:  function FindPlan(k, r)
2:      $n \leftarrow$ number of existing passengers in $k$
3:      $lst = \emptyset$
4:      $pickuporder \leftarrow$ plan to pickup $r$ after picking up $n$ passengers
5:      $dropofforders \leftarrow$ permutations of $r$'s and $n$ dropoff actions
6:      for each $dropofforder$ do
7:          $lst$ += CreateItinerary($pickuporder + dropofforder, k$)
8:      end for
9:      return FindPlanMinDelay(lst)
10: end function
```

We use two heuristics based on practical and real-world considerations. First, we assume that the new passenger request r is the last one to board on taxi, meaning the pickup time will not change for n previously scheduled passengers. Thanks to this heuristic, we will have only one given pickup order. We get dropoff orders as permutations of all dropoff actions from passengers. If a taxi already has n passengers on board, there will be $(n+1)!$ possible dropoff orders.

This algorithm may not be correct if the existing plan contains transfer actions. If we were to further modify the vehicle plan, the passenger's arrival at the transfer station could be delayed. Then the continuity of the transfer might not be ensured. If the vehicle serving the first segment of the passenger's route is delayed and the vehicle serving the second segment arrives to pick up the passenger earlier, an error will occur. Alternatively, if the second vehicle has a wait action scheduled and the first vehicle is delayed, the waiting time will no longer be long enough, and the transfer will also not be continuous.

To avoid this possible error, we will freeze the part of the plan that contains the transfer actions. We split the existing plan into two parts - the first part will be the part of the plan from the beginning up to and including the last transfer action, and the second part will be the rest of the plan that follows. The `FindPlan` function will then operate only with the pickup and dropoff actions contained in the second part of the plan. The first part is used to calculate the plan duration, which is then added to the expected service time (t_e) of all actions in the second part of the plan in `FindPlanMinDelay` function.

Second, among all pickup and dropoff orders that satisfy time constraints for passenger delays, we select the one with the lowest increase in delay for passengers. Such a plan is selected according to Algorithm 3 below.

Algorithm 3 Find Optimal Order of Actions

```
function FindPlanMinDelay(lst)
    $delays \leftarrow \emptyset$
    for each $itinerary \in lst$ do
        initialize $delay = 0$
        for each $action \in itinerary$ do
            $dest \leftarrow$ destination point of $action$
            $t_e \leftarrow$ expected time of arrival to $dest$
            $t_{max} \leftarrow$ time of arrival to $dest$ with maximum tolerable delay
            if $t_e > t_{max}$ then
                $delay = 0$
                break;
            end if
            $delay$ += $t_{max}$ - $t_e$
        end for
        $delays$ += $delay$
    end for
    $bestPlan \leftarrow$ plan with maximum $delay$
    return $bestPlan$
end function
```

In Algorithm 3, we calculate the expected time of arrival for each action in the itinerary. We check whether the time respects the time constraints, i.e., it is smaller than the maximum tolerated delay. If the time does not meet these constraints, this plan is not feasible. For each itinerary, we add changes in passenger delays in every iteration. We set the delay change as the difference between the maximum arrival time and the expected arrival time. Then the plan with optimal order of actions is the one that is both feasible and has the biggest delay.

4.2.1 Finding Transfer Plans

When creating a transfer plan, the action of boarding the second car must follow the action of getting off the first car. For this reason, we call the `CreateTranferPlan` function, which ensures the sequence of these two actions.

The `CreateTranferPlan` function described in Algorithm 4 calculates the expected arrival times of both vehicles at the station. The difference between these two times determines the potential waiting time. If the first vehicle arrives later than the second vehicle, we add a wait action to the second vehicle's plan with the appropriate duration. We then check that the delay caused by the waiting does not violate the maximum tolerated time for the following actions in the plan. If the plan is valid, we return both of its segments. Otherwise, we return error value indicating that the transfer at the station is not possible for the vehicles due to time constraints.

Algorithm 4 Find Plan with Transfer

```
function CreateTransferPlan(itnryp1, itnryp2)
    $valid \leftarrow$ True
    $t_1 \leftarrow$ time of arrival of the first car to transfer station
    $t_2 \leftarrow$ time of arrival of the second car to transfer station
    $waitTime = t_1 - t_2$
    if $waitTime > 0$ then
        add Wait action to $itnryp2$
        $valid \leftarrow$ check constraints for $itnryp2$
    end if
    if $valid$ then
        return $itnryp1, itnryp2$
    end if
    return $null$
end function
```

Chapter 5

Problem Solution - Improved method

In the second part of this paper, we try to improve the efficiency of ridesharing and propose a new method that overcomes the results of the baseline method.

5.1 Main ideas of the proposed method

We identify the biggest weaknesses of the Greedy Heuristic algorithm. The biggest limitation is undoubtedly the restriction to a fixed order of pickup actions. The Greedy Heuristic algorithm assumes that all passengers first board in a fixed order (a new passenger for whom we are looking for a schedule always boards last), and only then searches for the best order of dropoff actions.

The second major limitation is the transfer planning itself. If the Greedy Heuristic algorithm schedules a transfer, it is no longer able to effectively modify the schedules of the vehicles participating in the transfer. The permutation of dropoff actions and the fixed order of pickups could disrupt the temporal sequence of actions of dropping off the first vehicle and picking up the second. Therefore, if the vehicle schedule contains a transfer action, we freeze the segment up to the transfer action and perform permutations only with the remaining part.

In designing the new algorithm, we therefore focused on removing the constraint of a fixed order of pickup actions. Creating permutations from all actions in the plan (i.e., not only dropoff actions but also pickup actions) is not possible due to its computational complexity. Therefore, we decided to use an insertion heuristic approach, i.e., insert new actions into the vehicle plan one by one in such a way that the total travel time of the plan is extended as little as possible.

For the new method, we have also introduced a way to safely handle transfer pickup and dropoff actions. When planning a transfer, we calculate the arrival at the station of both vehicles participating in the transfer. If the vehicle serving the first segment arrives later than the vehicle serving the second segment, we need to add a waiting action with a corresponding waiting time to the second vehicle's plan. In this case, we cannot delay the arrival of the first vehicle at the station any longer because then the waiting time would be insufficient, and the second vehicle would leave the station too early. However, if the second vehicle arrives later than the first vehicle, the first vehicle may still adjust the schedule and delay the passenger's arrival at the station. However, the new arrival time shall not exceed the arrival time of the second vehicle in order to maintain the transfer sequence. Therefore, we introduce a parameter t_{max} for vehicle pickup and dropoff actions that will store the maximum arrival time at the station for the first vehicle. We do not modify the waiting actions or their duration (waiting time) during further calculations.

5.2 Insertion Heuristic Algorithm

The newly designed algorithm does not operate with a fixed pickup order but uses Insertion Heuristics to create a vehicle plan. As in the Greedy Heuristic case, the algorithm searches plans in two phases, first looking for plans without a transfer and then looking for possible stations and plans for the transfer. The algorithm is dynamic, so the travel route for the passenger may change over time. The algorithm is described in Algorithm 5.

5.2.1 Preprocessing

As in the case of the previous algorithm, Insertion Heuristics also looks for the vehicles that can handle the requests before computing plans. According to the number of vehicles, the heuristics assigns priority of computation to the requests and then sorts requests by their priorities.

In line 2, we first initialize P, which is a map that stores vehicles and their plans. In line 3 we initialize the set S, which contains all transfer points.

In line 4, we initialize a map with possible pickup taxis for each request. As a vehicle that can serve the request, we consider a vehicle that is able to reach the pickup location of the request from the vehicle's current position in time, i.e., the arrival time is less than or equal to the maximum time of the pickup event. If the vehicle is not able to arrive from its current position in time, it cannot serve the plan with certainty.

Depending on the number of vehicles that can serve the request, we assign the request priority for the computation of the plans. We assume that requests that can be served by more vehicles are more likely to be successfully served. Therefore, we are first looking for plans for those requests for which there are the least possible pickup vehicles. We assign priority for the calculation of plans to individual requests by sorting the requests in ascending order according to the number of vehicles that can serve them.

5.2.2 Finding possible itineraries

Route itineraries are searched in two phases. First, we are looking for a route plan without a transfer in line 9. In the second phase, in lines 16-21 we are looking for plans with a transfer at predefined stations. For each plan

Algorithm 5 Insertion Heuristic Algorithm with Transfer

```
1: function InsertionHeuristic(R, K)
2:     $P = \emptyset$
3:     $S \leftarrow$ transfer stations
4:     $K' \leftarrow$ taxis that can pickup requests in time
5:     sort $R$ by the number of possible pickup taxis
6:     for each $r \in R$ do
7:         $p, delays, waits = \emptyset$
8:         for each $k_1 \in K'$ can pickup $r$ do
9:             $positnry \leftarrow$ FindItinerary($r.src, r.dst, k_1$)
10:            $t_d \leftarrow$ GetDropoffTimeOfRequest($positnry, k_1, r$)
11:            $delays$ += delay $d$ induced by $positnry$
12:            $p$ += ($positnry, k_1$)
13:            for each station $s \in$ S suitable for transfer do
14:                for each $k_2 \in K$ do
15:                    if $k_2$ can reach $s$ in time then
16:                        $itnryp1 \leftarrow$ FindItinerary($r.src, s, k_1$)
17:                        $t_s \leftarrow$ GetDropoffTimeOfRequest($itnryp1, K_1, r$)
18:                        $itnryp2 \leftarrow$ FindTransferItinerary($s, r.dst, k_2, t_s$)
19:                        $t_{transfer} \leftarrow$
20:                                CountMaxTimeTransfer($itnryp1, k_1, itnryp2, k_2, r$)
21:                        set $t_{max}$ of dropoff action at $s$ in $itnryp1$ to $t_{transfer}$
22:                        $delays$ += delay $d$ induced by $itnryp1, itnryp2$
23:                        $waits$ += difference of the arrival of $k_1$ and $k_2$ at $s$
24:                        $p$ += ($itnryp1, k_1$),($itnryp2, k_2$)
25:                    end if
26:                end for
27:            end for
28:        end for
29:        sort $p$ by $waits$
30:        $tmp \leftarrow$ take $\beta\%$ of sorted $p$
31:        $P$ += itinerary with smallest delay in $tmp$
32:    end for
33:    return $P$
34: end function
```

found, we calculate the delay for the request that is created by the plan, and we also store information about the waiting time of vehicles at the station. All possible route plans are stored in p, which is a map with vehicles and their itineraries for a single request r. The method of finding a route plan will be described in detail later.

The algorithm finds multiple solutions (itineraries), how the request r may be served. To select one plan for each request in R, we use the criteria described in the following section.

5.2.3 Final Plan Selection

We apply several criteria to select the final plan.

- The vehicle cannot operate other requests and causes a delay for other passengers on board while waiting in the station. Passengers who are waiting for the second car at the transfer station do not cause any delay for others by waiting. Let a_{k_1} be the time of arrival of the first vehicle $k_1 \in K'$ at the station $s \in S$ and let a_{k_2} be the time of arrival of the second vehicle $k_2 \in K$ at the station $s \in S$. We therefore prefer such plans, where waiting time t_{wait}, $t_{wait} = a_{k_1} - a_{k_2}$, is as small as possible. To do so, we only keep the top β% plans with the shortest t_{wait} for the request. The variable β is a configurable parameter with default value $\beta = 20$. Note that the plans without a transfer have a t_{wait} of zero.
- Among the remaining candidate itineraries, we choose the plan with the shortest delay. Thanks to this, we choose a plan that has fewer detours and thus allows the passenger to travel to its destination without unnecessarily much delay.

5.3 Finding a feasible route plan

In this section, we will describe in more detail how the route plans are obtained. First, we will focus on transfer-free plans. Then we will explain how to assemble plans with transfers.

5.3.1 Itinerary without transfer

Insertion Heuristic algorithm in the first phase searches the plan without transfer. In line 9 of the `InsertionHeuristic` algorithm, the best positions in the current plan of vehicle k_1 where the pickup and dropoff actions of r should be inserted are found. This is solved in the `FindItinerary` function described in Algorithm 6 below. Then we calculate the time of the dropoff of r by the found plan in line 10. This time is used to calculate the delay for request r. Let t_{min} be the minimal travel time for request r, i.e, the duration of a direct route from its pickup location to the dropoff. Let t_c be the current time. Then delay $d = t_d$ - t_c - t_{min}. Since the plan does not include a transfer, we consider the waiting time to be zero.

FindItinerary function

The function takes the current plan of vehicle k and initializes a list P, which is a list with all possible route plans of k to serve new demand given by its pickup and dropoff action.

We are placing new pickup and dropoff actions in lines 7 and 8. We will try all possibilities of placing new actions. To do this, we will use a nested for loop and iterate over indices i and j. Index i represents the index for placing the pickup action, and index j represents the index for placing the dropoff action. The pickup action can be placed into the plan anywhere. Denoted the length of the original plan as l, index $i \in <0, l+1>$. The dropoff action must occur after the pickup action, so index $j \in <$i+1, $l+2>$.

After placing new actions into the plan, we will check whether the plan is valid. This means that for all actions in the plan, their maximum time will not be exceeded, i.e., all will be served in time. We must also not exceed the maximum capacity of the vehicle at any time.

We will then select one of all valid possible plans. The selection criterion is the increase of the travel time of the vehicle k to execute its plan, i.e., the difference between the time required to fulfill the modified plan *tmp* and the time required to fulfill the original plan. We will choose the plan that has the smallest such difference, i.e., where adding new actions cause the shortest detour.

Algorithm 6 Find Itinerary without Transfer

```
1: function FindItinerary(pickup, dropoff, k)
2:     $tmp \leftarrow$ current plan of $k$
3:     $pos = \emptyset$
4:     $l \leftarrow$ the length of $tmp$
5:     for each index $i \in <0, l+1>$ do
6:         for each index $j \in <i+1, l+2>$ do
7:             $tmp$ += add $pickup$ action at index $i$
8:             $tmp$ += add $dropoff$ action at index $j$
9:             if $tmpPlan$ is valid then
10:                $poss$ += $tmp$
11:            end if
12:        end for
13:    end for
14:    $selected \leftarrow$ plan with lowest increase in travel time in $pos$
15:    return $selected$
16: end function
```

5.3.2 Itinerary with Transfer

The transfer itinerary is searched for in the second phase of the Insertion Heuristic algorithm in lines 13 - 24. We will first select suitable stations for the transfer according to the travel time distance. If the route from the station s to the request's r dropoff location takes longer than the maximum dropoff time for r, there is no point in calculating transfers at the station s. From all vehicles, we only calculate with those that are able to reach the station on time, i.e., the arrival time at the station from their current position is less or equal to the difference between the maximum dropoff time of r and the minimum travel time from station s to dropoff location of r.

We will divide the transfer plan into two individual parts. The first vehicle k_1 will serve the part of the route from the pickup location of r to the station s. The second vehicle k_2 will serve the second route segment from the station s to the dropoff location of r. The plan for the first segment served by k_1 vehicle is found by the `FindItinerary` method described above. To find the second part of the plan, we need to calculate the arrival time of the vehicle k_1 to the station s, stored in t_s variable, in line 17. In line 18, plan for a second vehicle k_2 is found. The algorithm of `FindTransferItinerary` method will be described in the following section later. When plans for both vehicles and both segments are set, we can calculate the maximum time for transfer $t_{transfer}$ for the dropoff action of r at the station s in the first segment. In line 21, we set the t_{max} property of the dropoff action of r at station s equaled to the maximum time of transfer $t_{transfer}$.

The t_{max} property of the dropoff at station action is useful for further modifications of the vehicle plan of k_1. We will use a simple rule to determine the $t_{transfer}$ variable. If the second vehicle k_2 has waiting for r scheduled, we can no longer delay the arrival of the first vehicle k_1 at the station s in order to maintain the continuity of the transfer. Thus, $t_{transfer}$ is equal to the dropoff time of r from k_1 in station s. However, if the second vehicle k_2 does not have a waiting scheduled (i.e., it arrives at the station s later than the first vehicle k_1), we can possibly delay the arrival of the first vehicle. To keep the transfer valid, we set $t_{transfer}$ equal to the arrival time of the second vehicle k_2 at the station s.

We will save the computed itineraries in the list. Also we store the calculated delay and waiting time. Let t_c be the current time, t_{min} be the minimal travel time for the direct route from the start location of r to the destination of r. And let t_e be the actual dropoff time of r from k_2 vehicle. Then delay $d = t_e$ - t_c - t_{min}. We then store the difference between the arrival times of the two vehicles k_1, k_2 at the station s as the waiting time.

FindTransferItinerary function

The `FindTransferItinerary` function is described in Algorithm 7. The function is identical to the `FindItinerary` function, except for lines 9 - 13. We will therefore describe only this part of the method in more detail.

In line 9 of `FindTransferItinerary` function, we store the time of arrival of vehicle k_2 to the pickup location at the transfer station. In other words, we calculate the arrival time of the vehicle that serves the second segment of the transfer plan of r to the transfer station, where the passenger is being picked up. Subsequently, we calculate the difference between the arrival times of the two vehicles involved in the transfer so that we can determine the waiting time in line 10. If the waiting time is less than zero, it means that the first vehicle will arrive before the second vehicle, and the passenger will wait at the station. If the waiting time is greater than zero, we will have to schedule waiting for a second car to assure the continuity of the transfer.

Also, for this plan we have to check whether it does not violate the time constraint for any of the passengers so that all actions in the plan will be served on time and also the maximum capacity of the vehicle will not be exceeded at any time.

Algorithm 7 Find Itinerary with Transfer

```
 1: function FindTransferItinerary(pickup, dropoff, $k_2$, $a_{k_1}$)
 2:     $tmp \leftarrow$ current plan of vehicle
 3:     $pos = \emptyset$
 4:     $l \leftarrow$ the length of tmp
 5:     for each index $i \in <0, l+1>$ do
 6:         for each index $j \in <i+1, l+2>$ do
 7:             tmp += add pickup action to index i
 8:             tmp += add dropoff action to index j
 9:             $t_2 \leftarrow$ arrival time of $k_2$ to pickup location
10:             $waitTime = a_{k_1} - t_2$
11:             if $waitTime > 0$ then
12:                 add waiting to tmp before pickup action
13:             end if
14:             if tmp is valid then
15:                 pos += tmp
16:             end if
17:         end for
18:     end for
19:     $selected \leftarrow$ plan with lowest increase in travel time of $k_2$
20:     return selected
21: end function
```

Chapter 6
Implementation

Both of the above methods of ridesharing with transfers described above have been implemented in the SiMoD tool. SiMoD[1] is a simulation tool for Mobility-on-Demand developed by the Smart Mobility group of AI Center, CTU, Prague. It is based on the AgentPolis[2] traffic simulation framework, which was also developed by the Smart Mobility group. It allows to create MoD systems in a location of the user's choice. It is lightweight, highly customizable, and can easily run simulations with tens of thousands of vehicles and passengers.

Both SiMoD and AgentPolis are written in Java 11. To solve dependencies and build the application, Apache Maven[3] technology is used. The project also requires Gurobi optimization tool [4] and Maven support for Gurobi.

As part of the implementation, we first implemented the solvers (Greedy Heuristic and Transfer Insertion Heuristic solver) themselves. We then verified them using unit tests. The second task was to integrate the new solvers into the simulation environment.

6.1 Implementing of solvers

The DARP solvers in SiMoD operate with vehicle plans. The plans contain pickup and dropoff actions that have a specific location, information about the passenger to whom the action belongs, and the maximum time in which the action must be completed. As SiMoD did not yet operate with transfers and only allowed conventional ridesharing, several changes had to be made.

First, we had to find a way to distinguish a passenger pickup from a pickup at the transfer station (and the same for the dropoff). We therefore introduced new transfer actions (`PlanActionPickupTransfer.java`, `PlanActionDropoffTransfer.java`), which have the same hierarchy as the original pickup and dropoff actions. Next, we have also added a wait action (`PlanActionWait.java`) that includes extra information about the length of

[1]SiMoD simulation tool `https://github.com/aicenter/simod`; this thesis is available on the branch greedyTASeT `https://github.com/aicenter/simod/tree/greedyTASeT`

[2]Agentpolis framework `https://github.com/aicenter/agentpolis`

[3]Apache Maven `https://maven.apache.org`

[4]Gurobi Optimizer `https://www.gurobi.com/products/gurobi-optimizer/`

the waiting. The implementation of these three new classes was enough to implement both solvers and unit tests.

After implementing the solvers, we debugged their functionality using unit tests. Subsequently, we started to integrate them into the SiMoD tool.

6.2 Integration to simulation tool

The next task was to integrate the solvers into the simulation environment. To move vehicles in the SiMoD environment, various event activities are used. Different actions in the vehicle plans, when processed, will trigger different event activities for the vehicle agent. Here again, changes had to be made to allow transfers because the existing agent action handling system was not able to handle the newly introduced transfer actions and waiting at the stations.

The existing agent action handling system works as follows. When a vehicle plan is executed, the vehicle will start moving to the location of the current action in the plan. In order for the vehicle to start moving, a route must first be established. For this, the `createTrip` method in the `TripsUtil.java` class is used. This method finds the shortest route between the two defined locations. The trip is then defined by a list of all nodes in the graph that the vehicle passes through. The moment it reaches the designated location, it immediately performs the appropriate action (picking up or dropping off a passenger), creates a new route, and smoothly starts driving to the next action. If it completes the plan and has no further action in it, it will head to the nearest station (depot) where it will wait until its plan changes.

To enable transfers for passengers, we modified existing classes and introduced new methods for exiting and boarding vehicles at the transfer stations in `DemandAgent.java`. For the vehicle (`RidesharingOnDemandVehicle.java`), we had to introduce the handling of new transfer actions. For driving to a station (in order to board or exit a passenger), it was sufficient to use existing modified methods. But for waiting, we had to invent a completely new method and create new event activities.

For a vehicle to wait in a station, it is necessary to first initiate a driving to the station and then initiate the waiting. The new waiting activity therefore consists of two event activities - driving to the station and a subsequent waiting with the required duration. We implemented the waiting in classes `WaitWithStopActivityFactory.java` and `WaitWithStop.java`. The waiting can be interrupted when the plan is changed, in which case the elapsed time is stored in the `WaitWithStop` activity, and the elapsed time is subtracted when the activity is resumed.

In order to consider transfer points, it was also necessary to implement an initializer for the transfer stations (class `TransferPointsInitializer.java`). With the new initializer, the stations are defined in a text file from which the stations are then loaded into SiMoD. A similar method is used to load depots and vehicles.

In addition to the changes forced by the implementation of the new solvers, we also made several modifications related to statistics. We have introduced new statistics for counting the number of transfers and storing waiting times. We also added statistics on vehicle activity and inactivity over time. All the changes described above and a simplified project class diagram are described in Figure 6.1.

6.3 Description of the simulation environment

The running simulation is shown in Figure 6.2. Requests are marked with a red dot, and vehicles are marked with a blue or green triangle. If the vehicle carries more than one passenger, the number of carried passengers is indicated by a number above the vehicle. When clicked on a vehicle, its route plan will be displayed. In the picture, we can see the plan of the vehicle serving request `191` and its pickup and dropoff actions. At station `19` it is possible to notice requests awaiting the arrival of a vehicle serving the second part of the transfer plan. The green color of vehicles indicates empty vehicles that are performing rebalancing, i.e., driving to another station to balance the number of vehicles in the stations.

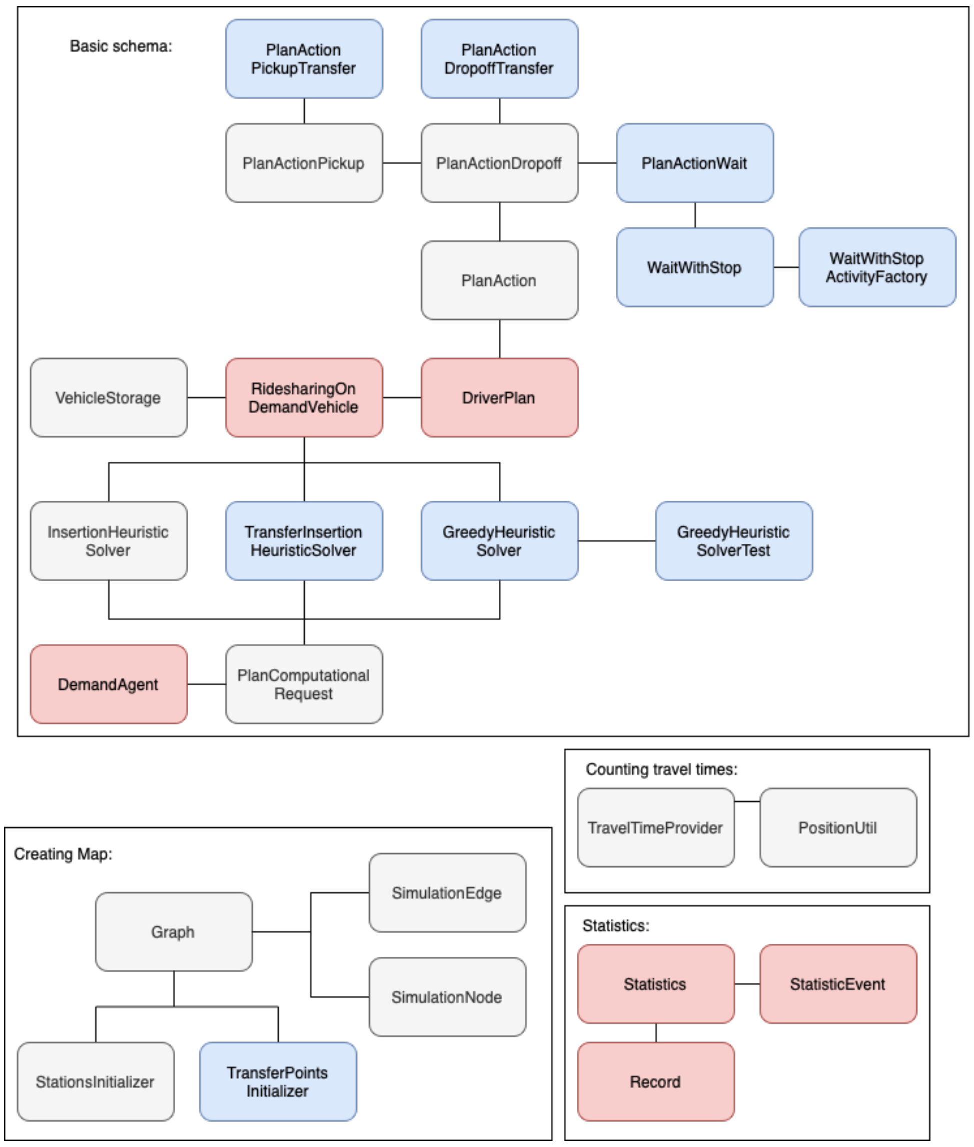


Figure 6.1: Simplified diagram of solvers and classes that solver uses. Newly implemented classes are colored blue. Edited classes are colored red.

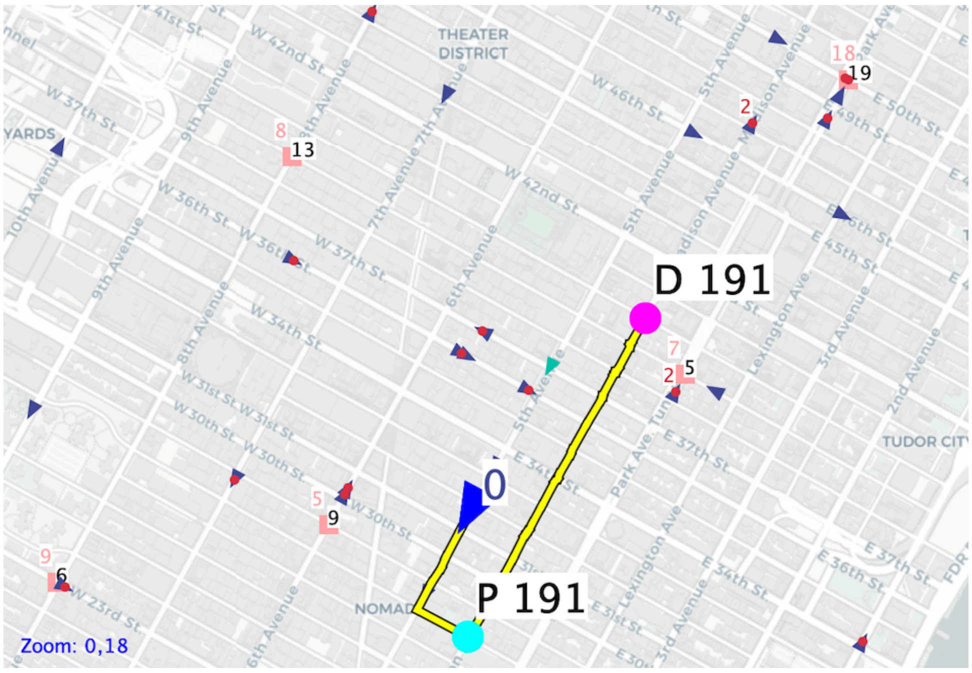


Figure 6.2: Running simulation in SiMoD

Chapter 7

Evaluation

In this section, we first describe the data used for the evaluation of the compared methods. We then compare the performance of the methods described in Chapter 4 and Chapter 5 and compare them to the conventional ridesharing solution without transfers (solved using the Insertion Heuristic). We first describe the summary results and then look at selected statistics in more detail.

7.1 Dataset description

In this section, we describe the different components of the dataset. It consists of a graph, a fleet of vehicles, transfer stations, and passenger demands for transport.

7.1.1 Road Graph

The graph that represents the map in the simulation was created from OpenStreetMap data and is shown in Figure 7.1. It encloses an area with latitude from N40.6923° to N40.8305° and longitude from W74.0652° to W73.8603° and is made up of 13,510 nodes (located at road intersections) and 31,357 edges.

7.1.2 Demand data

The New York City Taxi and Limousine Commission provides open access to trip data from New York City taxi providers on NYC OpenData website [1]. We took the 2014 dataset for evaluation. The dataset contains 165,114,361 records, which corresponds to an average of 452,000 taxi rides per day. For the evaluation, we took one typical weekday. By filtering, we selected only trips from Manhattan to reduce the area for evaluation. The demand locations are shown in Section 7.1.2. Further, we focused only on the 8:00 a.m. to 1:00 p.m. time window and randomly selected the number of requests determined by

[1]Datasets from New York City Taxi and Limousine Commission `https://data.cityofnewyork.us/browse/select_dataset?Dataset-Information_Agency=Taxi+and+Limousine+Commission+%28TLC%29`

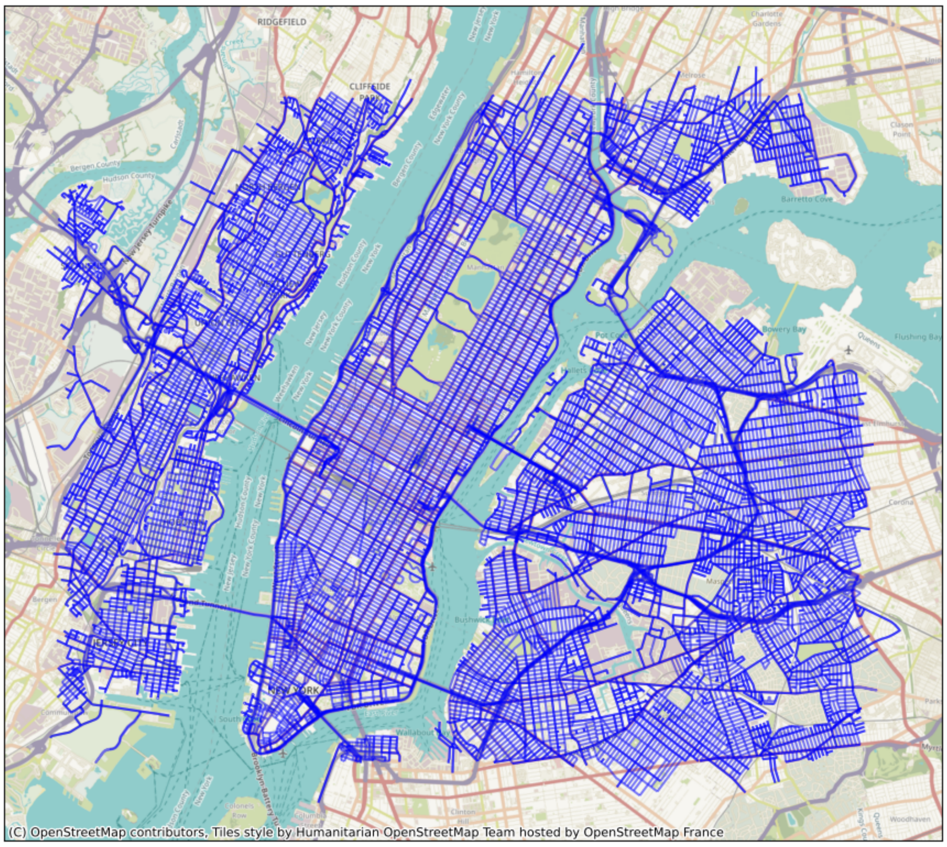

(C) OpenStreetMap contributors, Tiles style by Humanitarian OpenStreetMap Team hosted by OpenStreetMap France

Figure 7.1: Road graph used in simulation

Table 7.2 in each hour window. A histogram of selected demands over time is shown in Figure 7.3.

In total, we selected 3970 demands. Of these, 43 were discarded for further calculation because they appeared to have zero length after they were processed for SiMoD. The statistics on the distance of the origin and destination of the demands are described in Table 7.1.

Demands total	3970
Average trip length *[km]*	6.200
Longest trip *[km]*	45.266
Shortest trip *[m]*	82.27
Fleet size	300

Table 7.1: Demands description

8-9 am	*9-10 am*	*10-11 am*	*11-12 pm*	*12-1 pm*
640	840	920	820	750

Table 7.2: Number of demands by hour

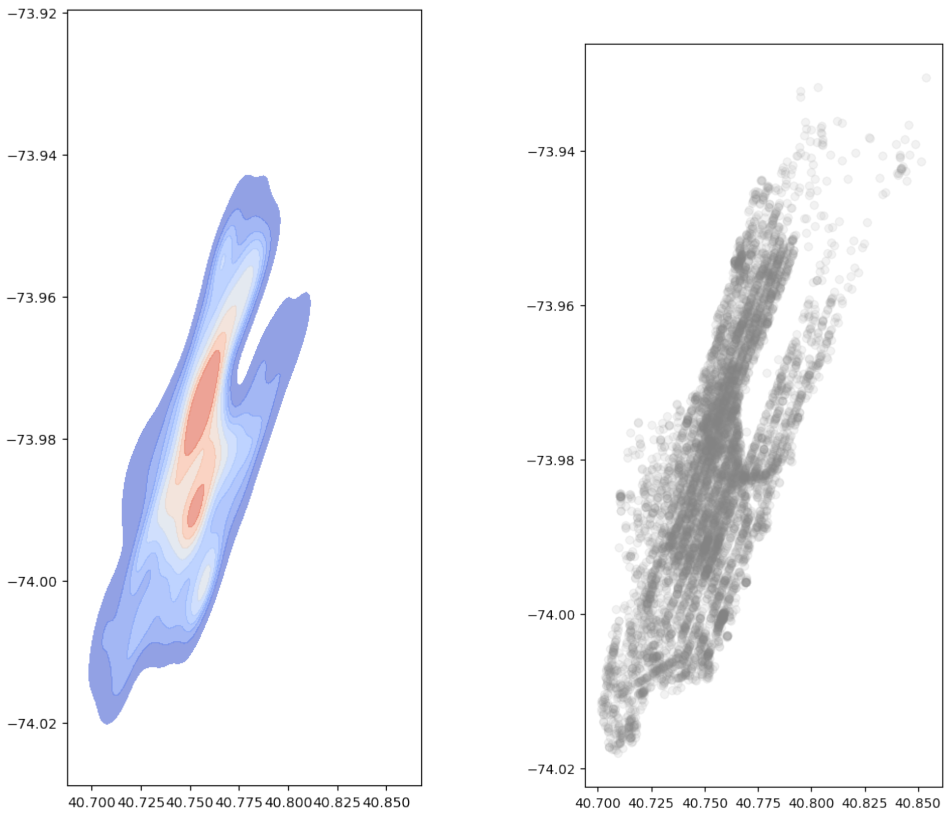


Figure 7.2: Demand heatmap and pickup and dropoff locations

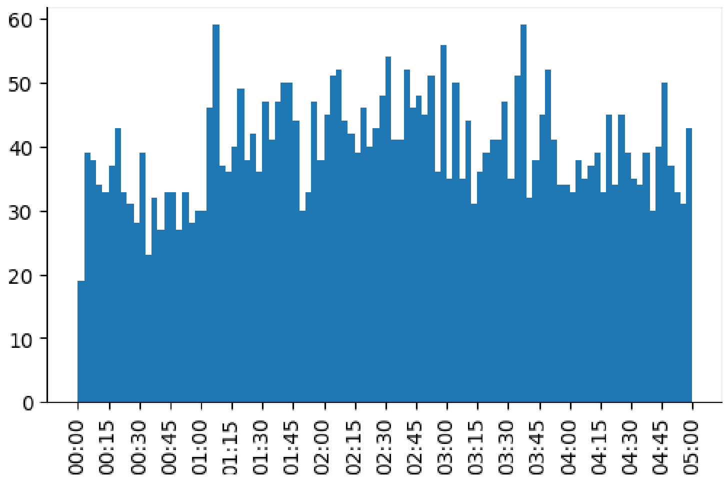


Figure 7.3: Number of demands in time [s]

7.1.3 Vehicle Fleet

We have chosen a fleet of 300 vehicles to serve the demand. The vehicles have a maximum capacity of 5 passengers. We have approximately evenly spaced 20 stations around the Manhattan area that serve as both vehicle depots and transfer points. At the start of the simulation, there is an equal number of 15 vehicles at each station. The locations of the stations can be seen in Figure 7.4.

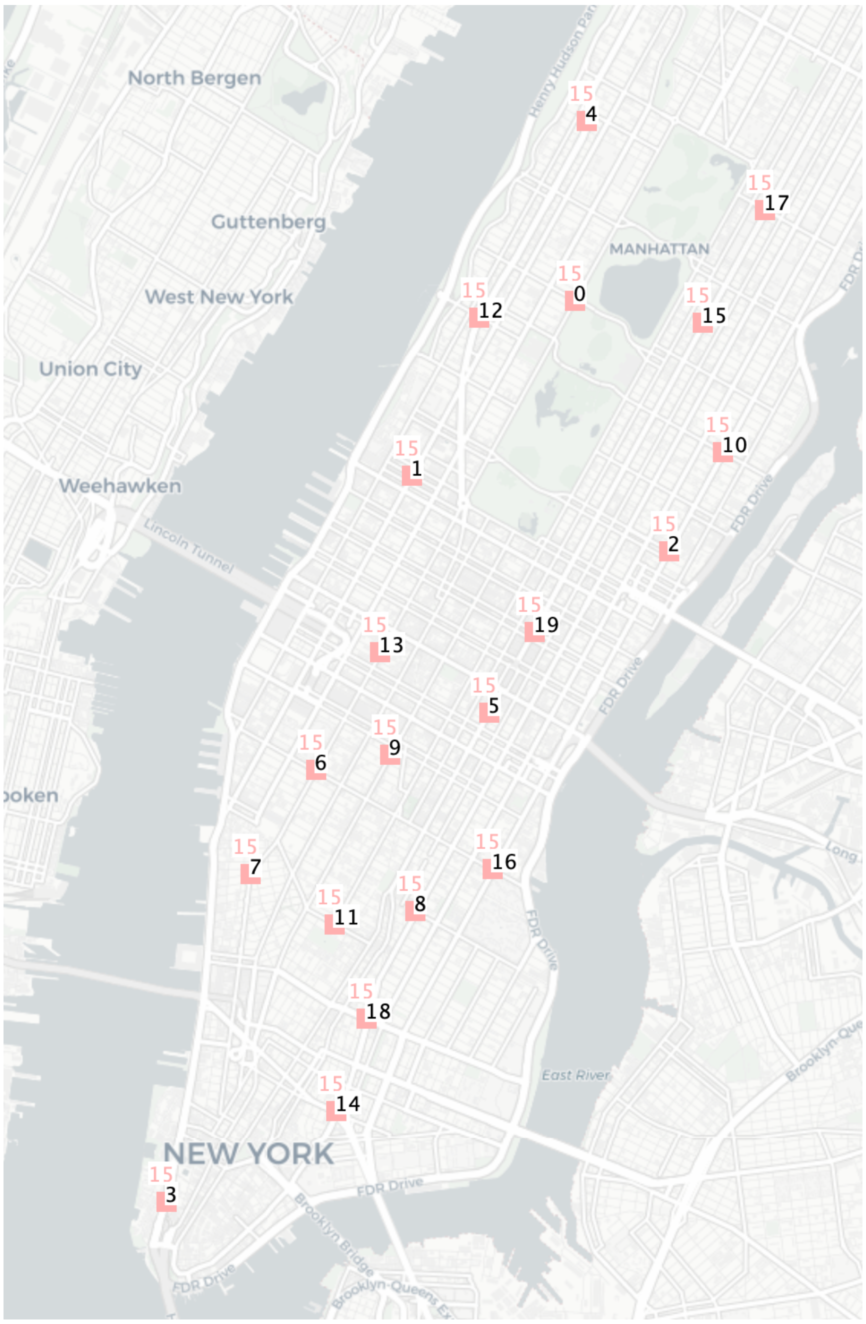


Figure 7.4: Locations of transfer stations and depots

7.2 Summary

We used the demand described above to evaluate and compare three methods - the Greedy Heuristic and the Transfer Insertion Heuristic algorithm, described in more detail above, and the Insertion Heuristic solving conventional ridesharing without transfers.

A basic comparison is shown in Table 7.3. The table shows that our proposed Transfer Insertion Heuristic indeed outperforms the Greedy Heuristic, even in all the statistics listed. Thus, the goal of improving the method in terms of performance has been achieved.

However, if we look at the comparison of the transfer-allowing methods with the conventional Insertion Heuristic method, the performance comparison is not so clear. The Greedy Heuristic achieved a slightly better result in terms of delay, but in all other statistics it performed worse. Transfer Insertion Heuristic achieved a significantly better result in terms of average delay and increased the number of requests served. On the other hand, it increased the total distance traveled and also needed more vehicles. This may be due to the fact that we do not aim to select plans with respect to distance traveled but select plans with the shortest waiting for a vehicle. Therefore, a vehicle that is far from the station and has a later arrival (that we prefer) may participate in the transfer.

Thus, the expectation that enabling transfers would reduce the total distance traveled was not confirmed. But the number of served demands is indeed slightly higher, and the average delay has decreased in the case of the Transfer Insertion Heuristic.

	INSERTION HEURISTIC	**TRANSFER INSERTION HEURISTIC**	**GREEDY HEURISTIC**
Dropped demands	95	78	128
Served demands	3832	3849	3799
Total transfers	0	1556	664
Total veh. distance *[km]*	28608	34519	37448
Tot. veh. dist. per demand served *[km]*	7.466	8.968	9.857
Used vehicles count	236	283	300
Average delay *[s]*	161	139	156

Table 7.3: Methods comparison

7.3 Other Detailed Statistics

In the following section, we focus on other selected statistics and compare all methods in more detail. We will analyze passenger delays, fleet utilization and finally focus on waiting at stations for transfers.

7.3.1 Delay

In this statistic, we focused on the percentage increase in a passenger's journey time compared to the fastest possible (i.e., direct) journey. The graphs (Figure 7.5) show only those demands that reached their destination with a non-zero delay. For the Insertion Heuristic this is 3824 out of 3832, for the Transfer Insertion Heuristic this is 3846 out of 3849, and for the Greedy Heuristic this is 3641 delayed out of 3799 total demands served.

The Insertion Heuristic and Greedy Heuristic have very similar, if not nearly identical, delay distributions. This is consistent with the average delays shown in Table 7.3, which differ by only 5 seconds.

For the Transfer Insertion Heuristic, we notice a lower distribution of the longest delays (an increase in travel time of 81 % or more). The distribution of delays shorter than 30 % of the minimum travel time has also increased. The Greedy Heuristic accounted for 20.3 % of the delayed demands, the Insertion Heuristic 23.5 %, and the Transfer Insertion Heuristic 29.9 %. If we add to these data the demands that arrived with zero delay, we get results of 24.5 %, 23.7 %, and 30.0 %, respectively. This reflects the improvement in average delay for the Transfer Insertion Heuristic method.

7.3.2 Vehicle occupancy

To evaluate the effectiveness of ridesharing, we use vehicle occupancy statistics. Figure 7.6 shows the percentage distribution of the number of passengers in the vehicle during the simulation time.

We note two highlights from the charts. First, the Greedy Heuristic has almost zero distribution of vehicle occupancy by two or more passengers. Thus, ridesharing is practically almost never occurring. For the Insertion Heuristic, on the other hand, we notice a (albeit very small) distribution of occupancy with three passengers. Ridesharing is applied the most in this method.

By far the most frequent is the occupancy of a single passenger in the vehicle, in all three cases. This situation occurs 60 % of the time. A large portion of the time, the vehicles are unoccupied. Moreover, from the Table 7.3 we can notice that in the case of Transfer Insertion Heuristic and Insertion Heuristic, some vehicles do not even start driving at all and remain unused for the whole 5 hours of the simulation scenario. If we choose not to consider in the graph vehicles that did not participate at all during the simulation (64 vehicles for the Insertion Heuristic, 17 for the Transfer Insertion Heuristic), the distribution of zero occupancy drops by 21 % and 5 % respectively. For the Insertion Heuristic, a drop in zero occupancy would result in a significant change in the distribution of the graph. From this distribution, we believe that the Insertion Heuristic uses the fleet most efficiently of all methods.

7.3.3 Fleet Utilization

In this statistic, we take a closer look at active vehicles. We consider an active vehicle to be one that has a non-empty plan. Vehicles that did not participate in the simulation at all are not included. From Figure 7.7, we can see how many vehicles were active for a given simulation time, and we can also calculate the total number of minutes of driving.

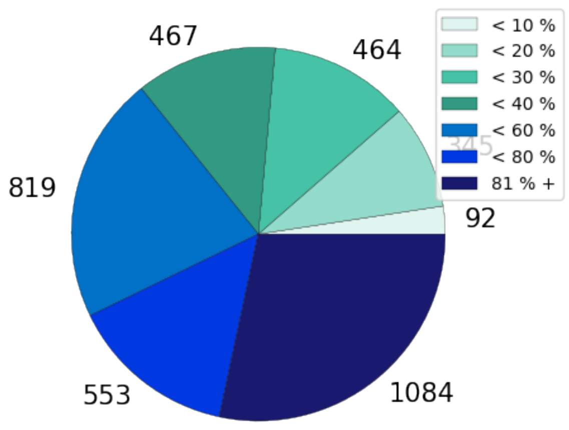
The Insertion Heuristic and Transfer Insertion Heuristic methods have very similar distributions, only the first bin of the histogram differs significantly by approximately 40 vehicles. The histogram is very different for the Greedy Heuristic method. By far, the largest number of vehicles are active for 10-20 % of the time, and this is approximately 100 vehicles more than the Insertion Heuristic and Transfer Insertion Heuristic methods.

If we calculate the total number of minutes of driving from the histograms, we get the lowest result for the Insertion Heuristic and the highest for the Greedy Heuristic. This corresponds to the efficiency of using ridesharing, which we also discussed in the previous section.

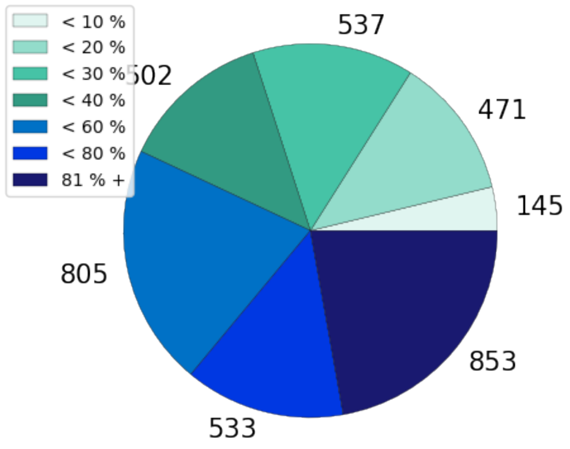
7.3.4 Vehicle Waiting Times at Transfers

The last statistic we will discuss is waiting at stations for transfers. This statistic is strongly dependent on the way we choose the final trip plan for the request. For the Greedy Heuristic, the plan with the longest possible wait is selected since the wait time was originally considered for electrotaxis recharging. In contrast, for the Transfer Insertion Heuristic, the plan that preferably does not contain the wait action or has a shorter wait time is selected.

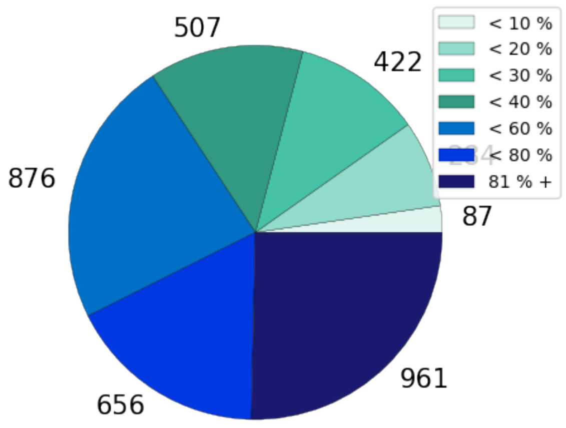
There were two points that caught our interest with this statistic. First, the Transfer Insertion Heuristic does not select a single plan containing a waiting action. That means, for all requests, the algorithm found some pair of vehicles to serve it such that the second vehicle arrived at the station later than the first or served them without transferring to avoid waiting. Therefore, we present only the Greedy Heuristic in Figure 7.8. Second, from the histogram of wait times, we notice a maximum of around 250 seconds. Such a time would not be sufficient for charging electric vehicles (as proposed in [11]) in today's conditions. In order to consider recharging vehicles in this way, we would have to extend the maximum tolerable delay for passengers several times to allow cars to wait long enough at stations, or passengers would have to demand significantly longer trips.



(a) : Insertion Heuristic

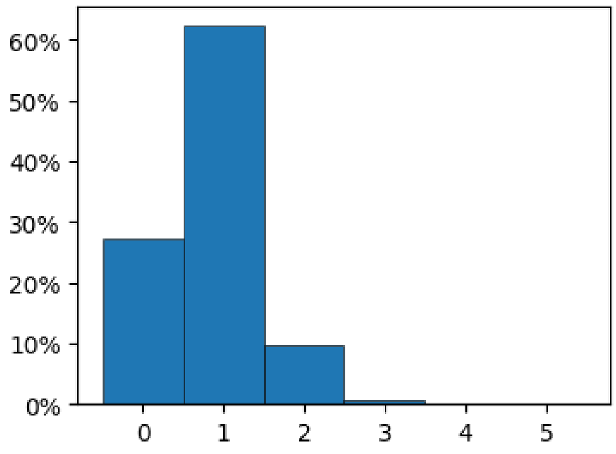


(b) : Transfer Insertion Heuristic

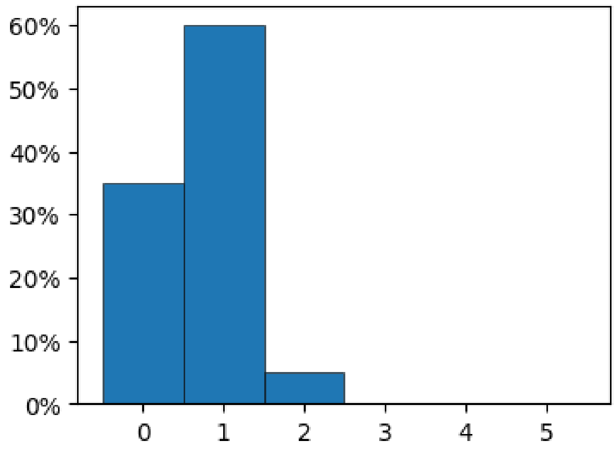


(c) : Greedy Heuristic

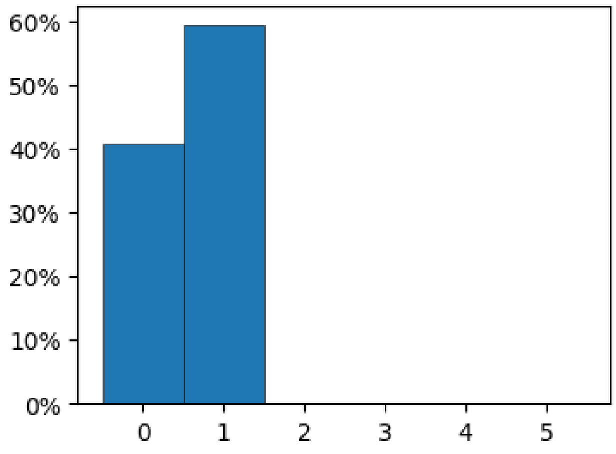
Figure 7.5: Delay in relation to the minimum trip duration



(a) : Insertion Heuristic

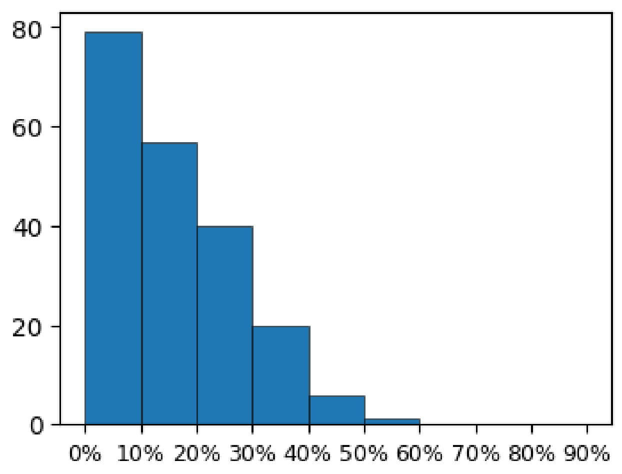


(b) : Transfer Insertion Heuristic

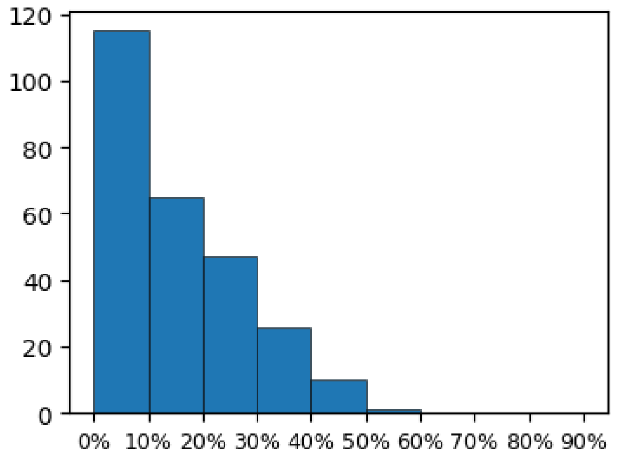


(c) : Greedy Heuristic

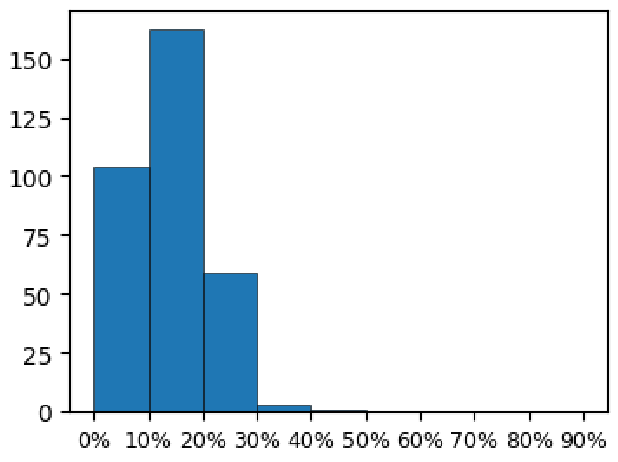
Figure 7.6: Average vehicle occupancy over simulation time



(a) : Insertion Heuristic



(b) : Transfer Insertion Heuristic



(c) : Greedy Heuristic

Figure 7.7: Vehicle utilization over simulation time

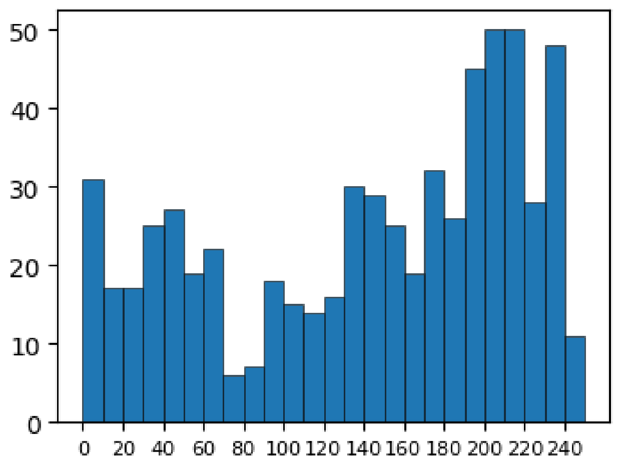


Figure 7.8: Waiting Times of Greedy Heuristic method [s]

Chapter 8
Conclusion

In this work, our motivation was to improve the efficiency of ridesharing by allowing transfers. We considered transfers at predefined stations and defined the problem as a MIP problem in Chapter 3. However, the optimal solution can only be found only for very small instances. That is why various heuristics are often used in practice. In Chapter 4, we described a solution method using the Greedy Heuristic, which is inspired from [11]. We tried to outperform this baseline method in terms of efficiency in the following Chapter 5, where we proposed a new method using the Insertion Heuristic.

The newly proposed method works in two phases. First, it searches for ride plans to serve the request without transferring. In the second phase, it then searches for stations where transfers are possible and finds pairs of vehicles that could serve the request with one transfer per station. For the final plan selection, the algorithm prefers those that do not have scheduled waiting time for vehicles at the stations, but at the same time, it also takes into account passenger delays.

We tested our solution and the baseline Greedy Heuristic method on real demand data of smaller instances first and evaluated it on a larger dataset of approximately 4000 demands. We compared the results of both methods and confirmed the better performance of our proposed algorithm compared to the baseline method from [11]. For comparison, we further evaluated the dataset using a conventional Insertion Heuristic method that does not consider transitions.

The experiment showed that the new solution with transfers could increase the total number of requests served and reduce the average delay for passengers. However, this solution did not meet expectations with a decrease in total distance. Neither of the methods considering transfer achieved better results in terms of kilometers traveled.

Along with the implementation of solvers that consider transfers, we integrated the actions and activities required for transfers into the SiMoD tool. The simulation environment is therefore ready for the further implementation of new methods for solving ridesharing with transfers.

For our proposed solution, there remains room for optimizing the final plan selection rules. It is worth considering defining the criteria for selecting the final plan so that it effectively decides when to use the transfer or serve the request by the direct route. For example, if demand is lower in a given time window and many vehicles are available, it does not make sense to extend the distance traveled for each request by driving to transfer stations. In other cases, it may be preferable to keep the vehicle waiting at the station for a while, thus reducing the distance that would otherwise have to be traveled by a vehicle that is further away from the station and therefore has a longer journey time.

Bibliography

[1] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012.

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

[3] Yosser Ben Cheikh, Christian Tahon, and Slim Hammadi. An evolutionary approach to solve the dynamic multi-hop ridematching problem. *SIMULATION*, 93, 12 2016.

[4] Wenyi Chen, Martijn Mes, Marco Schutten, and Job Quint. A Ride-Sharing Problem with Meeting Points and Return Restrictions. *Transportation Science*, 53(2):401–426, March 2019.

[5] Yen-Long Chen, Kuo-Feng Ssu, and Yu-Jung Chang. Real-time transfers for improving efficiency of ridesharing services in the environment with connected and self-driving vehicles. In *2020 International Computer Symposium (ICS)*, pages 165–170, 2020.

[6] Brian Coltin and Manuela Veloso. Ridesharing with passenger transfers. volume 2, pages 1299–1300, 05 2013.

[7] Jean-François Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[8] Wesam Herbawi and Michael Weber. Evolutionary multiobjective route planning in dynamic multi-hop ridesharing. In Peter Merz and Jin-Kao Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 84–95, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[9] Wesam Herbawi and Michael Weber. A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows. *GECCO'12 - Proceedings of the 14th International Conference on Genetic and Evolutionary Computation*, 07 2012.

[10] Yunfei Hou, Xu Li, and Chunming Qiao. Tictac: From transfer-incapable carpooling to transfer-allowed carpooling. pages 268–273, 12 2012.

[11] Yunfei Hou, Weida Zhong, Lu Su, Kevin Hulme, Adel Sadek, and Chunming Qiao. Taset: Improving the efficiency of electric taxis with transfer-allowed rideshare. *IEEE Transactions on Vehicular Technology*, 65:1–1, 12 2016.

[12] Sepide Lotfi, Khaled Abdelghany, and Hossein Hashemi. Modeling framework and decomposition scheme for on-demand mobility services with ridesharing and transfer. *Computer-Aided Civil and Infrastructure Engineering*, 34, 03 2018.

[13] Shuo Ma, Yu Zheng, and Ouri Wolfson. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.

[14] Neda Masoud and R. Jayakrishnan. A decomposition algorithm to solve the multi-hop peer-to-peer ride-matching problem. *Transportation Research Part B: Methodological*, 99:1–29, 05 2017.

[15] A. Rais, F. Alvelos, and M.S. Carvalho. New mixed integer-programming model for the pickup-and-delivery problem with transshipment. *European Journal of Operational Research*, 235(3):530–539, 2014.

[16] Douglas O. Santos and Eduardo C. Xavier. Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. AAAI Press, 2013.

[17] Jörn Schönberger. Scheduling constraints in dial-a-ride problems with transfers: a metaheuristic approach incorporating a cross-route scheduling procedure with postponement opportunities. *Public Transport*, 9, 07 2017.

[18] Ashutosh Singh, Abubakr Alabbasi, and Vaneet Aggarwal. A distributed model-free algorithm for multi-hop ride-sharing using deep reinforcement learning, 2019.

[19] Amirmahdi Tafreshian and Neda Masoud. Trip-based graph partitioning in dynamic ridesharing. *Transportation Research Part C: Emerging Technologies*, 114:532–553, 2020.

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

[21] Raja Subramaniam Thangaraj, Koyel Mukherjee, Gurulingesh Raravi, Asmita Metrewar, Narendra Annamaneni, and Koushik Chattopadhyay. Xhare-a-ride: A search optimized dynamic ride sharing system with approximation guarantee. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1117–1128, 2017.

[22] Rui Yao and Shlomo Bekhor. A dynamic tree algorithm for peer-to-peer ridesharing matching. *Networks and Spatial Economics*, 21, 12 2021.

Appendix A
List of abbreviations

VRP	Vehicle Routing Problem
P2P	Peer-to-Peer
ILP	Integer Linear Programming
MMTP	Multi-modal Trip Planner
MOEA	Multiobjective Evolutionary Algorithm
MoD	Mobility on Demand
MACGeO	Metaheuristics Approach Based on Controlled Genetic Operators
TASeT	Transfer-Allowed Shared eTaxis
MIP	Mixed-Integer Programming
DARP	Dial-a-Ride Problem

Appendix B
Attachment content

`dataset/`	directory with dataset files used for the evaluation of this project
`dataset/test.cfg`	main config file for setting simulation parameters
`thesis/`	directory with source file for this thesis
`thesis/zadani.pdf`	Specification of the bachelor project assignment
`simod/`	project source code
`simod/python/`	scripts to plot statistics, histograms and tables
`simod/src/`	source code in Java
`simod/.../ridesharing/`	folder with implemented solvers

```
attachement
 ├─ dataset/
 │   └─ test.cfg
 └─ simod/
     ├─ python/
     └─ src/main/
         └─ ...aic/simod/ridesharing/
             ├─ transferinsertion/
             ├─ greedyTASeT/
             └─ insertionheuristic/
thesis/
 └─ zadani.pdf
```