Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Combined Solution for Ridesharing and Delivery in Urban Areas

Martin Bláha

Supervisor: Ing. David Fiedler
Field of study: Open Informatics
Subfield: Artificial Intelligence and Computer Science
January 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Bláha Martin** Personal ID number: **492307**

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:

Combined Solution for Ridesharing and Delivery in Urban Areas

Bachelor’s thesis title in Czech:

Kombinované řešení sdílených jízd a doručování v městských oblastech

Guidelines:

Mobility-on-demand (MoD) systems with ridesharing are widely discussed mobility concepts. At the same time, the growing share of online shopping leads to an extensive amount of package delivery requirements. To improve efficiency, it is natural to combine these two concepts and deliver packages with the MoD system fleet. However, the ridesharing and delivery requirements are principally different. The packages can be picked up and delivered during the whole day, and the travel demand is usually known upfront. In contrast, MoD passengers have to be picked up and dropped off within short time windows, and the demand is unknown. Therefore, we need to design a system that will combine the knowledge from the ridesharing and delivery fields.
1) Study the literature covering the ridesharing in MoD systems combined with delivery.
2) Based on your research, implement a state-of-the-art method for ridesharing with delivery 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 and parcel delivery 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) With the help of your research and experience from the baseline implementation, design a new/improved algorithm for ridesharing with delivery. You should focus on ridesharing efficiency, computational time, or on removing the limitations of the existing methodology.
6) Implement the designed algorithm in the SiMoD framework.
7) Evaluate your algorithm in SiMoD and analyze the results.

Bibliography / sources:

[1] B. Li, D. Krushinsky, H. A. Reijers, and T. Van Woensel, “The Share-a-Ride Problem: People and parcels sharing taxis,” European Journal of Operational Research, vol. 238, no. 1, pp. 31–40, Oct. 2014, doi: 10.1016/j.ejor.2014.03.003.
[2] N.-Q. Nguyen, N.-V.-D. Nghiem, P.-T. DO, K.-T. LE, M.-S. Nguyen, and N. MUKAI, “People and parcels sharing a taxi for Tokyo city,” in Proceedings of the Sixth International Symposium on Information and Communication Technology, New York, NY, USA, Dec. 2015, pp. 90–97. doi: 10.1145/2833258.2833309.
[3] D. Földes and C. Csiszár, “Model of information system for combined ride-sourcing service,” in 2017 Smart City Symposium Prague (SCSP), May 2017, pp. 1–6. doi: 10.1109/SCSP.2017.7973841.
[4] A. D. Febbraro, D. Giglio, and N. Sacco, “On Exploiting Ride-Sharing and Crowd-Shipping Schemes within the Physical Internet Framework,” in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Nov. 2018, pp. 1493–1500. doi: 10.1109/ITSC.2018.8569761.
[5] A. Godart, H. Manier, C. Bloch, and M.-A. Manier, “MILP for a Variant of Pickup Delivery Problem for both Passengers and Goods Transportation,” in 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Oct. 2018, pp. 2692–2698. doi: 10.1109/SMC.2018.00460.
[6] F. Wang, Y. Zhu, F. Wang, and J. Liu, “Ridesharing as a Service: Exploring Crowdsourced Connected Vehicle Information for Intelligent Package Delivery,” 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), 2018, doi: 10.1109/IWQoS.2018.8624152.
[7] K. Manchella, A. K. Umrawal, and V. Aggarwal, “FlexPool: A Distributed Model-Free Deep Reinforcement Learning Algorithm for Joint Passengers and Goods Transportation,” IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 4, pp. 2035–2047, Apr. 2021, doi: 10.1109/TITS.2020.3048361.

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: **09.01.2022** Deadline for bachelor thesis submission: **10.01.2023**

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 his 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

I want to sincerely thank Ing. David Fiedler for his time, advice, and guidance throughout this project, and for his helpfulness during the consultations. I also want to thank my family for supporting me during my studies at CTU FEE.

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 theses. Prague, January 10, 2023

Abstract

Due to the increasing demand for urban transport in the last few years, the number of vehicles in cities is rising, which results in traffic jams, lack of parking places, and excessive pollution. These problems can be solved with ridesharing. Nowadays there are two transport service types: taxi service for people and delivery service for parcels, which operate separately.

In this work, we propose a solution that combines these two services together, which should increase the number of shared rides and therefore lower the number of vehicles in cities. We study the literature covering the topic of shared rides between people and parcels. We introduce an existing heuristic method and then we propose improvements to this method using the Insertion heuristic. We evaluate both of these methods on real data from New York City and compare the performances. The improved method managed to lower the mileage by 40 % and decrease the number of used vehicles by 28 %.

Keywords: DARP, SARP, ridesharing, ridesharing in urban areas, package delivery, ridesharing with delivery, insertion heuristic, SiMoD

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

Abstrakt

Vzhledem k rostoucí poptávce po městské dopravě v posledních letech stoupá počet vozidel ve městech, což má za následek dopravní zácpy, nedostatek parkovacích míst a nadměrné znečištění ovzduší. Tyto problémy lze vyřešit ridesharingem. V současné době existují dva typy přepravních služeb: taxi služba pro lidi a doručovací služba pro zásilky, které fungují samostatně.

V této práci navrhujeme řešení, které kombinuje tyto dvě služby dohromady, což by mělo zvýšit počet sdílených jízd a tím snížit počet vozidel ve městech. Studujeme literaturu zabývající se tématem sdílených jízd lidí a zásilek. Představíme existující heuristickou metodu a poté navrhneme vylepšení této metody pomocí Insertion heuristiky. Obě tyto metody vyhodnocujeme na reálných datech z New Yorku a porovnáváme výsledky. Vylepšená metoda dokázala snížit počet najetých kilometrů o 40 % a snížit počet použitých vozidel o 28 %.

Klíčová slova: DARP, SARP, sdílené jízdy, sdílené jízdy v městských oblastech, doručování zásilek, sdílené jízdy s doručováním, insertion heuristika, SiMoD

Překlad názvu: Kombinované řešení sdílených jízd a doručování v městských oblastech

Contents

1 Introduction **1**
1.1 Project Target 1

2 Literature review **3**

3 Problem description **7**
3.1 Mathematical Formulation 8

4 Solution - Baseline method **11**
4.1 Find taxi plans 11
4.2 Selecting available taxis 12
4.3 Checking validity of vehicle plan 12
4.4 Time complexity 13

5 Solution - Improved method **15**
5.1 Multiple passengers heuristic . . . 15
5.1.1 Selecting available taxis 15
5.1.2 Checking validity of vehicle plan 16
5.2 Multiple passenger Insertion Heuristic 16
5.3 Time complexity 16

6 Implementation **19**
6.1 Solvers implementation 19
6.2 Integration into SiMoD 20
6.3 SiMoD input and output 20
6.4 Visualisation 20

7 Evaluation **25**
7.1 Testing environment description 25
7.1.1 Demand data 25
7.1.2 Road network 26
7.1.3 Vehicles 27
7.2 Results 27

8 Conclusion **37**

Bibliography **39**

A Attachment content **41**

Figures

6.1 Simplified class diagram for vehicle and request classes. Newly implemented classes are colored green, modified classes are colored orange, and original classes are colored white. A name in italics indicates an interface. The arrow means that the class, it is pointing to, inherits from the other class. The dashed line indicates that the connected classes have some interaction between them. 21
6.2 Simplified class diagram for main simulation-related classes, dispatcher, visualization layer, and heuristic solvers. Newly implemented classes are colored green, modified classes are colored orange, and original classes are colored white. The arrow means that the class, it is pointing to, inherits from the other class. The dashed line indicates that the connected classes have some interaction between them. 22
6.3 The simulation window at the start of an experiment 23
6.4 Running simulation in SiMoD with highlighted taxi. Red dots mark passenger requests and green dots mark parcel requests. Taxis are marked with blue triangles. The highlighted taxi also shows the number of passengers onboard (in this case 0). The current taxi plan is highlighted with yellow color. Pickup points on highlighted route are marked with turquoise dots and delivery points are marked with pink dots. These points also show an ID of the request they belong to. The pink squares mark the depots. Each depot has a numeric ID marked with a black number and the pink number indicates the number of taxis currently in the depot. 24
7.1 Demand histogram for the first scenario 26
7.2 Demand histogram for the second scenario 26
7.3 Demand histogram for the third scenario 27
7.4 Used road network for Manhattan and surrounding areas 28
7.5 Map of the depots 31
7.6 Demand heatmaps 32
7.7 Combined occupancy for the first scenario. The vertical axis shows the percentage of the time for the occupancies. 33
7.8 Combined occupancy for the second scenario. The vertical axis shows the percentage of the time for the occupancies. 34
7.9 Combined occupancy for the third scenario. The vertical axis shows the percentage of the time for the occupancies. 35

Tables

7.1 Ridesharing rates table for the 3rd experiment (50k requests in 1h) . . 29
7.2 Methods comparison table for the 1st experiment (24k requests in 24h) 30
7.3 Methods comparison table for the 2nd experiment (25k requests in 1h) 30
7.4 Methods comparison for the 3rd experiment (50k requests in 1h) . . 30

Chapter 1
Introduction

The number of vehicles in urban areas is rising due to ever increasing demand for urban transport. In recent years, the growing demand for delivery of parcels from online shops and for food delivery also contributes to the increase in the number of vehicles in cities. This causes several issues: traffic jams, noise, pollution and lack of parking places. These issues can be solved by ridesharing. It decreases the number of vehicles needed to satisfy the demand, and therefore also lowers transportation costs.

In today's transportation systems, passengers and parcels are transported separately since they have different transport requirements. The packages can usually be picked up and delivered during the whole day and the demand is known beforehand. In contrast, passengers have to be picked up and delivered within short time windows, and the demand is not known in advance. And most of the taxi seats and the trunks are usually empty which is inefficient and uneconomical.

In urban areas, parcels usually have small dimensions and low weight, so they can be transferred along with a passenger without difficulties since passenger travel comfort and travel time will be affected minimally or not at all. This leads us to the idea of combining these two services into one that could simultaneously use the capacity of vehicles for people and for parcels.

1.1 Project Target

This project aims to study the methods for solving ridesharing in Mobility-on-Demand systems combined with delivery.

We choose a suitable method that is able to solve real-world-sized instances in a reasonable amount of time, uses a fleet of vehicles to transport passengers and parcels, and was tested on real data. Then we integrate the selected method into the SiMoD [1] tool and evaluate its performance and efficiency.

As a next step, we design an improved algorithm for ridesharing with delivery, also implement it in SiMoD, and evaluate its results. We compare the baseline method with the improved method in terms of mileage, the number of used vehicles, the number of shared demands, the computational

[1] `https://github.com/aicenter/simod`

time, and vehicle occupancy. Finally, we analyze and discuss the results.

Chapter 2

Literature review

Ridesharing with parcel delivery is a relatively new idea. Ridesharing allows taxi services to reduce mileage and the number of cars, lowering transportation costs and the number of vehicles in urban areas. By combining ridesharing with parcel delivery we should improve these parameters even further.

The ridesharing problem has two basic scenarios: static and dynamic. In a static scenario, all the requests are known beforehand thus we can calculate complete vehicle plans in advance. In a dynamic scenario, however, we do not know the demand beforehand, instead, the requests appear gradually and the algorithm has to work with the new requests and the currently available vehicles and create the plans in real-time.

Study [3] reviews different concepts of shared transportation of passengers and freight. A common model of ridesharing with delivery uses centralized ridesharing, i.e., we have a fleet of vehicles (taxis) that serves passengers and parcels. But there is a wide variety of other models, for example, the parcels can be transferred and temporarily stored in special lockers, the vehicles can be autonomous, instead of vehicles we can use a dedicated railroad network with automated parcel loading/unloading system [6], or we can utilize an already existing network of public transport [9].

The problem of finding the best possible schedules for taxis to serve the customers' requests is known as Dial-a-Ride-Problem (DARP), which is a generalization of Vehicle Routing Problem (VRP) [11].

One of the first models for ridesharing with delivery was proposed by Li et al. in [7]. The authors introduce a newly developed class of models called Share-a-Ride Problem (SARP) which allows passengers and parcels to share the same vehicles. The presented model allows shared transportation of passengers and parcels under the following conditions: there can be at most 1 passenger onboard in each taxi and the ride of the passenger can be interrupted only by a predefined number of stops, i.e., picking up or dropping off a parcel. It also proposes the Freight Insertion Problem (FIP) which is reduced SARP. FIP starts from an existing route for transporting the passengers and it inserts the freight requests into the route (so that the given conditions are fulfilled). Compared to a direct delivery service, the shared model had better results in the total distance and profit. It was tested,

however, on small instances only (3 taxis and 12 requests) since the solver was searching for an optimal solution.

In [10] authors use the model from [7], but they do not allow any stops during the ride of a passenger. Unlike in [7], authors propose a heuristic method to solve this problem and they tested it on real data from Tokyo taxis consisting of 22 000 requests throughout the whole day and over 4000 taxis. This study investigates static scenario only. The improvement (in comparison with direct delivery) was only around 5 % in terms of mileage, the number of used cars, and overall profit.

The authors of [10] further extend this method in [5]. They introduce a path-finding algorithm that speeds up the search of the shortest paths in a graph, and a local search technique that improves the quality of the solution. First, a solution for all requests is found using greedy search. Then a value of effectiveness is calculated for each request and part of the requests with the lowest effectiveness value are reinserted to increase overall benefit. This routine is repeated until there is no increase in benefit. Here the difference between the results of direct delivery and the heuristic method is more significant than in [10]. The overall benefit is higher by 11 %, nearly 70 % of requests are shared, and the number of used taxis is lower by 40 %.

The studies [1] and [14] propose models with Shared Autonomous Vehicles (SAVs). Beirigo, Schulte, and Negenborn in [1] suggest a model that allows transporting people and freight together with no limitations. For this purpose, the model uses the SAVs. The authors also introduce the possibility of transferring the packages between the taxis. In [14] authors explores the variable capacity share-a-ride problem. This means the autonomous vehicles used for transportation have different sizes and they are scheduled so that the transport capacity is used where needed and is not wasted. The traffic simulations are done on a square grid which is divided into several areas and each area is of a specific type: residential, industrial, or campus. The model uses these areas to simulate different demands flow at different times during the day, for example, in the morning most requests will be from the residential area to the industrial area. Both [1] and [14] solve the problem for small instances only (up to 32 requests) using Mixed-integer linear programming.

Febbraro, Giglio, and Sacco explore the possibility of peer-to-peer ridesharing with delivery in [4]. Peer-to-peer ridesharing is based on private drivers, who are willing to share their ride with a passenger or a parcel. An algorithm will match drivers to those passengers and parcels who have a similar transportation itinerary.

An unusual model is introduced in work [12]. The passengers and parcels are transported not by vehicles but by public bus service. The parcels are transferred to bus stops and from there they are delivered to their destination by micro-logistics operators who will use environmentally friendly vehicles.

Since AI is more and more popular, there are attempts to use it as the solution to the ridesharing problem. Study [8] uses a deep learning method to create the scheduling algorithm that is supposed to learn optimal taxi dispatching policies. Authors in [13] try to achieve the same result by using

a genetic algorithm for this purpose.

Most of the methods mentioned above were designed to solve static scenario only. Unfortunately, this is not applicable in the real world, where the scenario is dynamic. However methods from [5], [8], and [13] are able to solve the dynamic scenario and method in work [10] is designed well enough it can be also used in dynamic mode.

When selecting a method to implement, we wanted a model that uses a fleet of vehicles and centralized ridesharing. We also considered the size of instances. We wanted a method that was capable of solving instances of several thousand to several tens of thousands of requests per hour. As a baseline method, we chose the heuristic algorithm from [10] because it matches these requirements.

Chapter 3

Problem description

The problem of finding the best possible schedules for taxis to serve the customers' requests is known as Dial-a-Ride-Problem (DARP), which is a generalization of Vehicle Routing Problem. DARP is an NP-hard problem [7] therefore we are not able to find the optimal solution for instances from the real world. Thus it is typically solved using heuristic algorithms that approximate the optimal solution and find it in a reasonable amount of time. In this chapter, we will describe the Share-a-Ride-Problem (SARP). SARP is an extension of DARP, where passengers share taxis with parcels.

The problem under consideration is supposed to be handled by a taxi company. During the day, its dispatching system receives requests to serve. The system supports two types of transportation requests: passenger requests and parcel requests. Each transportation request is characterized by a pair of actions - pickup action and delivery action. Both pickup and delivery actions contain information about their locations with corresponding time windows, in which they must be served.

From the list of given requests, the system will schedule taxis driving around the city to serve requests. Each taxi has to depart at a depot and return to this depot after finishing all its scheduled requests. At any given time, neither the number of passengers nor the amount of freight can exceed the taxi's maximum capacity. The system allows the following transportation scenarios for each taxi:

1. carry only one passenger
2. carry parcels
3. carry both one passenger and parcels

A taxi is allowed to deliver some parcels along with a passenger but is not allowed to carry more than one passenger at the same time. When a passenger is onboard, he will be delivered directly to the destination without any interruption (i.e., no stop for parcel pickups or deliveries). Taxis are allowed to wait at a pickup or delivery spot for a limited time. For safety reasons, the total traveling time of a taxi in one day can not exceed a specified maximum duration. The problem is to find the set of valid taxi trajectories minimizing total cost. The total cost is the sum of the costs of all taxi routes.

3.1 Mathematical Formulation

The Share-a-Ride-Problem can be defined on a weighted and directed graph $G = (V, E)$. The set of nodes V includes subsets $V^{po}, V^{fo}, V^{pd}, V^{fd}, V^{de}$. The sets $V^{po} = \{1, ..., n\}$ and $V^{fo} = \{n+1, ..., n+m\}$ indicate the passenger and freight origins respectively. Passenger and freight destinations are represented by sets $V^{pd} = \{s+1, ..., s+n\}$ and $V^{fd} = \{s+n+1, ..., 2s\}$ respectively. The numbers n and m are the numbers of passengers and parcels respectively, and s is the total number of requests, so $s = n + m$. Each node $i \in V^{po} \cup V^{fo} \cup V^{pd} \cup V^{fd}$ has a pair of parameters (ω_i, ϕ_i): ω_i represents the maximum waiting time to serve request i and ϕ_i represents the change of weight of the taxi after visiting node i. In the case of pickup, ϕ_i is positive, in the case of delivery, it is negative. Additionally, every node i has a time window $[e_i, l_i]$ (early time, late time) required for either the pickup or the delivery. The time to travel from node i to node j is denoted by $t_{i,j}$.

The set of depots is represented by $V^{de} = \{2s+1, ..., 2s+|K|\}$, and $K = \{1, ..., |K|\}$ is the set of taxis to serve possible requests. Taxi k has a limited capacity σ_k and can work no longer than τ_k^{max} minutes per day. Each taxi k has its starting depot $\theta_k \in V^{de}$ and can end its ride in arbitrary depot $\theta \in V^{de}$.

For every node $i \in V$, the set of nodes following i is $\Gamma^+(i)$ and the set of nodes preceding i is $\Gamma^-(i)$. The list T contains time points of taxi arrivals to nodes and departures from nodes. The maximum waiting time allowed at point $v \in V$ is ω_v^{max}. At every node $i \in V^{po} \cup V^{pd} \cup V^{fo} \cup V^{fd}$, a passenger or parcel is loaded on the taxi or unloaded from the taxi. The change of weight of the taxi after visiting point i is denoted by ϕ_i. In the case of pickup, ϕ_i is positive, in the case of delivery, it is negative. The cost of the route from node i to node j is denoted by $\gamma_{i,j}$.

The following integer program represents the problem of optimization of taxi routes:

$$\min \sum_{k \in K} \sum_{(i,v) \in E} \gamma_{i,j} x_{i,j}^k \tag{3.1}$$

subject to:

$$\sum_{j \in \Gamma^+(i)} \sum_{k \in K} x_{i,j}^k \leq 1, \qquad \forall i \in V^{po} \cup V^{pd} \cup V^{fo} \cup V^{fd} \tag{3.2}$$

$$\sum_{j \in \Gamma^+(i)} x_{i,j}^k = y_i^k, \qquad \forall i \in V^{po} \cup V^{fo}, k \in K \tag{3.3}$$

$$\sum_{j \in \Gamma^+(i)} x_{i,j}^k = \sum_{j \in \Gamma^-(i)} x_{j,i}^k, \qquad \forall i \in V, k \in K \tag{3.4}$$

$$\sum_{j \in \Gamma^+(i)} x_{i,j}^k = \sum_{j \in \Gamma^-(i+s)} x_{j,i+s}^k, \qquad \forall i \in V^{po} \cup V^{fo}, k \in K \tag{3.5}$$

$$x_{i,i+s}^k = y_i^k, \qquad \forall i \in V^{po}, k \in K \tag{3.6}$$

$$a_j^k = (d_i^k + t_{i,j}) x_{i,j}^k, \qquad \forall (i,j) \in E, \forall k \in K \tag{3.7}$$

$$e_i \leq a_i^k \leq l_i, \qquad \forall i \in V^{po} \cup V^{pd} \cup V^{fo} \cup V^{fd}, k \in K \tag{3.8}$$

$$a_\theta^k - d_{\theta_k}^k \leq \tau_k^{max}, \qquad \forall k \in K, \theta_k \in V^{de}, \theta \in V^{de} \tag{3.9}$$

$$d_i^k - a_i^k \leq \omega_i^{max}, \qquad \forall i \in V^{po} \cup V^{pd} \cup V^{fo} \cup V^{fd}, k \in K \tag{3.10}$$

$$w_j^k = (w_i^k + \phi_j) x_{i,j}^k, \qquad \forall (i,j) \in E, \forall k \in K \tag{3.11}$$

$$w_i^k \geq max\{0, \phi_i\}, \qquad \forall i \in V, k \in K \tag{3.12}$$

$$w_i^k \leq min\{\sigma_k, \sigma_k + \phi_i\}, \qquad \forall i \in V, k \in K \tag{3.13}$$

The variable $x_{i,j}^k$ is equal to 1 if taxi k goes from point i to point j (0 otherwise) $\forall (i,j) \in E$. Variable y_i^k is equal to 1 if request i is served by taxi k (0 otherwise) $\forall i \in V^{po} \cup V^{fo}, k \in K$. The time point when taxi k arrives at point i and departs from point i is denoted by variables a_i^k and d_i^k respectively. A load of taxi k after visiting stop v is $w_v^k, \forall v \in V$.

The objective function (3.1) minimizes the overall cost which is calculated as the sum of the costs of all edges that were used by taxis. Constraints (3.2) and (3.3) ensure that every request is served at most once. Constraint (3.4) maintains conservation flow at all nodes, i.e., the same number of vehicles that came into a node must also come out of this node. Constraint (3.5) enforces that if a passenger or parcel is picked up, it must be delivered and constraint (3.6) enforces that the passenger is delivered directly without any intermediate stops. Constraint (3.7) links the arrival, departure, and travel time between two points. Constraint (3.8) forces that the arrival time of the taxi to the node is inside of the time window. Constraint (3.9) limits the maximum riding time of the taxi. Constraint (3.10) limits the maximum

waiting time at stops. Constraints (3.11), (3.12), and (3.13) restrain the taxi loads - (3.11) ensures that the carried weight is updated after the taxi leaves the node, (3.12) controls the lower bound of the taxi load and (3.13) controls the upper bound of the taxi load.

Chapter 4

Solution - Baseline method

In this chapter, we will describe the solution using the heuristic method from [10]. The proposed method tries to minimize the overall cost, as described in the previous chapter. The input of the algorithm is requests sorted in ascending order by their pickup times. For each request, the algorithm tries to insert the request into the plan of every taxi, which is available at the time of pickup. From all possibilities, it selects the option which increases the overall cost the least. The algorithm considers the possibility of rejecting the request if there are no available taxis found.

We used *Heuristic algorithm for sharing model* and *Find a schedule for trajectory P_k of taxi k* from chapter 4 of study [10] and modified them for our purposes. We work with a dynamic scenario instead of a static one, so the *Heuristic algorithm* is called repeatedly and every time it gets a batch of new requests to process. In comparison with *Find a schedule* algorithm, we do not search for a free parking place in case a taxi arrives too early to the next point. That is because it simply cannot happen in a dynamic scenario. When the solver receives a new request, the passenger or the parcel is already waiting at the pickup location, therefore the taxi cannot arrive there early. To simplify the problem we calculate the cost by the traveled distance. This means that the objective of our algorithm is to minimize the overall traveled distance.

4.1 Find taxi plans

Each action in the plan has an *early time* value (the earliest time when the request can be picked up), and an *late time* value (the latest time when the request can be picked up). In the first step of Algorithm 1, all requests are sorted incrementally by the late time of their pickup action and are put into a new list V'. For each request $r \in V'$, we select taxis that are currently available to serve request r into a list K_a. This selection process is described in Algorithm 2. If the list K_a is not empty, we proceed to check the validity of potential new vehicle plans (if there are no available taxis at the moment, the request is rejected). This process is described in Algorithm 3. If the new plan with request r is valid, the algorithm proceeds to calculate the new total

cost. If the new total cost c_r^k is lower than the current lowest cost c^*, the c^* is updated along with the best taxi k^*. If the k^* was found, it gets selected to serve request r. Otherwise, we reject the request.

Algorithm 1 Find taxi plans

```
1:  V' = ∅
2:  Sort new requests incrementally by pickup lateTime into list V'
3:  for all request r ∈ V' do
4:      K_a ← SelectAvailableTaxis()
5:      if K_a is not empty then
6:          c* = ∞
7:          for all taxis k ∈ K_a do
8:              Insert request r into vehicle plan P_k of taxi k
9:              TrySchedule(P_k)
10:             c_r^k = new total cost in case taxi k serves request r
11:             if c_r^k < c* then
12:                 c* = c_r^k
13:                 k* = k
14:             end if
15:         end for
16:         if k* ≠ None then
17:             Insert request r into route of taxi k*
18:             Update the total cost
19:         else
20:             Reject request r
21:         end if
22:     else
23:         Reject request r
24:     end if
25: end for
```

4.2 Selecting available taxis

The Algorithm 2 goes through all taxis and selects those which are currently available. A taxi is considered available when it currently is not carrying a passenger and if the request is a parcel request, it has to have enough free freight capacity as well.

4.3 Checking validity of vehicle plan

The Algorithm 3 illustrates the procedure to set up time windows in the vehicle plan P_k of taxi k. We consider only one variant of plan P_k in which the actions' order is determined by their late time (the maximum time in which they must be picked up). Then for every node $P_k[j], j \in [0, |P_k| - 1]$

Algorithm 2 Select available taxis

```
1: Let K be the set of all taxis
2: K_a = ∅
3: for all taxi k ∈ K do
4:     if taxi k is not currently carrying a passenger then
5:         if current request is passenger then
6:             K_a += taxi k
7:         else
8:             if taxi k has sufficient capacity for the package then
9:                 K_a += taxi k
10:            end if
11:        end if
12:    end if
13: end for
14: Return K_a
```

we check the type of action. If the action is a dropoff, we can proceed to the calculation of time windows straight away. If the action is a pickup, we need to distinguish between passenger and parcel. For the passenger, we simply check whether there is another passenger onboard already. For the parcel, we check the sufficient capacity of the taxi k at the moment of pickup. If the capacity is sufficient then the algorithm proceeds to calculate the time windows. Otherwise, the algorithm terminates, because the schedule is not feasible. Next, we set up the time windows for the current action. We calculate the time needed for the taxi to get from $P_k[j]$ to $P_k[j+1]$ and store it in *time*. Then we calculate the early time and late time in which taxi k can reach node $j+1$. On line 9, we check if the early time is not greater than $l_{P_k[j+1]}$ (the taxi arrives at the pickup point late). If everything is correct, then the new schedule is feasible and we return it.

4.4 Time complexity

Let s is the overall number of requests, and K be the set of taxis. We take every request and for every available taxi, we check the validity of its plan. So we do the plan checking for at most s requests and at most $|K|$ taxis, which is at most $s \times |K|$ times. Theoretically, the plan can contain all requests, thus having the length s. Finding the distance between the two following locations in the vehicle plan is constant since we use the distance matrix. Therefore the resulting time asymptotic complexity of our algorithm is $\mathcal{O}(s \times |K| \times s)$, which reduces to $\mathcal{O}(s^2 \times |K|)$.

However, our algorithm will never reach the complexity $\mathcal{O}(s^2 \times |K|)$, because the length of the plan and the number of requests are bound together. In the beginning, all taxi plans are empty and all requests are in a queue. We are trying to insert 1 request into some plan, but since all plans are empty we do only $|K|$ checks of length 1. Theoretically, the longest plan we can

Algorithm 3 Try schedule (P_k)

for all node index $j \in [0, |P_k| - 1]$ in vehicle plan P_k **do**
    a = current action at node $P_k[j]$
    if a is delivery OR (a is pickup AND taxi k can carry current passenger/parcel) **then**
        Let $time$ is the time to travel from $P_k[j]$ to $P_k[j+1]$
        Let $earlyTime = e_{P_k[j]} + time$
        Let $lateTime = l_{P_k[j]} + time$
        if $earlyTime \geq l_{P_k[j+1]}$ **then**
            Terminate - Schedule is not feasible
        end if
        $e_{P_k[j+1]} = maximum(earlyTime, e_{P_k[j+1]})$
        $l_{P_k[j+1]} = minimum(lateTime, l_{P_k[j+1]})$
    else
        Terminate - Schedule is not feasible
    end if
end for
Return new plan P_k

have is if all requests except the one we are currently checking were in one plan - it would have the length of $s - 1$. In that case, we would do only one check of the plan of length s, and at most $|K| - 1$ checks of a plan of length 1 (all other taxis would have empty plans). The last extreme is if we have all requests except the one equally distributed in all taxi plans. In this case, we are trying to insert 1 request into $|K|$ plans that have a length of some fraction of s. So the asymptotic complexity here is $\mathcal{O}(s \times |K|)$. The usual case will be something between these extremes.

Chapter 5

Solution - Improved method

In this chapter, we propose two improvements to the baseline method. As the first improvement, we allow more than one passenger to be onboard at the same time and we also allow interruptions of the passenger's ride. We expect the number of shared requests to increase, thus, reducing the mileage and the number of used vehicles, and also increasing the vehicle occupancy. As the second improvement, we implement the insertion heuristic [2]. The insertion heuristic (for Vehicle Routing Problem) takes every request and for every vehicle plan, it tries to insert the request on every plan position. This should utilize the vehicle fleet even better and further lower the total traveled distance and the number of vehicles needed because we search through many possible variants of one taxi plan.

5.1 Multiple passengers heuristic

In this section, we introduce the Multiple passengers heuristic. We decided to allow transportation of more than one passenger onboard at the same time. This means that we have to allow stops during the ride of a passenger so that another passenger could get in the taxi. When the passenger's ride can be interrupted because of another passenger, it makes sense to allow interrupting the passenger's ride to load and unload the package as well.

5.1.1 Selecting available taxis

Since some conditions are relaxed, we need to tweak the Algorithm 2 for selecting available vehicles. Now, for passenger requests, we can select as an available vehicle also the one that is currently carrying a passenger (or several passengers), but it must have a free seat. Similarly, for parcel requests, we can select a vehicle carrying a passenger, as his ride may be interrupted. The only condition is that there is enough free freight capacity. This modified algorithm is shown in Algorithm 4.

Algorithm 4 Select available taxis

```
1: Let K be the set of all taxis
2: K_a = ∅
3: for all taxi k ∈ K do
4:     if current request is of type person then
5:         if taxi k has sufficient capacity for the person then
6:             K_a += taxi k
7:         end if
8:     else
9:         if taxi k has sufficient capacity for the package then
10:            K_a += taxi k
11:        end if
12:    end if
13: end for
14: Return K_a
```

5.1.2 Checking validity of vehicle plan

Request actions are inserted into the vehicle plan the same way as in the base version. The algorithm for checking the plan's validity is the same as Algorithm 3 however the condition on line 3, determining the taxi's ability to perform the current action of its plan, is relaxed. When checking this, we do not need to check if there is a passenger onboard. If the action is a delivery, we can perform it without any limitations. If the action is a pickup, we check the sufficient capacity, either for a passenger or for a parcel.

5.2 Multiple passenger Insertion Heuristic

As the second improvement, we implemented the insertion heuristic [2]. Instead of inserting the new pickup and dropoff action at only one position in the current vehicle plan determined by the late time, we try to put the new actions at all possible positions in the current vehicle plan. The pseudocode for this method is shown in Algorithm 5. For a given request, we try to put the pickup action on every position in the current vehicle plan, and for each of these possibilities, we try to put the dropoff action on every position after the pickup action. On line 13 we proceed to the Algorithm 3 with the new P_k, however, when we check if taxi k can carry the current passenger/parcel, we use the check from the Multiple passenger heuristic 4 (i.e., we check only sufficient capacity).

5.3 Time complexity

In the Multi-passenger Insertion heuristic, we do the same procedure as in the Baseline heuristic but we do it for every combination of pickup action and

Algorithm 5 Insertion heuristic to find taxi plans

```
1: V' = ∅
2: Sort new requests incrementally by pickup lateTime into list V'
3: for all request r ∈ V' do
4:     K_a ← SelectAvailableTaxis()
5:     if K_a is not empty then
6:         c* = ∞
7:         for all taxis k ∈ K_a do
8:             P_k ← current plan of taxi k
9:             for all index i ∈< 0, |P_k| > do
10:                for all index j ∈< i + 1, |P_k| + 1 > do
11:                    Insert pickup action into P_k at index i
12:                    Insert drop-off action into P_k at index j
13:                    TrySchedule(P_k) with updated trajectory P_k
14:                    c_r^k = new total cost if taxi k serves request r using P_k
15:                    if c_r^k < c* then
16:                        c* = c_r^k
17:                        k* = k
18:                    end if
19:                end for
20:            end for
21:        end for
22:        if k* ≠ None then
23:            Insert request r into route of taxi k*
24:            Update the total cost
25:        else
26:            Reject request r
27:        end if
28:    else
29:        Reject request r
30:    end if
31: end for
```

dropoff action positions in the vehicle plan. The plan theoretically can have the length s (the number of all requests), so the Insertion heuristic increases the complexity by s^2. The resulting time complexity for the Multi-passenger Insertion heuristic is the time complexity of the Baseline heuristic multiplied by s^2: $\mathcal{O}(s^2 \times |K| \times s^2)$, which reduces to $\mathcal{O}(s^4 \times |K|)$.

Chapter 6

Implementation

We implemented both of the previously described methods in SiMoD [1]. SiMoD is a simulation tool for Mobility-on-Demand systems developed by the Smart Mobility group of AI Center, CTU in Prague. It is based on Agentpolis [2] traffic simulation framework. It is lightweight, highly customizable, and can run simulations with tens of thousands of vehicles and passengers with no problems. SiMoD and Agentpolis are both written in Java 11.

First, we implemented the solver of the baseline method on its own and tested it using unit tests. Then we integrated it into the simulation environment and measured the results. Finally, we implemented our improved solver and measured the performance.

6.1 Solvers implementation

The DARP solvers in SiMoD receive requests and create new vehicle plans. These plans are lists of pickup and dropoff actions that contain information about the passenger/parcel to which the action belongs, the position where the action takes place, and the maximum time in which the action must be completed. When implementing our solvers we had to extend SiMoD with package delivery functionality first because SiMoD was not capable of transporting anything else besides people.

First, we had to create a new class for a package. Second, we had to find a way to distinguish between passenger requests and package requests. We needed to be able to put both passenger and parcel requests in one list and differentiate them afterward. The parcel request also needed an extra parameter: parcel weight. Therefore we introduced new request classes (one for passenger and one for parcel) which inherit from the existing default request. Finally, we added new vehicles with package-handling functionality and separated capacity for passengers and freight. These vehicles also keep track of the combined vehicle occupancy, i.e., the number of both passengers and parcels currently carrying.

[1] `https://github.com/aicenter/simod`; this thesis is available on the branch `https://github.com/aicenter/simod/tree/blaham17_people_parcels_share`

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

After these modifications, we were able to implement our solver and test its correctness using unit tests.

6.2 Integration into SiMoD

The requests in SiMoD are loaded from a text file. We needed to make a new method for loading the package requests. When the requests are loaded, they are transformed into events and put into an event queue. Due to this, we needed to make a new type of event for a package. When the simulation time reaches the time of request origin, the event emerges from the event queue and it is handled by the `RidesharingDispatcher`. This class is responsible for handling new requests in SiMoD and triggering the calculation of the new plans. It receives the new requests, stores the waiting requests, and every 30 seconds the `replan()` method is called. This method first removes from waiting requests every request that has been waiting too long, i.e., the current simulation time exceeded the maximum pickup time of the request. Next, it calls the `solve()` method of our DARP solver with the new and waiting requests. Lastly, the Ridesharing Dispatcher class updates the vehicle plans of all taxis - it rewrites the current vehicle plans with the new ones obtained from the DARP solver. The `RidesharingDispatcher` also keeps track of the number of dropped requests. We had to modify it so it could handle also package requests and keep track of total shared requests (a request is considered shared if it shares part of or the whole ride with another request). Simplified class diagrams are shown in Figures 6.1 and 6.2.

6.3 SiMoD input and output

As an input, SiMoD takes a text file with trip records. On each line, there is one record with several parameters: request origin time, coordinates of request origin, and coordinates of request destination. For parcel request records we had to add a weight parameter. SiMoD has a console output, where there is periodically printed information about ongoing simulation and several statistics, e.g., the number of new requests for this period, the total number of dropped requests, or the total number of shared requests. SiMoD outputs these and other statistics (mileage, occupancy, etc.) also into CSV and JSON files.

6.4 Visualisation

When we run the simulation, the graphical user interface appears in a new window, as shown in Figure 6.3. We can zoom in and out and also move the map view. In the top left corner, we see the current simulation time in seconds, and in parentheses, we see the simulation speed which can be increased and decreased. We can also pause the simulation. There is also the framerate and the timestamp. In the bottom left corner is shown information

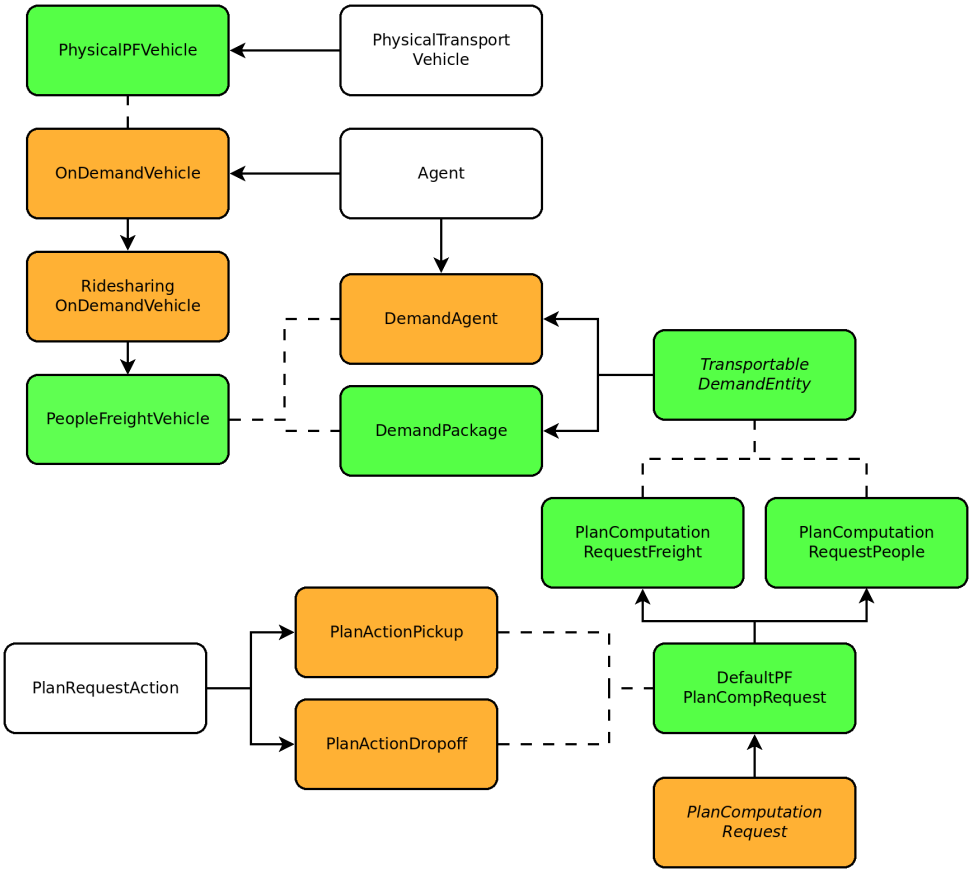


Figure 6.1: Simplified class diagram for vehicle and request classes. Newly implemented classes are colored green, modified classes are colored orange, and original classes are colored white. A name in italics indicates an interface. The arrow means that the class, it is pointing to, inherits from the other class. The dashed line indicates that the connected classes have some interaction between them.

about the current view. On the right side is a user interface for highlighting nodes or vehicles. Finally, at the bottom right corner is the interface for recording the simulation and toggling the viewed layers of the simulation. In Figure 6.4 we can see the running simulation.

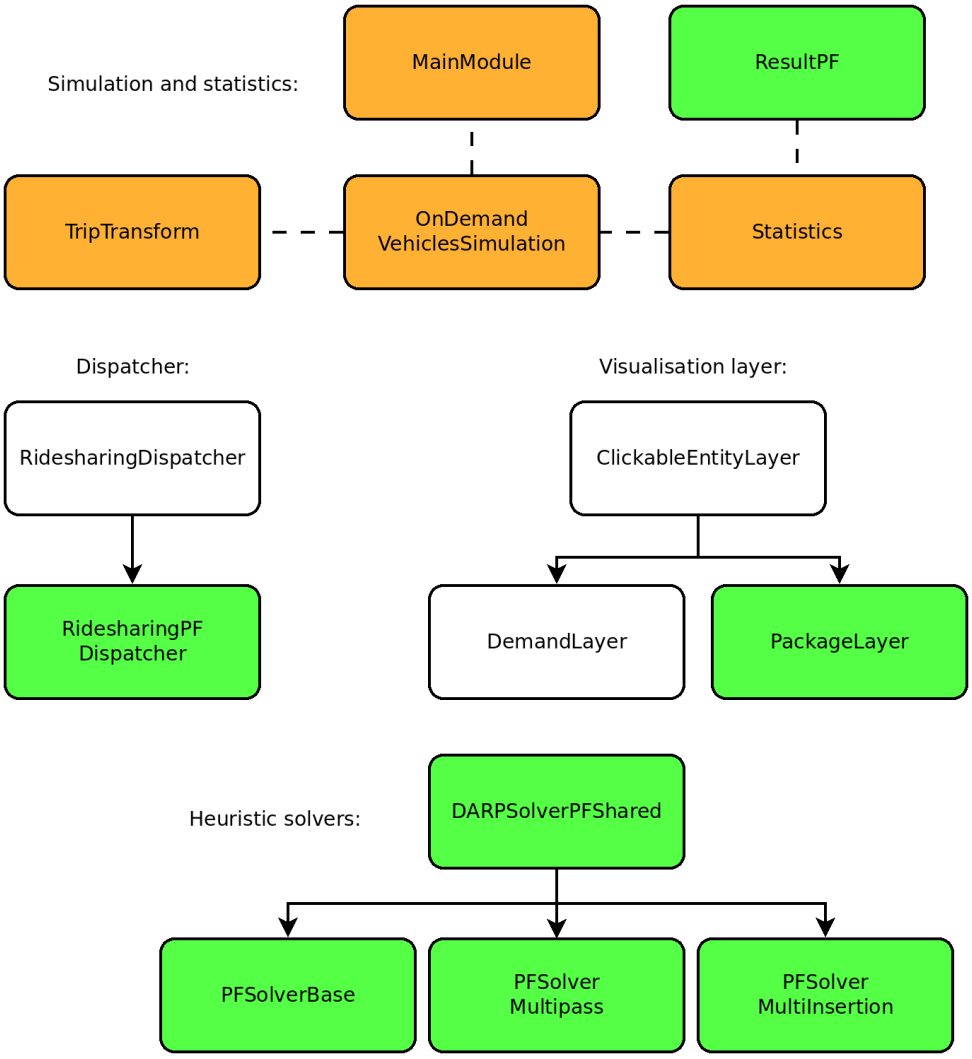


Figure 6.2: Simplified class diagram for main simulation-related classes, dispatcher, visualization layer, and heuristic solvers. Newly implemented classes are colored green, modified classes are colored orange, and original classes are colored white. The arrow means that the class, it is pointing to, inherits from the other class. The dashed line indicates that the connected classes have some interaction between them.

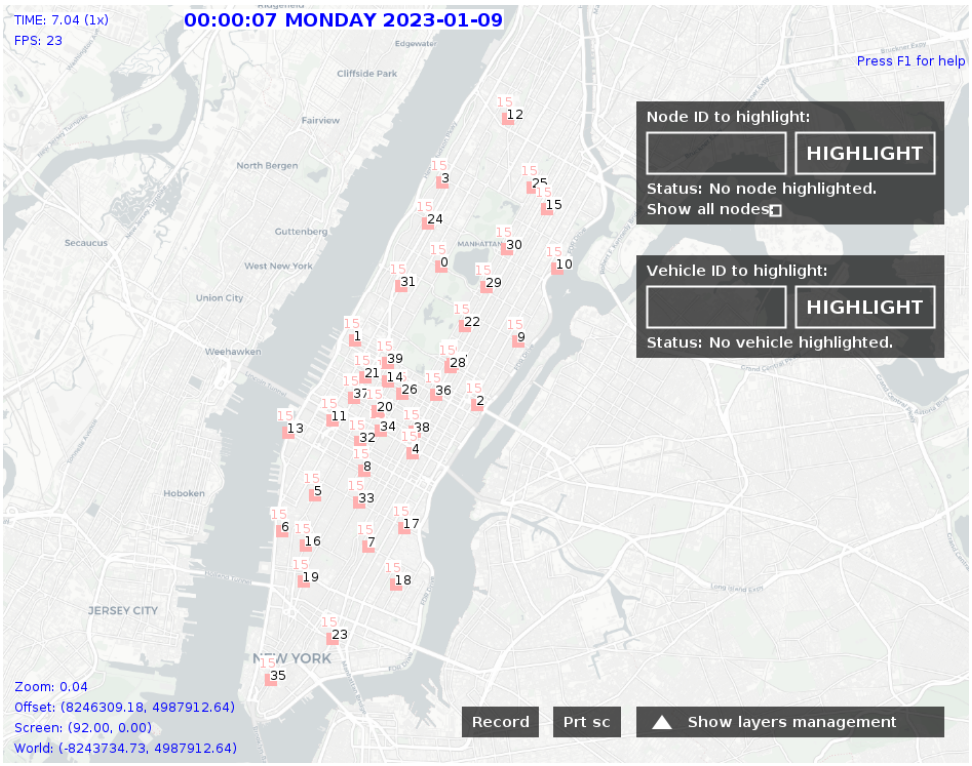


Figure 6.3: The simulation window at the start of an experiment

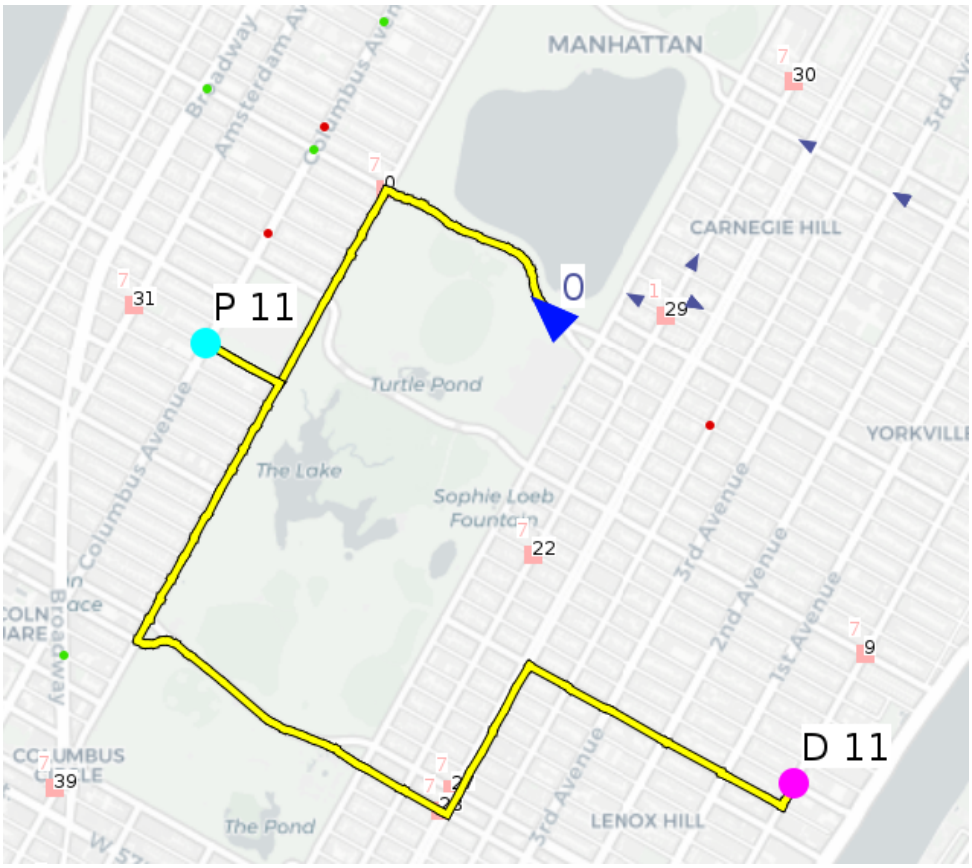


Figure 6.4: Running simulation in SiMoD with highlighted taxi. Red dots mark passenger requests and green dots mark parcel requests. Taxis are marked with blue triangles. The highlighted taxi also shows the number of passengers onboard (in this case 0). The current taxi plan is highlighted with yellow color. Pickup points on highlighted route are marked with turquoise dots and delivery points are marked with pink dots. These points also show an ID of the request they belong to. The pink squares mark the depots. Each depot has a numeric ID marked with a black number and the pink number indicates the number of taxis currently in the depot.

Chapter 7

Evaluation

In this chapter, we will describe the dataset and map we used for the evaluation of methods presented in Chapter 4 and Chapter 5. Then we compare the results of those methods.

7.1 Testing environment description

In this section, we describe the conditions under which we performed the experiments: the characteristics of demand, the road network used, the vehicle fleet, and the depots.

7.1.1 Demand data

For testing our heuristic methods we used 2014 Yellow Taxi Trip Data available on the NYC Open Data website [1]. This dataset contains 165 114 361 trips. To reduce the area for evaluation, we chose only requests which have both pickup and dropoff action located in Manhattan. We created datasets for three scenarios:

1. 24k requests in 24 hours
2. 25k requests in 1 hour
3. 50k requests in 1 hour

For all three scenarios, we selected a certain amount of trips from a usual weekday 12th of January. In the first scenario, we filtered the trips so that it was similar to the dataset in [10], i.e., on average 1000 requests per hour throughout the whole day. Therefore we divided the overall Manhattan demand from the 12th of January (360 000 requests) by 15 and got 24 000 requests within 24h. For the second scenario, we filtered requests between 0:00 and 1:00 which is 25 030 demands. For the third scenario, we filtered 50 415 requests from 13:00 to 16:00 and shifted their times to be within the 1-hour time span. Of these three amounts of requests, some were discarded after processing in SiMoD since they had zero length. Therefore the resulting numbers are lower by 150 to 300 requests. Demand histograms for every scenario can be seen in Figures 7.1, 7.2, and 7.3. In each of these scenarios,

[1] `https://data.cityofnewyork.us/Transportation/2014-Yellow-Taxi-Trip-Data/gn7m-em8n`

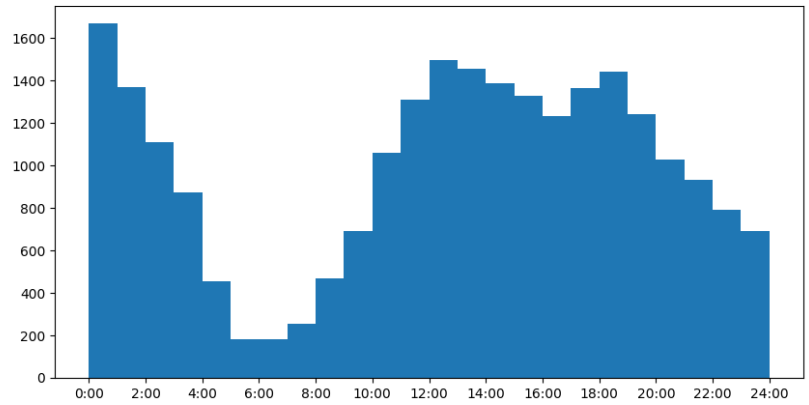


Figure 7.1: Demand histogram for the first scenario

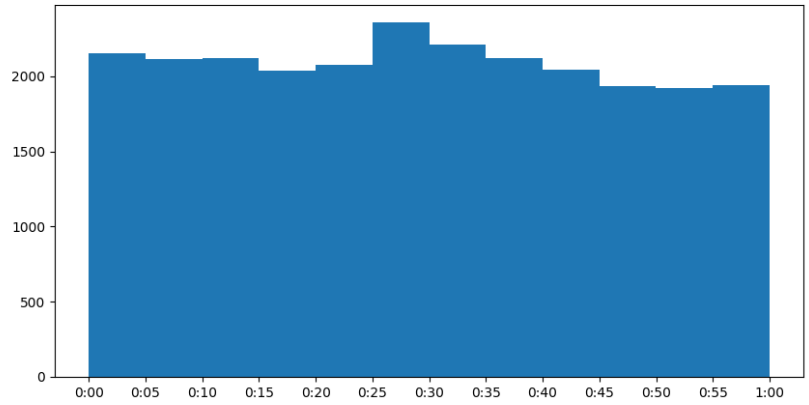


Figure 7.2: Demand histogram for the second scenario

randomly half of the requests were changed from people to parcels (every request had a probability of 0.5 to be changed to a parcel request) and they were assigned random weight values from set {5, 10, ..., 30}, where smaller packages were more common than bigger ones. Demand heatmaps are shown in Figure 7.6.

7.1.2 Road network

In our simulation environment, we used the area of Manhattan island and its surroundings. In terms of coordinates, it is an area with a latitude from N40.692274° to N40.83049° and a longitude from W74.065236° to W73.865596°. The graph representing this area has 13 512 nodes and 31 357 edges. Due to the demand distribution, almost all taxis do not leave Manhattan during the simulation, however exceptionally part of their route lies outside of Manhattan. The road network used in the simulation is shown in Figure 7.4.

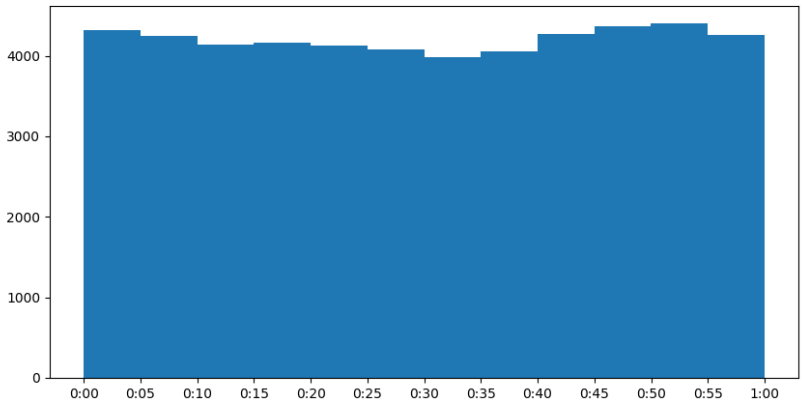


Figure 7.3: Demand histogram for the third scenario

7.1.3 Vehicles

For each scenario, we chose the number of vehicles proportional to the frequency of requests. To make the methods comparable, we set the number of vehicles such that the number of dropped demands was little to none for all methods. For the first scenario, it was 600 vehicles, for the second it was 5000 and for the third one, it was 9000. Each vehicle has a maximum capacity of 4 passengers and a maximum weight capacity of 100. Due to the weights of the packages, the vehicle can carry at most 3 largest packages (with a weight of 30) or at most 20 smallest ones (with a weight of 5). At the start of each experiment, the vehicles were equally distributed in 40 depots across Manhattan island. The map of depots is shown in Figure 7.5.

7.2 Results

In the three experiments described above, we measured several statistics, such as the total traveled distance, the number of used vehicles, the number of shared requests, etc. These statistics are shown for the first experiment in Table 7.2, for the second in Table 7.3, and for the third in Table 7.4. Served demands, dropped passenger demands, and dropped package demands together give the overall number of requests for the current experiment. Shared demands are the number of requests which were shared, i.e., for a part of its route or the whole route it was in a taxi together with at least one another request. The average computation time is the average time to compute vehicle plans for one batch of requests. The average delay is how much were the requests delayed on average in seconds.

From Table 7.2 we see that in the first experiment, the number of shared requests of the Multi-passenger heuristic is more than triple the baseline method, and the number of shared requests of the Multi-Insertion heuristic is almost 7 times more than the baseline method. However, the mileage of

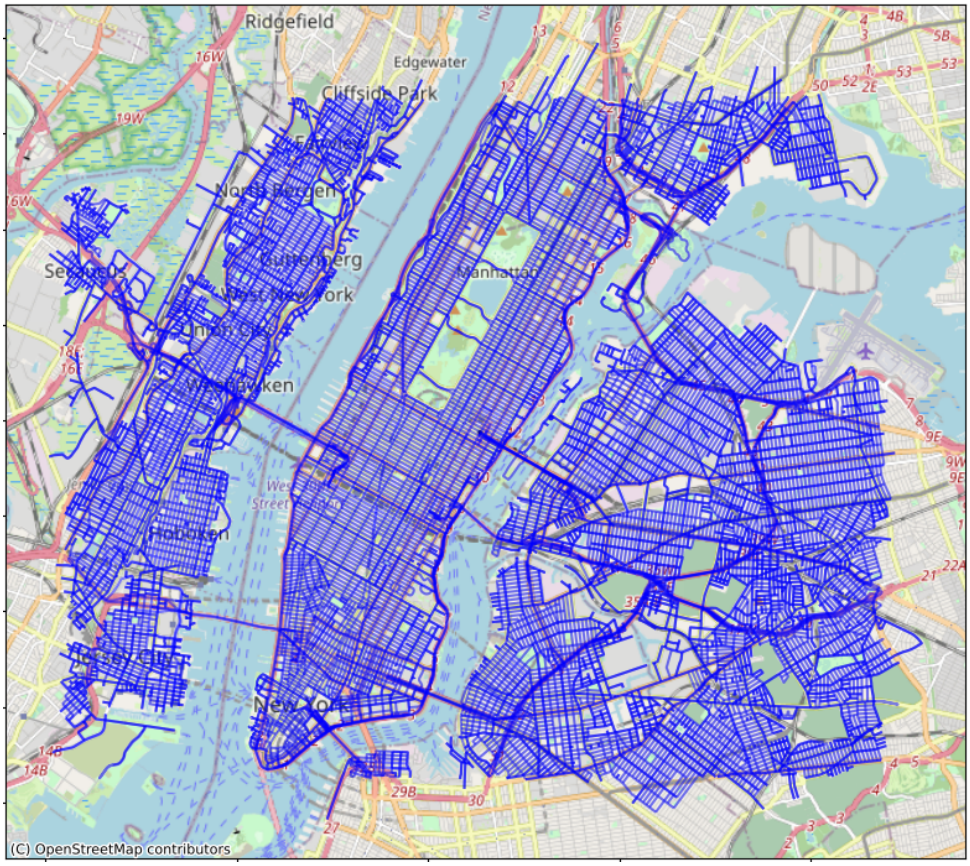


Figure 7.4: Used road network for Manhattan and surrounding areas

the Multi-passenger heuristic compared to the baseline method is lower by only 4 % and the mileage of the Multi-Insertion heuristic is lower by 11 %. In the number of used vehicles, the difference is only 4 % and 8 % respectively. The average computational time is zero because it was rounded down from less than 1 ms. In this experiment, our methods do not improve the overall efficiency significantly.

In the second experiment (Table 7.3), there was again only a small difference in the performance of the baseline heuristic and the Multi-passenger heuristic, except for the number of shared requests. However, in the results of the Multi-Insertion heuristic, we can see significant improvement. The number of shared requests is more than double compared to the baseline method - 83 % of all requests were shared. We used 16 % fewer vehicles and reduced the mileage by 19 %.

Table 7.4 shows the results of the third experiment. Here are the differences between the methods most significant. The percentage of shared requests out of all requests is 36 % for the baseline heuristic, 57 % for the Multi-passenger heuristic, and 90 % for the Multi-Insertion heuristic. Compared to the baseline method, the Multi-Insertion heuristic reduced the number of used vehicles by 28 % and the mileage by 40 %.

Along with the statistics in the Tables, we also measured vehicle occupancy,

	Baseline heuristic	**Multi passenger heuristic**	**Multi-Insertion heuristic**
Ridesharing rates *[%]*	7.7	11.8	23.71
Ridesharing with delivery rates *[%]*	5.2	6.3	13.2

Table 7.1: Ridesharing rates table for the 3rd experiment (50k requests in 1h)

which displays the effectiveness of ridesharing. We measured the occupancy of passengers and parcels combined, otherwise, we would not see the dependence of these two quantities. Combined occupancy 3D histograms for the first, the second, and the third experiment can be seen in Figures 7.7, 7.8, and 7.9 respectively.

In the first two experiments, the differences between methods in terms of mileage and the number of used vehicles were minor. The only significant difference was in the number of shared requests. The Multi-Insertion heuristic had 41 % shared requests in the first experiment and 83 % in the second one. However, when we look at the occupancy histograms, the ridesharing rates are only 5 % and 18 %. And the ridesharing with delivery rates are even lower: less than 5 % in the first experiment, and only 11 % in the second experiment. For the other methods, the ridesharing with delivery rates in the first experiment were negligible. In the second experiment, however, the Baseline heuristic and the Multi passenger heuristic had ridesharing with delivery rates around 7.5 %. The number of shared requests is quite deceiving because it does not say anything about how efficiently were the vehicles used in terms of occupancy. In terms of vacant taxis, in the first experiment, all methods performed similarly. In the second experiment, the Multi-Insertion heuristic was 10 percentage points better than the other methods.

In the third experiment, the differences between the three methods were most significant. The Multi-Insertion heuristic reduced the mileage by 40 % and used 28 % fewer vehicles, compared to the Baseline heuristic. Shared demand rates for the Baseline heuristic, Multi passenger heuristic, and Multi-Insertion heuristic were 36 %, 57 %, and 90 % respectively. However, the ridesharing rates are again much lower. They are shown in Table 7.1. The vehicle vacancy rates in the third experiment are over 40 % for the Baseline method, over 35 % for the Multi-passenger method, and 25 % for the Multi-Insertion method.

From the tables and occupancy histograms, we see that in all three experiments, our improved method performed out of the three methods the best in terms of all measured statistics (except for the computational time due to greater time complexity). The difference in efficiency of the solution was more significant in bigger instances with greater demand density. It met our expectations and outperformed the Baseline method in terms of fleet utilization. However, it could still be improved in terms of ridesharing rates.

	Baseline heuristic	**Multi passenger heuristic**	**Multi-Insertion heuristic**
Total veh. distance *[km]*	**333 601**	**321 677**	**297 733**
Served demands	23 764	23 830	23 831
Dropped passenger demands	51	10	13
Dropped package demands	36	11	7
Shared demands	1417	4317	9880
Used vehicles	475	458	437
Avg. comp. time *[ms]*	0	0	0
Avg. delay *[s]*	385	389	381

Table 7.2: Methods comparison table for the 1st experiment (24k requests in 24h)

	Baseline heuristic	**Multi passenger heuristic**	**Multi-Insertion heuristic**
Total veh. distance *[km]*	**193 415**	**189 864**	**157 860**
Served demands	24 827	24 873	24 879
Dropped passenger demands	40	5	3
Dropped package demands	17	6	2
Shared demands	9641	14 099	20 676
Used vehicles	5000	5000	4195
Avg. comp. time *[ms]*	3	4	4
Avg. delay *[s]*	421	405	376

Table 7.3: Methods comparison table for the 2nd experiment (25k requests in 1h)

	Baseline heuristic	**Multi passenger heuristic**	**Multi-Insertion heuristic**
Total veh. distance *[km]*	**362 231**	**342 511**	**251 109**
Served demands	50 116	50 119	50 120
Dropped passenger demands	5	2	1
Dropped package demands	3	3	3
Shared demands	18 246	28 752	45 112
Used vehicles	9000	8671	6477
Avg. comp. time *[ms]*	9	11	11
Avg. delay *[s]*	387	385	353

Table 7.4: Methods comparison for the 3rd experiment (50k requests in 1h)

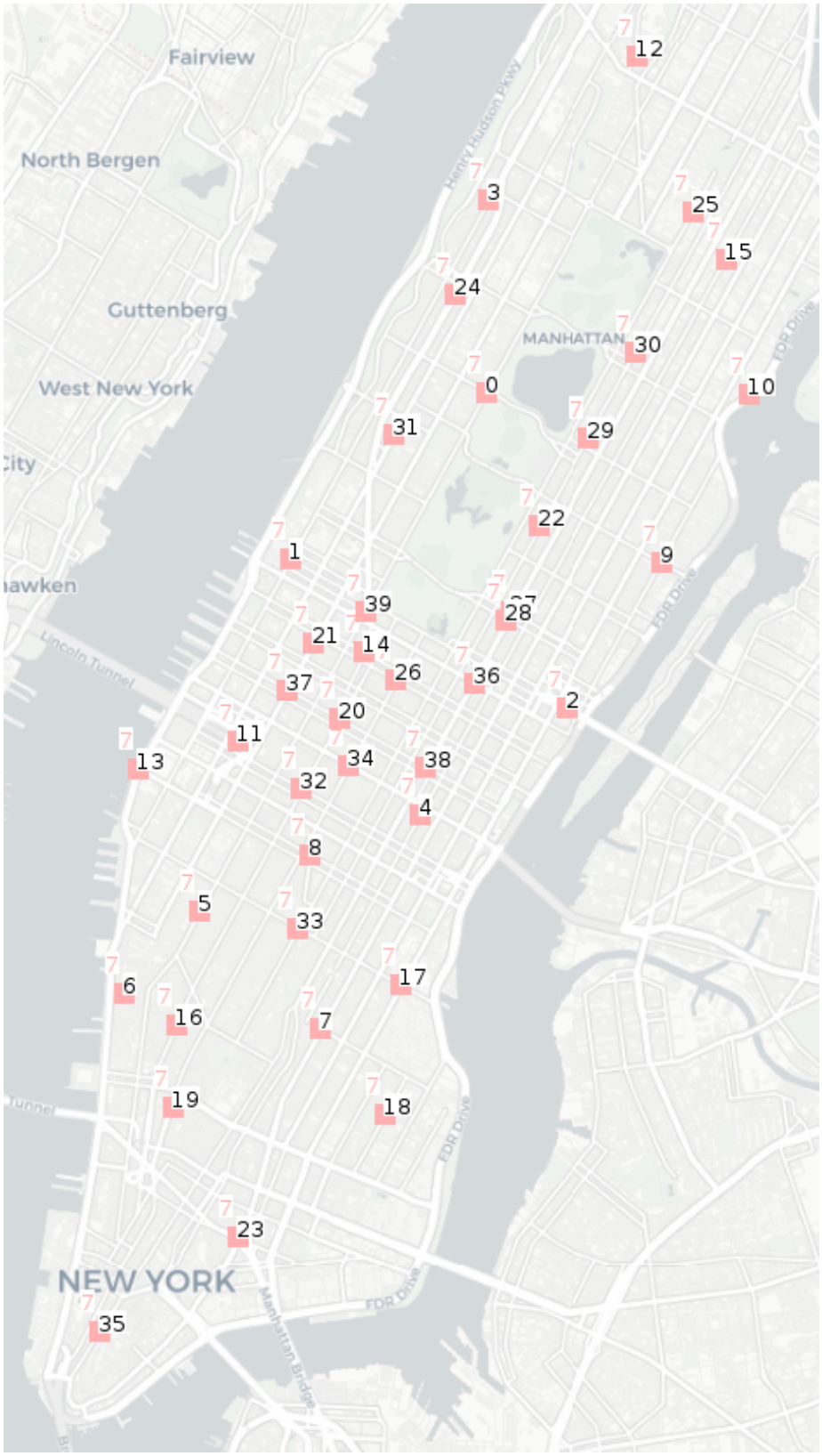
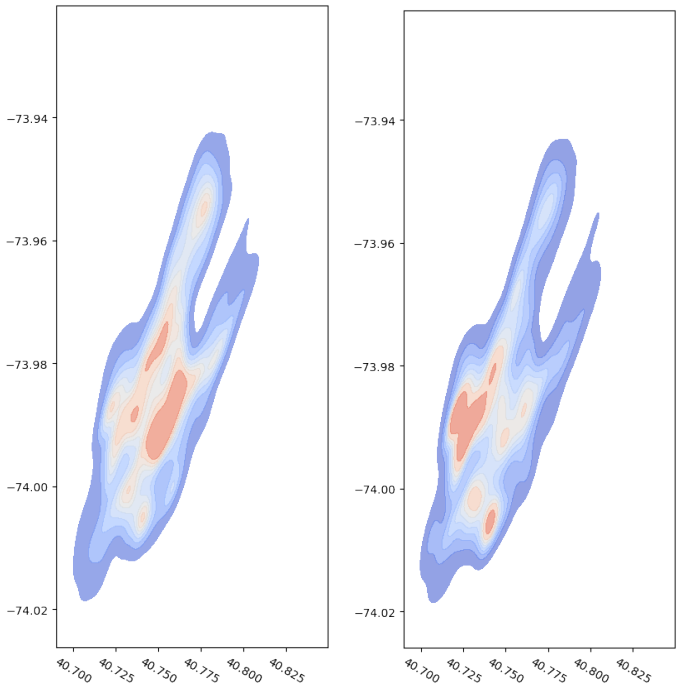
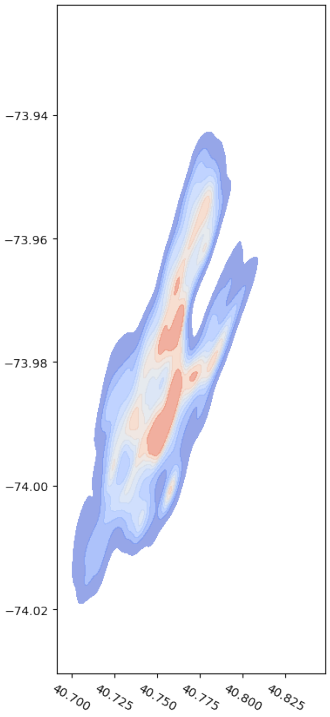


Figure 7.5: Map of the depots

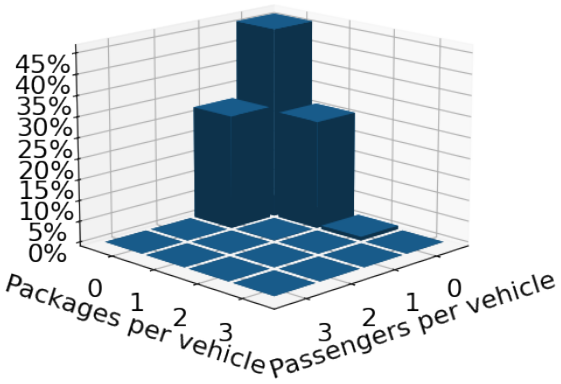


(a) : The first scenario **(b) :** The second scenario

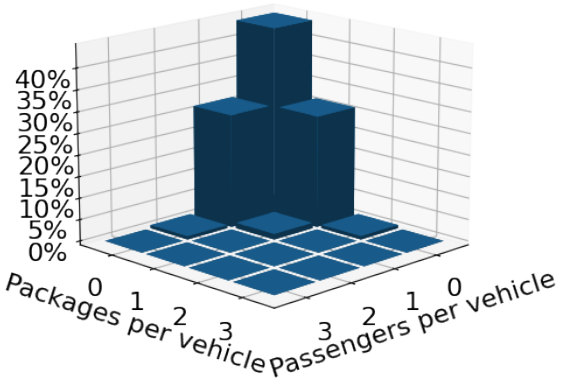


(c) : The third scenario

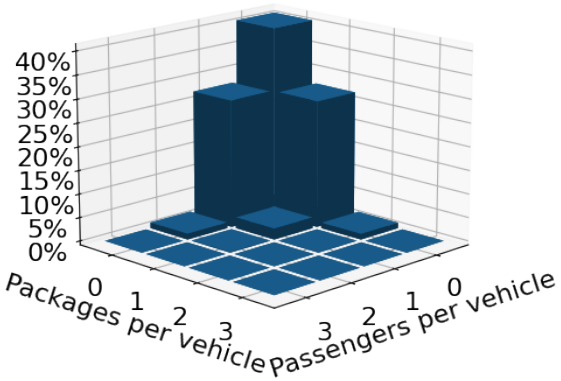
Figure 7.6: Demand heatmaps



(a) : Baseline

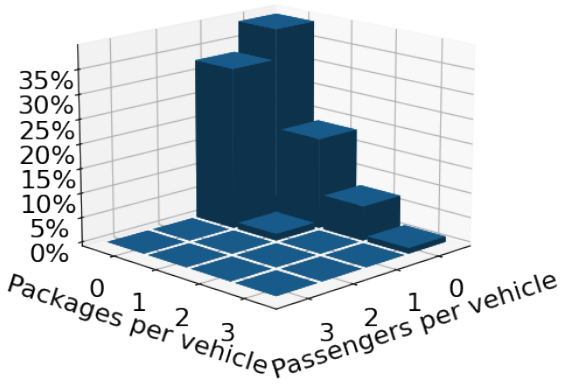


(b) : Multi passenger

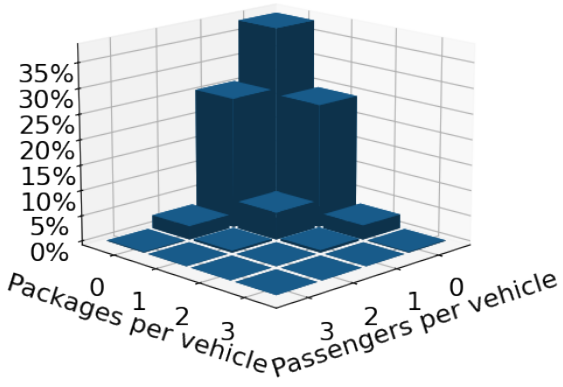


(c) : Multi-Insertion

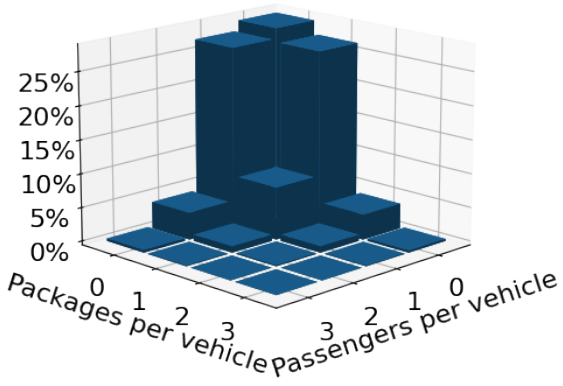
Figure 7.7: Combined occupancy for the first scenario. The vertical axis shows the percentage of the time for the occupancies.



(a) : Baseline

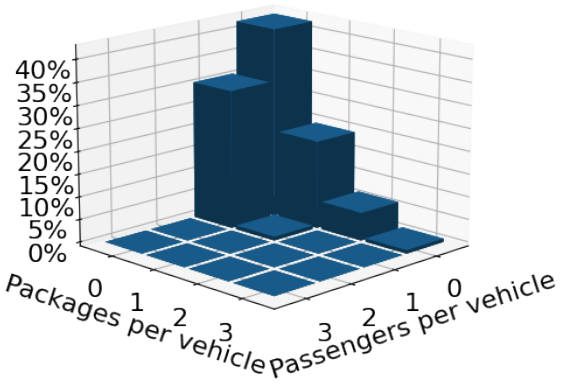


(b) : Multi passenger

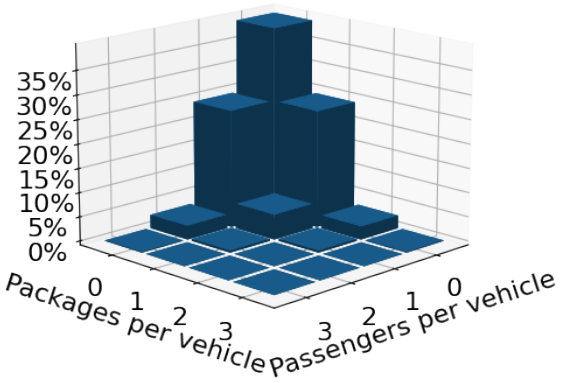


(c) : Multi-Insertion

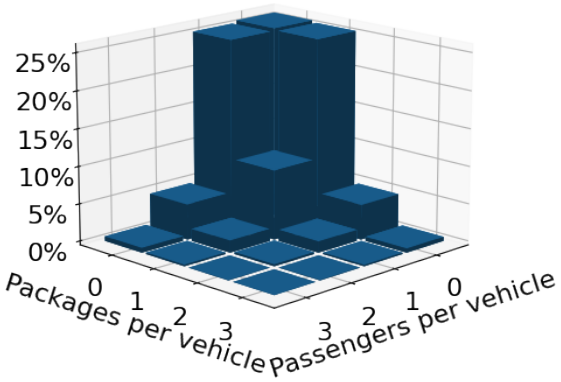
Figure 7.8: Combined occupancy for the second scenario. The vertical axis shows the percentage of the time for the occupancies.



(a) : Baseline



(b) : Multi passenger



(c) : Multi-Insertion

Figure 7.9: Combined occupancy for the third scenario. The vertical axis shows the percentage of the time for the occupancies.

Chapter 8

Conclusion

In the last few years, the demand for urban transport is increasing, and the number of vehicles in cities is rising. Ridesharing has a big potential since it could solve the problems arising from too many cars occupying urban streets, like excessive air pollution, traffic jams, and a lack of parking places. There are two vehicle transport services in cities nowadays: taxi and delivery service. But these two operate separately.

In this work, we tried to improve the efficiency of taxi services and delivery services in urban areas by combining these two together. First, we studied the literature on the topic of ridesharing and package delivery and defined this problem mathematically in Chapter 3. Then we implemented an existing method described in Chapter 4 into SiMoD. Next, we proposed some improvements to this method in Chapter 5. We also implemented it and for this, we needed to create several new classes and methods in SiMoD, and modify some of the existing classes, as described in Chapter 6. Finally, we measured the performances of all these methods and compared them in Chapter 7. We successfully created an improved method, which outperformed the baseline method in terms of fleet utilization. In future work, we could improve the low vehicle occupancy. By modifying the available-taxi selecting algorithm, we could for example prefer to select occupied taxis, therefore increasing the ridesharing rates.

Bibliography

[1] Breno A. Beirigo, Frederik Schulte, and Rudy R. Negenborn. Integrating people and freight transportation using shared autonomous vehicles with compartments. *IFAC-PapersOnLine*, 51(9):392–397, 2018. 15th IFAC Symposium on Control in Transportation Systems CTS 2018.

[2] Ann Campbell and Martin Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38:369–378, 08 2004.

[3] Federico Cavallaro and Silvio Nocera. Integration of passenger and freight transport: A concept-centric literature review. *Research in Transportation Business Management*, 43:100718, 2022.

[4] Angela Di Febbraro, Davide Giglio, and Nicola Sacco. On exploiting ride-sharing and crowd-shipping schemes within the physical internet framework. pages 1493–1500, 2018.

[5] Phan-Thuan Do, Nguyen-Viet-Dung Nghiem, Ngoc-Quang Nguyen, and Quang-Dung Pham. A time-dependent model with speed windows for share-a-ride problems: A case study for tokyo transportation. *Data Knowledge Engineering*, 114:67–85, 2018. Special Issue on Knowledge and Systems Engineering (KSE 2016).

[6] Ezzeddine Fatnassi, Jouhaina Chaouachi, and Walid Klibi. Planning and operating a shared goods and passengers on-demand rapid transit system for sustainable city-logistics. *Transportation Research Part B: Methodological*, 81:440–460, 2015. Optimization of Urban Transportation Service Networks.

[7] Baoxiang Li, Dmitry Krushinsky, Hajo A. Reijers, and Tom Van Woensel. The share-a-ride problem: People and parcels sharing taxis. *European Journal of Operational Research*, 238(1):31–40, 2014.

[8] Kaushik Manchella, Abhishek K. Umrawal, and Vaneet Aggarwal. Flexpool: A distributed model-free deep reinforcement learning algorithm for joint passengers and goods transportation. *IEEE Transactions on Intelligent Transportation Systems*, 22(4):2035–2047, 2021.

[9] Abood Mourad, Jakob Puchinger, and Tom Van Woensel. Integrating autonomous delivery service into a passenger transportation system. *International Journal of Production Research*, 59, 04 2020.

[10] Ngoc-Quang Nguyen, Nguyen-Viet-Dung Nghiem, Phan-Thuan DO, Khac-Tuan LE, Minh-Son Nguyen, and Naoto MUKAI. People and parcels sharing a taxi for tokyo city. page 90–97, 2015.

[11] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[12] C. Pimentel and F. Alvelos. Integrated urban freight logistics combining passenger and freight flows - mathematical model proposal. 30:80–89, 2018. Cited By :31.

[13] Teng Ren, Zhuo Jiang, Xiangyu Cai, Yongzhuo Yu, Lining Xing, Yuan Zhuang, and Zhenping Li. A dynamic routing optimization problem considering joint delivery of passengers and parcels. *Neural Computing and Applications*, 33:1–12, 08 2021.

[14] Max van der Tholen, Breno A. Beirigo, Jovana Jovanova, and Frederik Schulte. The share-a-ride problem with integrated routing and design decisions: The case of mixed-purpose shared autonomous vehicles. pages 347–361, 2021.

Appendix A

Attachment content

- `packages_manhattan_24k_24h.txt` packages demand for the 1st scenario
- `packages_manhattan_25k_1h.txt` packages demand for the 2nd scenario
- `packages_manhattan_50k_1h.txt` packages demand for the 3rd scenario
- `people_manhattan_24k_24h.txt` passengers demand for the 1st scenario
- `people_manhattan_25k_1h.txt` passengers demand for the 2nd scenario
- `people_manhattan_50k_1h.txt` passengers demand for the 3rd scenario